Memórias do WTA 2016

X 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
2016

Ficha catalográfica

Workshop de Tecnologia Adaptativa (10: 2016: São Paulo)
Memórias do WTA 2016. – São Paulo; EPUSP, 2016. 103p.

ISBN 978-85-86686-86-3

Agência Brasileira do ISBN
ISBN 978-85-86686-86-3

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 décima edição do Workshop de Tecnologia Adaptativa realizou-se em São Paulo, Brasil, nos dias 28 e 29 de Janeiro de 2016, nas dependências da Escola Politécnica da Universidade de São Paulo. As contribuições encaminhadas na forma de artigos relacionados à Tecnologia Adaptativa, nas seguintes áreas, abrangeram, de forma não exclusiva, os tópicos abaixo:

Fundamentos da Adaptatividade

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

Tecnologia Adaptativa – Técnicas, Métodos e Ferramentas

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

Aplicações da Adaptatividade e da Tecnologia Adaptativa

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

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

– Inferência, reconhecimento e classificação de padrões;

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

– Simulação, arte por computador e jogos eletrônicos inteligentes;

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

Comissão de Programa

Comissão Organizadora

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

Apoio

- Escola Politécnica da Universidade de São Paulo (EPUSP)
- FAPESP – Fundação de Amparo à Pesquisa do Estado de São Paulo, processo número 2015/24114-0
- IEEE – Institute of Electrical and Electronics Engineers
- METMAT – Departamento de Engenharia Metalúrgica e de Materiais da EPUSP
- Olos Tecnologia e Sistemas S.A.
- Pagga Tecnologia de Pagamentos Ltda.
- PCS – Departamento de Engenharia de Computação e Sistemas Digitais da EPUSP
- São Paulo Convention & Visitors Bureau
- SBC – Sociedade Brasileira de Computação
- SPC – Sociedad Peruana de Computación
- Universidade de São Paulo

Memórias do WTA 2016

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 2016, o Décimo Workshop de Tecnologia Adaptativa, realizado em São Paulo nos dias 28 e 29 de Janeiro de 2016. A exemplo da edição de 2015, 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 e textos técnicos apresentados no WTA 2016 – Décimo Workshop de Tecnologia Adaptativa, realizado na Escola Politécnica da Universidade de São Paulo nos dias 28 e 29 de Janeiro de 2016.

Esta edição contou com 20 trabalhos, entre comunicações técnicas, textos técnicos e artigos, relacionados à área de Tecnologia Adaptativa e aplicações nos mais diversos segmentos. Adicionalmente, foram ministrados dois tutoriais, a saber: *Proposta de implementação adaptativa de programas complexos*, por João José Neto, e *Aplicação de técnicas adaptativas na manipulação de textos*, por Paulo Roberto Massa Cereda. O evento foi registrou uma participação efetiva de uma centena de pesquisadores durante os dois dias, constatando-se o sucesso do mesmo.

Esta publicação

Estas memórias espelham o conteúdo apresentado no WTA 2016. A exemplo do que foi feito no ano anterior, todo o material referente aos trabalhos apresentados no evento estará acessível no portal do WTA 2016, 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 2017.

Agradecimentos

Às instituições que apoiaram o WTA 2016, 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

- Ákio Nogueira Barbosa
- Leia Sicília
- Maria Cristina Biasoli
- Newton Kiyotaka Miura
- Nilton Araújo do Carmo
- Rodrigo Kavakama
- Suellen Alves

Apoio operacional

- Daniel Costa Ferreira
- Edson de Souza

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 décima edição do Workshop de Tecnologia Adaptativa. Também agradecemos ao Departamento de Engenharia Metalúrgica e de Materiais pela disponibilização do anfiteatro para a realização do evento e à Fundação de Amparo à Pesquisa do Estado de São Paulo pelo apoio financeiro. Por fim, agradecemos aos engenheiros Newton Kiyotaka Miura e Paulo Roberto Massa Cereda pelo valioso auxílio na organização.

São Paulo, 29 de Janeiro de 2016

João José Neto
Coordenador geral

Processo número 2015/24114-0, Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP)

As opiniões, hipóteses e conclusões ou recomendações expressas neste material são de responsabilidade do(s) autor(es) e não necessariamente refletem a visão da FAPESP.

Mensagens

As seguintes mensagens foram enviadas por Margarete Keiko Iwai, Joel Camargo Dias Pereira e Cinthia Itiki para registrar suas participações na décima edição do Workshop de Tecnologia Adaptativa.

Margarete Keiko Iwai

Saudações a todos os participantes da décima edição do Workshop de Tecnologia Adaptativa.

Tenho o prazer de dirigir-me a todos os membros do Workshop para agradecer o convite para participar deste evento de aniversário de comemoração do laboratório, em especial ao meu mestre e querido amigo Prof. João José Neto, a quem tenho muito respeito, carinho e admiração.

Gostaria de parabenizar a todos os membros do Laboratório de Tecnologias Adaptativas pelo excelente trabalho sendo desenvolvido em todos esses anos dedicados à pesquisa.

Tenho orgulho em dizer que tive uma pequena contribuição inicial nos trabalhos relacionados a esta área.

Minha experiência como pós-graduanda foi extremamente enriquecedora. Ajudou-me a ser uma pessoa mais determinada e uma profissional com um perfil independente e crítico.

Para todos os interessados em fazer um curso de mestrado ou doutorado, gostaria de passar algumas palavras de incentivo baseado em minha experiência pessoal.

Embora a carreira em pesquisa científica seja pouco atrativa para a maioria da população, temos inúmeros exemplos de trabalhos que inicialmente poderiam ser considerados impraticáveis ou desnecessários, mas no decorrer da história, provaram ser de extrema utilidade.

As tecnologias adaptativas são um bom exemplo, pois podemos considerar alguns produtos comerciais existentes atualmente que empregam o conceito semelhante de adaptabilidade.

Nos primórdios do desenvolvimento de software, os programadores utilizavam apenas um editor de texto para escrever os programas, compilavam e executavam através de comandos executados diretamente no sistema operacional.

Os ambientes de desenvolvimento de software evoluíram de forma a facilitar o trabalho de edição, compilação e execução. Esses aplicativos permitem que, à medida que os comandos são editados, ele seja auto completável ou apresente sugestões para possíveis comandos, dirigidos pela sintaxe da linguagem.

Verifica-se que os trabalhos científicos podem gerar produtos tecnológicos práticos que podem ser utilizadas por todas as pessoas e que, sendo utilizado de forma consciente, podem trazer o progresso da humanidade.

Por esses motivos e outros mais, gostaria de incentivar os futuros e atuais pesquisadores a continuarem se dedicarem à pesquisa e ao estudo, pois é disso que todos precisam, de pessoas dedicadas e criativas que fazem a diferença.

Desejo muito sucesso a todos. Muito obrigada pela oportunidade e pela atenção.

Margarete Keiko Iwai

Joel Camargo Dias Pereira

Olá, professor,

Fazendo uma reflexão dos acontecimentos deste ano de 2015, recordei com muito carinho e gratidão o tempo do meu mestrado. Queria aproveitar a ocasião para desejar um Natal e Ano Novo abençoado abundantemente para toda sua família e para você também. Que o WTA deste ano seja um grande sucesso. E que em 2016 a gente consiga se reencontrar, gostaria de novamente apertar sua mão e agradecê-lo pelo investimento de tempo e dedicação durante o meu programa de mestrado.

Meu sincero Muito Obrigado.

Abraços,

Joel Camargo Dias Pereira

Cinthia Itiki

Em 2004, eu e o Prof. João José Neto publicamos um artigo sobre uma aplicação inusitada dos conceitos de programação adaptativa (C. Itiki; J. José Neto. *Complete automation of the generalized inverse method for constrained mechanical systems of particles*, Applied Mathematics and Computation, Amsterdã, vol. 152, pp.561-580, 2004).

Foram geradas rotinas que implementavam automaticamente a solução de Udwadia-Kalaba para sistemas mecânicos vinculados, no ambiente de programação do Matlab.

Apesar de não ter mais atuado diretamente na área, tenho acompanhado a evolução dos trabalhos ao longo dos 10 anos do WTA. É com alegria que observo o crescimento das aplicações para os mais diversos ramos: processamento de sinais, reconhecimento de padrões, linguagens naturais, análise de sentimentos e implementação de programas complexos.

É rica a trajetória passada e inspiradora para o caminho ainda a percorrer. Vida longa ao WTA!

Cinthia Itiki

Sumário

Lista de autores

Parte I

Comunicações técnicas

Uma Arquitetura Adaptativa

S.S. Barretto and J. J. Neto

***Abstract*— We try to modify an existing CPU architecture to include adaptivity. The chosen architecture, the "Patinho Feio", was the first computer developed in Brazil. The authors list the reasons for this choice, the changes made to implement adaptivity and future plans.**

***Keywords*— Adaptivity, Adaptive Devices. Architecture**

I. INTRODUÇÃO

O computador Patinho Feio foi desenvolvido na Escola Politécnica da USP no início da década de 70 por alunos do curso de pós graduação e foi utilizado pelos autores para aprendizado no desenvolvimento de software básico.

Em um reencontro recente dos autores foi apresentada uma nova "versão" do Patinho Feio, agora no formato de uma APP desenvolvida para dispositivos móveis da Apple.

Surgiu então a idéia de utilizar essa APP como base de estudo para o desenvolvimento de uma arquitetura adaptativa.

Entre os fatores que levaram a essa escolha podemos citar:

- Pelo fato de ser totalmente feita por "software", permite fácil modificação, com o único custo sendo o tempo gasto no seu desenvolvimento.

- Por se tratar de uma arquitetura razoavelmente simples, qualquer modificação pode ser feita em pequeno espaço de tempo.

- Apesar de se tratar de um projeto acadêmico, existe farta documentação sobre o Patinho Feio, tanto no formato de teses como em livro [1][2].

- Do mesmo modo existe uma razoavel quantidade de "sofware" básico desenvolvido para ele, como montador, compilador, ligador e carregador.

II. DESENVOLVIMENTO

Quando se fala em arquitetura adaptativa estamos nos referindo a uma arquitetura que possa ser alterada por meio de programas.

Embora a existência de hardware com instruções que possam ser modificadas por programa não seja exatamente um assunto novo[3], não existe muita literatura a respeito de arquiteturas adaptativas.

Existe pelo menos uma proposta sobre uma linguagem de alto nível para facilitar o desenvolvimento de programas automodificáveis[4].

Para o desenvolvimento, fizemos algumas premissas básicas:

- A arquitetura adaptativa deveria ser compatível com os softwares existentes, permitindo que eles pudessem ser executados sem adaptação.

- A indicação de que se trata de uma instrução adaptativa seria realizada por meio de um prefixo na instrução.

- O código que trata as instruções adaptativas seria formado por instruções similares àquelas que compõem o conjunto de instruções do Patinho Feio.

- A área de memória reservada ao código adaptativo deveria ficar separada da área de memória reservada aos outros programas.

Baseado nessas premissas, escolhemos o código de operação 9F, que na primeira versão do Patinho Feio era uma instrução reservada, sem operação associada, como indicador de adaptatividade.

Posteriormente foi lembrado que, embora a versão 1.0 do Patinho Feio tivesse apenas 4k bytes de memória, foi feita uma segunda versão com 32k bytes, onde o acesso às posições de memória além dos 4k da página atual era realizado por meio de endereçamento indireto.

Para indicar que uma instrução de acesso à memória tinha endereçamento indireto era usado o prefixo 9F, e o endereço de memória (de 12 bits) da instrução indicava a área de 16 bits onde estava o endereço real de memória, sendo o bit mais significativo reservado para indicar níveis adicionais de indireto.

Assim, para manter compatibilidade com a versão 1.0 e 2.0, resolvemos criar uma versão 3.0, onde podem existir vários modos de operação, que são selecionados por uma combinação do código 9F com instruções que são raramente usadas, como operações lógicas do acumulador com dados lidos do registrador de chaves do painel.

As combinações escolhidas foram:

- 9F8A : seleciona modo de operação 1.0
- 9F8B : seleciona modo de operação 2.0
- 9F8C : seleciona modo de operação 3.0
- 9F8D: se estiver no modo 3.0, esta instrução funciona como o prefixo de indireto da versão 2.0, com a diferença que o bit mais significativo é um bit de endereço, permitindo o acesso a 64k bytes de memória.
- 9F9F : este é o prefixo que indica que a próxima instrução e uma instrução adaptativa.
- se a instrução 9F for seguida por outro valor, ela opera como no modo 2.0, permitindo que os programas tenham acesso a qualquer posição dos primeiros 32k bytes de memória.

Na arquitetura proposta os programas de usuário e sistema operacional ficariam limitados aos primeiros 32k de memória e a área acima disso seria reservada ao código adaptativo.

A execução de uma instrução adaptativa (desencadeada pela sequência 9F9F) provocaria o seguinte sequência de eventos:

- a interrupção é inibida (independente da interrupção já

estar inibida ou não, e sem alterar essa condição).

- a instrução a ser executada é armazenada no endereço 8002 (hexa), 8003 se longa e 8004 se indireta.
- o endereço de retorno (próxima instrução) é armazenado nas posições 8006 e 8007 e é executado o código adaptativo armazenado a partir da posição 8008.
- o código adaptativo pode decidir por executar operações pré-adaptativas, ou seja, operações executadas antes da execução da instrução atual, decidir se executa ou não a instrução atual e se executa ou não operações pós-adaptativas.
- o código adaptativo pode inclusive optar por se automodificar, sendo que a único ponto a ser considerado nesse caso, é que todo esse processamento é efetuado com interrupção inibida, podendo causar problemas em alguma operação de entrada ou saida que esteja em andamento, caso esse processamento seja muito longo.
- durante a execução do código adaptativo algumas instruções não podem ser executadas e portanto serão ignoradas se forem encontradas, como as instruções que habilitam e inibem interrupção, as instruções que seecionam o modo de operação e a que indica uma instrução adaptativa.
- o retorno do código adaptativo acontece quando é feito um desvio indireto ao endereço 8006. Após o desvio, e antes de executar a próxima instrução, é desligado o estado de adaptatividade, e restaurado o estado de habilitação da interrupção.

No estado atual estamos implementando o código nas APPs para iPhone e iPad e procurando uma aplicação para testar o conceito. Uma provável aplicação seria a implementação do problema do coletor de nomes[5], uma aplicação adaptativa já bem estudada.

Para o futuro, há a perspectiva de gerar um modelo mais real, com a construção de um painel com leds e chaves controlados por uma placa raspberry pi, ou equivalente, ao qual poderiam ser conectados periféricos reais, como impressoras ou terminais de vídeo.

III. CONCLUSÃO

A emulação de arquiteturas por meio de APPs fornece um meio rápido e barato de realizar o teste de arquiteturas adaptativas ou experimentais.

A utilização de uma arquitetura conhecida como ponto de partida permite que se utilize ferramentas já existentes no auxílio ao desenvolvimento de software.

No desenvolvimento procuramos manter compatibilidade com o software já existente, e estender a arquitetura de modo a ter uma ampla gama de possibilidades de criação de novas instruções adaptativas.

Para testar o conceito, estão sendo modificadas APPs de iPhone e iPad, para incluir a adaptatividade.

Para o futuro pretende-se também implementar o conceito num modelo mais real, com um painel de leds e uma placa processadora onde poderão ser conectados periféricos reais.

REFERÊNCIAS

[1] JUNIOR, G.L.; FREGNI, E. “Projeto de Computadores Digitais” Editora Edgard Blucher, Editora da Universidade de São Paulo , cap 5, 1974

[2] NETO J.J “Aspectos do Projeto de Software de um Minicomputador” Dissertação de mestrado na Escola Politécnica da USP, 1975
.

[3] GRASSELLI, A. “The Design of Program-Modifiable Microprogrammed Control Units” IRE Transactions on Electronic Computers, p 336-339, junho de 1962

[4] NETO, J.J. “Proposal of a High-Level Language for Writing Self Modifying Programs” IEEE Latin America Transactions, p 192-198 abril 2011

[5] SIQUEIRA F.L. “AdapLib: Uma Biblioteca para Execução de Dispositivos Adaptativos” 3º Workshop de Tecnologia Adaptativa, 2009

Sebastião Santiago Barretto Graduado em Engenharia Eletrônica, modalidade Sistemas Digitais pela Escola Politécnica da Universidade de São Paulo (EPUSP) em 1976, onde também estagiou no Laboratório de Sistemas Digitais (LSD) de Dezembro 1972 a Dezembro de 1976. Trabalhou Na Bireme, Scopus Tecnologia e Diebold Procomp. Atualmente desenvolve APPs para iPhone e iPad na Ciência e Arte Informática. Tem também desenvolvido projetos utilizando módulos Arduino.

João José Neto 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, SP, Brasil. Atualmente é professor associado da Escola Politécnica da Universidade de São Paulo e coordena o LTA – Laboratório de Linguagens e Tecnologia Adaptativa do PCS – Departameto de Engenharia de Computaçao e Sistemas Digitais da EPUSP. Tem experiência na área de Ciência da Computação, com ênfase nos Fundamentos da Engenharia da Computação, atuando principalmente nos seguinte 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.

Parte II

Textos técnicos

Introdução

O Laboratório de Linguagens e Técnicas Adaptativas está iniciando uma chamada geral à produção de uma série de textos técnicos, em formato de monografias, versando sobre assuntos ligados à Adaptatividade, tema central da pesquisa à qual tem se dedicado desde seu surgimento.

O objetivo da série não é simplesmente o de resgatar e consolidar o conteúdo de publicações já existentes, mas, principalmente, o de atualizar e completar o acervo, incluindo novos assuntos, complementando, corrigindo e atualizando materiais referentes a assuntos anteriormente publicados, atualizando terminologias e ideias que já tenham sofrido modificações em virtude do constante progresso da pesquisa e dos desenvolvimentos da área.

Dessa maneira, sugere-se que as monografias desta série tenham todas o centro da sua atenção voltado à Adaptatividade, contudo cada publicação pode e deve incluir suas próprias especificidades, aprofundando aspectos relevantes às particularidades a que a obra for direcionada.

Existe uma vasta gama de temas e de tópicos específicos que podem ser escolhidos, de forma que os autores têm uma liberdade muito grande para escolher as combinações mais apropriadas, bem como para graduar o nível de profundidade e o foco do tratamento a ser dado aos assuntos, podendo assim resultar, do ponto de vista editorial, um grande número de variantes, cada qual adequada para atender uma particular exigência.

Textos técnicos recebidos

A comissão de programa recebeu 8 textos técnicos, em formato de monografias, para análise e avaliação. Os materiais estão em fase de edição e serão disponibilizados em breve. A seguir, são listados todos os trabalhos, com seus respectivos títulos e resumos.

Mecanismos adaptativos na Gramática de Celso Luft

Djalma Padovani

Este texto apresenta uma proposta para inclusão de mecanismos adaptativos na Gramática de Celso Luft, usando como modelo subjacente o formalismo gramatical adaptativo para linguagens dependentes de contexto apresentado por Margarete Iwai em sua tese de doutorado. A proposta visa estender as funcionalidades da Gramática de Luft para que possa ser usada de acordo com contexto sintático.

Utilização de técnicas adaptativas em aprendizagem de máquina e reconhecimento de padrões

Renata Luiza Stange, João José Neto

A adaptatividade é uma característica atribuída ao comportamento automodificável de sistemas computacionais, que ocorre em resposta à estímulos de entrada e ao histórico de operação destes sistemas. A área de aprendizagem de máquina e reconhecimento de padrões apresentam, uma rica fonte de pesquisa para a exploração prática dos fundamentos da tecnologia adaptativa. Isso porque, é intuitivo afirmar que a aprendizagem de um modo geral, possui características adaptativas. O uso da tecnologia adaptativa em diversas áreas tem mostrado

versatilidade deste conjunto de métodos e técnicas na resolução de problemas. Este trabalho tem o objetivo de levantar alguns problemas de aprendizagem de máquina e reconhecimento de padrões e propor soluções baseadas em técnicas adaptativas. A demonstração da aplicabilidade da adaptatividade através de estudos de caso e exemplos de aplicação, é importante para identificar novos desafios e oportunidades na pesquisa, além de mostrar na prática a sua utilização.

AWARE: Adaptive softWARE – Da concepção à simulação de máquinas de estados adaptativas – Aspectos tecnológicos

Ian Silva Oliveira

Os sistemas computacionais viabilizam a automatização de fenômenos dirigidos por regras. Uma instância particular desses fenômenos reside nos dispositivos adaptativos, em que o conjunto de regras é dinâmico, isto é, auto modificável em tempo de execução. Usada nos primórdios da computação, às vezes imperativamente, a auto modificação dificultava a compreensão, manutenção, confiabilidade e a segurança dos programas. Por esse motivo, foi considerada inconveniente/proibitiva como técnica de programação e caiu em desuso por décadas; no entanto, recentemente, voltou a ser empregada em aplicações específicas, já que se mostra aderente às aplicações de linguagens formais, inteligência artificial, modelagem de fenômenos de aprendizagem, inferência e outras. Para que fosse possível o desenvolvimento de uma aplicação adaptativa de qualidade, em computação, meios careciam. Além disso, projetar, controlar, visualizar e documentar um fenômeno adaptativo não são tarefas triviais; dessa forma, inferiu-se uma demanda por uma ferramenta que proporcionasse de forma viável o projeto, o controle e a simulação de dispositivos com essas características, que dispõem de modelos computacionais subjacentes disponíveis distintos. O AWARE consiste em um metassistema dedicados a dispositivos adaptativos decorrentes de Máquinas de Estados. Possui tanto a finalidade de suprir a demanda por uma ferramenta apta ao desenvolvimento de dispositivos adaptativos, pelos iniciantes na área, quanto a de se estabelecer como uma ferramenta alternativa, privilegiada e interessante para modelagem e análise de processos adaptativos. Este livro também descreve como a criação desse sistema deu a luz à um novo formalismo adaptativo: a Máquina de Estados Adaptativa.

Adaptatividade em jogos

Jéssica Leite Pituba

A indústria dos jogos investe cada vez mais em novas tecnologias para tornar o seu conteúdo mais interativo e divertido para o jogador. Uma das tecnologias que podem ser utilizada com jogos para tornar o conteúdo mais dinâmico e personalizado para o jogador é a adaptatividade, que altera os recursos do jogo de acordo com o modelo do jogador, a fim de mantê-lo em um estado de diversão, evitando a frustração e o tédio. Pode ser aplicada em várias partes do jogo, tais como o mundo onde o jogo se passa, o comportamento dos inimigos e aliados, conhecido como AI, as mecânicas do jogo em si e eventos da história do jogo. A adaptação em jogos pode ser feita com vários tipos de tecnologias atuais, assim como o modelo, porém, a eficiência da adaptatividade depende do modelo e nunca deve se tornar invasiva a experiência do jogador, ou o mesmo irá se distrair do conteúdo e quebrar o canal de flow. As técnicas encontradas na pesquisa estão espalhadas pelas várias opções de implementação

de adaptatividade, com um pequeno défcit nas mecânicas, mas ainda há oportunidade para testar novas aplicações em qualquer uma delas.

Uma abordagem adaptativa aplicada no aprimoramento de representações em geometria digital

Leoncio Claro de Barros Neto

As tecnologias adaptativas são formalismos da ciência da computação capazes de alterar seu comportamento dinamicamente, sem a interferência de agentes externos, em resposta a estímulos de entrada. Ao serem capazes de responder às mencionadas condições variáveis do ambiente, os dispositivos adaptativos naturalmente tendem a apresentar a flexibilidade requerida para atuarem em cenários dinâmicos. Analogamente a Euclides que introduziu o conceito de linha reta como um dos elementos básicos de sua geometria, há mais de dois mil anos, da mesma maneira este estudo apresenta inicialmente fundamentos da geometria digital, dentre estes retas e arcos digitalizados em 2-D ($Z \times Z$) , examinados posteriormente sob um enfoque adaptativo, destacando-se a introdução do formalismo de retitude adaptativa (adaptive straightness), que possibilitam uma nova maneira de definir e caracterizar os demais elementos da geometria digital. A seguir, este texto apresenta trabalhos representativos do estado da arte, concluindo com a apresentação de diferentes aplicações em que a adaptatividade é um dos componentes a contribuir para uma melhor compreensão dos aparentes paradoxos, dos desafios apresentados, e para o aprimoramento das representações nessa geometria.

Adaptatividade e computação bioinspirada

Ítalo Santiago Vega, Carlos Eduardo Pires de Camargo, Francisco Supino Marcondes

Um dos objetivos a serem alcançados pela leitura deste texto refere-se ao problema de elaborar modelos de comportamentos mecânicos bioinspirados. Tal elaboração pode ser realizada com o emprego do método MTS. Além disso, também há a preocupação referente com a construção de software de grande porte com comportamentos adaptativos. No texto, encontra-se uma proposta básica para lidar com este ponto: unidades biomiméticas adaptativas.

Assim como a tecnologia adaptativa, a criação de sistemas de software biomimético também vem se desenvolvimento ao longo da última década. A novidade desta obra remete a um processo de desenvolvimento de software que procura sistematizar a etapa de análise e, por meio de padrões de modelagem, apontar questões sobre a elaboração de uma arquitetura de software adaptativa.

O texto organiza-se em duas partes. Na primeira parte, apresentam-se os fundamentos do desenvolvimento de software bioinspirado, a tecnologia Adaptativa e as principais áreas de utilização. Na segunda parte, trata-se do desenvolvimento de software com MTS e UBA.

Dentre as contribuições encontradas nesta publicação citam-se: uma notação para especificar autômatos adaptativos, um método para análise de software bioinspirado (MTS) e um catálogo de padrões biomiméticos que ajudam na elaboração de uma arquitetura de software adaptativa.

Mecanismos adaptativos aplicados à representação e processamento de linguagens naturais

Miryam de Moraes

Em sua tese de livre docência "Contribuições à Metodologia de Construção de Compiladores" [1], o Prof. Dr. João José Neto introduziu o conceito de autômatos adaptativos, cujos mecanismos apresentam "potencial suficiente para tratar dependências de contexto e linguagens extensíveis".

Nas Linguagens Naturais são profusas as ocorrências de não-determinismos e as ambiguidades. Os não-determinismos surgem sempre que duas ou mais construções sintáticas apresentem prefixo comum. Ambiguidades são fenômenos linguísticos em que uma sentença pode ter duas ou mais interpretações válidas na mesma linguagem.

O estudo dos esquemas de projetos presentes em [1] sugere a especificação de uma Linguagem de programação do paradigma declarativo;

O objetivo desta monografia é mostrar que uma Linguagem de programação do paradigma declarativo, cujo compilador faz uso de estratégias adaptativas de projeto tais quais aquelas apresentadas na tese de Livre Docência do professor José Neto são capazes de processar eficientemente não-determinismos e ambiguidades.

O presente trabalho teve como principal referência a tese de José Neto e inicialmente relata observações sobre os mecanismos adaptativos apresentados lá apresentados.

Seguidamente, apresenta-se uma série de especificações de Raposo (1992) para o tratamento de anáforas, representadas em uma Linguagem declarativa compilada pelos mecanismos adaptativos apresentados em [1], procurando assim ilustrar o potencial deste formalismo, para o reconhecimento e processamento da linguagem natural.

Ao final, propõe-se uma arquitetura para o processamento de Linguagem Natural.

Processamento digital para redução de ruído de sinais de aúdio – Noções gerais e proposta de abordagem adaptativa

Antonio Vieira da Silva Neto

Tal como em diversas outras áreas de aplicação, o aumento das capacidades de processamento e armazenamento dos computadores contemporâneos têm viabilizado, com abrangência crescente, a manipulação digital de sinais de áudio. Os avanços relacionados a essa atividade relacionam-se tanto ao processo de captura dos sinais de áudio quanto a eventuais processamentos adicionais deles.

Entre os principais avanços na captura digital de áudio frente aos primórdios dessa atividade, nas décadas de 1970 e 1980, é possível salientar o emprego de frequências de amostragem maiores e o aumento da resolução (número de bits) das amostras coletadas. Sob o ponto de vista qualitativo, tais melhorias técnicas tipicamente conduzem a uma reprodução mais fiel do áudio capturado, com resposta mais homogênea das altas frequências e percepção mais precisa de nuances e transientes.

Com relação à qualidade do processamento do áudio capturado, os avanços de processamento de microcontroladores, microprocessadores e processadores digitais de sinais permitem um tratamento mais adequado de ruídos e imprecisões do áudio original. Esse processamento pode ser executado tanto em tempo real, durante a reprodução ou a gravação do áudio, quanto off-line, manipulando-se um arquivo de áudio previamente capturado.

Dado esse panorama, este livro tem como objetivo apresentar ao leitor as noções gerais dos principais problemas relacionados ao processamento digital para redução de ruído de

arquivos de áudio e possíveis abordagens para resolvê-los. São exploradas nesta obra tanto soluções já referendadas na literatura especializada no tema quanto uma proposta de inclusão dos preceitos da adaptatividade com o objetivo de permitir que os problemas apresentados possam ser resolvidos de forma potencialmente mais eficaz do que por intermédio das abordagens típicas.

Parte III

Submissões

Dispositivos Adaptativos Cooperantes

José Maria Novaes dos Santos

Abstract— Some complex problems can be modeled using more than one type of device thus having some interaction between them to represent their behavior. From this perspective, we do not have a common formulation to represent both the formalism and its interaction. The purpose of this paper is to fill in this gap by proposing a formulation that represents, for a group of devices, their behavior and interactions. We call this formalism as "Cooperating Adaptive Devices". For illustration purposes, we presented an application where its behavior can be modeled using this formalism.

***Keywords*— Reactive systems modeling, adaptive devices, rule-driven formalisms, self-modifying machines, adaptive automata, cooperating adaptive devices, behavior modeling, Decision Table.**

I. INTRODUÇÃO

A adaptatividade é a capacidade de um sistema modificar seu próprio comportamento com base apenas em seu histórico de operações e dados de entrada [1]. Duas instâncias idênticas de um mesmo sistema, porém submetidos a eventos diferentes gerando históricos também distintos, podem evoluir para configurações finais diferentes.

A tecnologia adaptativa é o uso da adaptatividade no desenvolvimento de algum modelo, sistema ou dispositivo dirigido por regras [1].

A adaptatividade pode ser aplicada aos formalismos tradicionais, tais como: Statechart, Autômatos Finitos, Redes de Markov e Tabela de Decisão.

Alguns problemas podem ser melhor representados com a utilização de vários formalismos distintos, contendo uma interação entre si.

Este trabalho apresenta uma formulação para o uso simultâneo de vários formalismos e a representação da sua comunicação, ou seja, entre os dispositivos utilizados na modelagem do problema.

II. TECNOLOGIA ADAPTATIVA

Um dispositivo adaptativo pode ser representado por um conjunto de regras expressas em qualquer formato (que podem ser descritas do tipo "se-então") e um conjunto de operações associado a cada uma das regras. Isto implica em se ter uma relação entre um conjunto de regras (condições) e um conjunto de ações.

Aplicamos a tecnologia adaptativa para representar fenômenos que possam ser representados por um modelo computacional na forma de regras, conforme a definição descrita acima.

Algumas vezes, após a execução do modelo computacional representando várias situações do fenômeno estudado, conclui-se que os conjuntos que representam as regras e ações podem ser modificados para uma melhor adaptação ou aperfeiçoamento do modelo computacional. Esta alteração pode ser representada nas regras iniciais do dispositivo, como regras adaptativas. As regras adaptativas, quando executadas, alteram o conjunto de regras e ações, representando a experiência e aprendizado do dispositivo.

A aplicação da tecnologia adaptativa pode ser aplicada em várias áreas da Ciência da Computação, dentre eles o Processamento de Linguagem Natural (PLN), Robótica, Engenharia de Software, Computação Evolutiva, Inteligência Artificial e Teoria da Computação.

III. DISPOSITIVOS ADAPTATIVOS

Se um dispositivo formal é adaptativo, seu conjunto de regras se modifica dinamicamente, sem ajuda externa, apenas em resposta a estímulos que ocorrem no ambiente e seu histórico de operação.

Segundo [2] um dispositivo adaptativo pode ser decomposto em dois elementos, um dispositivo subjacente, tipicamente não adaptativo, e um mecanismo adaptativo, responsável pela incorporação da adaptatividade. Assim, o comportamento do dispositivo adaptativo segue o comportamento do dispositivo subjacente equivalente, até a execução de alguma ação adaptativa não vazia, mudando a configuração do seu conjunto de regras

Os dispositivos adaptativos dirigidos por regras foram formalmente definidos em [1], no qual foi inspirado o resumo a seguir.

Define-se, em tempo de construção, um contador de tempo T com valor inicial zero, que é automaticamente incrementado por uma unidade, cada vez que uma ação adaptativa não vazia é executada. Assim, cada nome de um conjunto que varia no tempo é identificado pelo valor k, assumido por T.

Dessa forma, pode-se dizer que um dispositivo $AD_k = (ND_k, AM)$ é adaptativo quando para todos os passos de operação k, AD_k segue o comportamento do dispositivo subjacente ND_k até que alguma ação adaptativa, não nula, do mecanismo adaptativo AM seja executada, iniciando-se então o (k+1)-ésimo passo de operações, alterando o conjunto de regras de ND_k, que passa a ser então o conjunto de regras do dispositivo subjacente ND_{k+1}. Um único incremento acarreta a execução de uma ação adaptativa anterior e uma ação adaptativa posterior. Ambas as ações adaptativas podem ser constituídas por um conjunto de ações adaptativas elementares, ou seja, podem ocasionar várias inclusões e/ou exclusões de regras no dispositivo.

O dispositivo AD inicia a sua operação em c_0, com uma forma inicial $AD_0 = (C_0, AR_0, S_0, c_0, A, NA, BA, AA)$. Em um passo $k \geq 0$, um estímulo de entrada sempre altera o dispositivo AD para a próxima configuração, iniciando a operação em (k+1) se e somente se alguma ação adaptativa não vazia foi executada. Então, em qualquer passo k ($k \geq 0$) o dispositivo tem a forma:

$AD_k = (C_k, AR_k, S_k, \mathbf{c_0}, A_k, NA, BA, AA)$. (**1**)

Nessa formulação, temos:

- AD= (ND0, AM) é algum dispositivo adaptativo, composto de um dispositivo subjacente inicial ND0 e um mecanismo adaptativo AM,
- ND_k é um dispositivo não adaptativo de AD no passo k, sendo ND_0 o dispositivo subjacente inicial, definido pelo conjunto de regras NR_0 de regras não adaptativas. Por definição, qualquer regra não adaptativa em qualquer NR_k ($k \geq 0$) espelha a sua correspondente em AR_k,
- C_k é o conjunto de todas configurações possíveis para ND no passo k ($k \geq 0$),
- S é um conjunto finito de todos os possíveis estímulos de entrada considerados como eventos válidos de entrada para AD, sendo que o evento vazio pertence a S ($\epsilon \in S$),
- w é a cadeia de entrada de estímulos, e $w = w_1\ w_2\ w_3\ w_4 \ldots\ldots w_n$ ($w \in S$),
- c_0 pertence a C e é a configuração inicial do dispositivo ($c_0 \in C$),
- A_k é o conjunto das configurações finais (de aceitação), $A_k \subseteq C_k$,
- NA é o conjunto finito de símbolos de saída,
- AR_k é o conjunto de regras adaptativas, dado pela relação $AR_k \subset BA \times C \times S \times C \times NA \times AA$. As regras do conjunto AR0 definem o comportamento inicial do dispositivo AD. Ações adaptativas mapeiam o conjunto corrente de regras adaptativas do dispositivo, AR_k, de AD, em um novo conjunto de regras AR_{k+1} adicionando e/ou excluindo regras adaptativas. As regras em AR_k são da forma ar = (ba, c_i, s, c_j, z, aa), significando que em resposta a algum estímulo $s \epsilon S$ ela executa inicialmente a ação adaptativa anterior, ba, em seguida a regra não adaptativa nr = (c_i, s, c_j, z), e por fim a ação adaptativa posterior aa. Uma ação adaptativa pode eventualmente excluir a regra à qual está associada. Nessa situação, se a ação elementar de exclusão de regra pertencer à ação adaptativa anterior, a atividade de execução da regra será descontinuada assim que a ação adaptativa anterior tiver sido concluída.
- Define-se AR como o conjunto de todas as possíveis regras adaptativas para AD,
- Define-se NR como o conjunto de todas as possíveis regras não adaptativas subjacentes para AD,
- BA e AA são conjuntos de ações adaptativas, ambos podendo ser a ação adaptativa vazia ε ($\varepsilon \in BA \cap AA$) e
- Define-se para um dispositivo AD particular o mecanismo adaptativo como o conjunto $AM \subseteq BA \times NR \times AA$, aplicado em qualquer passo k em cada regra em $NRk \subseteq NR$. AM é de tal forma que opera como uma função aplicada a qualquer subdomínio $NR_k \subseteq NR$. Isso significa associar um par de ação adaptativa a cada regra não adaptativa. AR_k é o conjunto de todas as ações adaptativas que estão associadas às respectivas regras não adaptativas de NR_k.

Em [1] podem ser encontrados os detalhes formais relacionados aos dispositivos adaptativos guiados por regras.

IV. Dispositivos Adaptativos Cooperantes

Os dispositivos dirigidos por regras podem ser utilizados na modelagem de problemas complexos em Inteligência Artificial, Linguagem Natural e outras especialidades da computação, em que haja a necessidade de evidenciar uma alteração de comportamento, em função de mudanças no ambiente com o qual o sistema está interagindo.

Em algumas situações a modelagem pode tornar-se mais compreensível quando efetuada com o auxílio de vários tipos de dispositivos, um grupo heterogêneo de dispositivos em que o comportamento de um, pode afetar o comportamento dos demais.

Os Dispositivos Adaptativos Cooperantes (DAC) ampliam a representação formal de dispositivos adaptativos, dirigidos por regras, nos quais o comportamento de um representante do grupo pode afetar o comportamento de outro dispositivo, resultando na alteração de suas regras.

Dessa forma, seu uso é apropriado para descrever formalmente a representação e modelagem da influência de fatores externos sobre um dispositivo, que pode reagir a eles através de mudanças comportamentais.

Os Dispositivos Adaptativos Cooperantes são constituídos de um grupo finito de dispositivos adaptativos, dirigidos por regras, cujos dispositivos subjacentes podem ser de naturezas distintas, uma camada de comunicação e um mecanismo responsável pelo gerenciamento e coordenação das mensagens de comunicação entre os dispositivos. Cada dispositivo possui um módulo de comunicação e interpretação de mensagens, as quais podem ser trocadas entre qualquer par de dispositivos.

Cada dispositivo é composto de:

• Dispositivo subjacente dirigido por regras,

• Camada adaptativa e

• Módulo de comunicação e interpretação, capaz de enviar e receber mensagens de comunicação padronizadas.

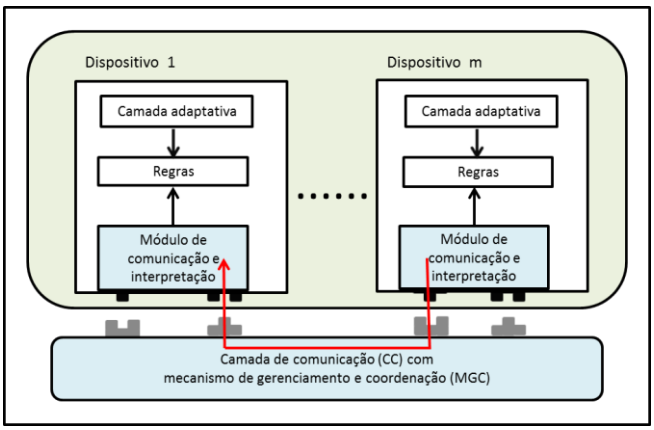


Fig 1 – Comunicação entre dois dispositivos

O conceito de dispositivos adaptativos cooperantes pode ser entendido como sendo a propriedade que um dispositivo, pertencente a um grupo finito de dispositivos, tem de alterar, de forma autônoma, o comportamento de outro dispositivo pertencente ao mesmo grupo, baseado no seu comportamento ou em função de estímulos de entrada.

As mensagens de comunicação são gerenciadas e controladas por meio de um mecanismo de controle.

A Figura 1 ilustra o funcionamento da interação entre dois dispositivos. O dispositivo "m" envia uma mensagem ao dispositivo "1", através da camada de comunicação, representada pela seta vermelha.

Todo dispositivo do conjunto possui a capacidade de enviar e receber solicitação de alteração de regras. A comunicação entre os dispositivos é efetuada através de mensagens padrões, que compõem um protocolo de comunicação (PC).

DEFINIÇÃO

Seja AD o dispositivo adaptativo definido por Neto em (NETO, 2001). Dispositivos Adaptativos Cooperantes podem ser representados por m (m>1) dispositivos adaptativos {ADr} para r=1,.., m, uma camada de comunicação (CC) e um mecanismo responsável pelo gerenciamento e coordenação das mensagens de comunicação (MGC):

$$DAC = (\{AD^r \mid r=1,..,m\}, CC, MGC) \quad (2)$$

Ao grupo de dispositivos adaptativos AD^r (r=1,..,m) podemos denominar de G_mAD, e dessa forma:

$$DAC = (G_mAD , CC, MGC) \quad (3)$$

Onde:

$$G_mAD = \{AD1, AD2, ... ,AD_m\} \quad (4)$$

Da mesma forma que aplicado ao comportamento dos dispositivos adaptativos, define-se para os dispositivos adaptativos cooperantes, em tempo de construção, um sequenciador T2 com valor inicial zero e que é automaticamente incrementado de uma unidade quando uma mensagem não vazia entre dois dispositivos é interpretada pelo módulo de comunicação e interpretação. Assim, cada nome de um dispositivo adaptativo dinamicamente modificável é identificado através de um diferente valor tc para a variável T2. Portanto, os dispositivos adaptativos cooperantes podem ser descritos como:

$$DAC_{tc}= (G_mAD_{tc}, CC, MGC) \quad (5)$$

ou,

$$DAC_{tc}=(\{AD^r_{kr,tc} \mid r=1,..,m; kr \geq 0 \text{ e } tc \geq 0\}, CC, MGC) \quad (6)$$

DAC_{tc} é dito cooperante quando, para todos os passos de operação tc ($tc \geq 0$), DAC_{tc} segue o comportamento de todos os dispositivos adaptativos do conjunto G_mAD_{tc}, até que a geração de alguma mensagem não vazia, do mecanismo CC, enviada por um dispositivo "b" ($b \in \{1,...,m\}$), seja interpretada por um dispositivo destino "r" ($r \in \{1,...,m\}$) iniciando-se um novo passo de operações no (tc+1)-ésimo passo, alterando o conjunto de regras do dispositivo AD^r ($1 \leq r \leq m$, $1 \leq b \leq m$ e $r \neq b$), que passa a ser então o novo conjunto de regras dos dispositivos G_mAD^{tc+1}.

De forma análoga ao que ocorre com as ações adaptativas, um único passo de incremento pode resultar na execução de dois grupos de mensagens, um anterior e o outro posterior à execução de regras. Os dois grupos são constituídos de mensagens elementares (mensagens padrão), de cuja execução podem resultar inclusões e/ou exclusões de regras do dispositivo, e são executados respectivamente antes e depois da execução da regra à qual estão associados, e ao menos um dos grupos de mensagens não é vazio.

Os dispositivos adaptativos cooperantes DAC iniciam a sua operação com uma forma inicial DAC_0, onde cada dispositivo adaptativo AD^r (r=1...m, m>1) tem uma configuração inicial relativa à camada de comunicação, quando o valor tc da fórmula (8) é zero, sendo expresso por:

$$DAC_0=(\{AD^r_{kr,0} \mid r=1,..,m \text{ e } kr \geq 0\}, CC, MGC) \quad (7)$$

Na configuração inicial, representada por DAC_0, nenhuma mensagem entre os dispositivos ocorreu, porém, cada dispositivo individualmente pode ter sofrido alterações em suas regras, decorrentes de suas ações adaptativas, indicadas por kr (r=1,…,m). Em um passo tc ($tc \geq 0$), um estímulo de entrada altera o conjunto de dispositivos AD para a próxima configuração, iniciando a operação no (tc+1)-ésimo passo, se e somente se, alguma mensagem não vazia da camada de comunicação for executada. Então, para qualquer passo tc ($tc \geq 0$), DAC_{tc} corresponde à configuração associada ao passo tc de todos os seus componentes.

Em qualquer combinação de passos kr (k1, k2, ...km) e tc (tc >= 0), cada um dos m dispositivos, (AD^r) para $1 \leq r \leq m$, tem a forma:

$$AD^r_{kr,tc}=(C^r_{kr,tc}, IAR^r_{kr,tc}, S^r, c^r_{kr,tc}, A^r, NA^r, BA^r, AA^r, IBA, IAA) \quad (8)$$

Portanto, $(AD^r)_{tc}$, para r=1,..., m, representa a configuração de cada dispositivo adaptativo, de DAC, no passo tc sendo $(AD^r)_0$ a sua configuração inicial, definida pelo conjunto de regras $IAR^r_{kr,0}$, para qualquer passo kc (para $kc \geq 0$), relativo à execução de ação adaptativa. Por definição, qualquer regra que não tenha mensagem da camada de comunicação (regra de $IAR^r_{kr,tc}$) espelha a sua correspondente em AR^r_{kr}. Assim temos:

- $G_mAD_{tc} =\{ AD^r_{kr,tc} \mid r=1,..,m; kr \geq 0 \text{ e } tc \geq 0\}$ representa a configuração dos dispositivos adaptativos no passo tc, sendo

$AD^r_{0,0}$ a sua configuração inicial. Cada dispositivo $(AD^r)_{tc}$ é o espelho do dispositivo adaptativo AD^r no passo tc, cada um deles em seu passo kr ($kr \geq 0$, para r=1...m), relativo à execução de ação adaptativa.

- $C^r_{kr,tc}$ é o conjunto de todas as configurações possíveis para $AD^r_{kr,tc}$ no passo tc e kr ($tc \geq 0$ e $kr \geq 0$, para r=1...m).

- IBA e IAA (para r=1,…,m) são grupos de mensagens entre dispositivos, ambos podendo ser vazios .

- S^r (para r=1,…,m) é um conjunto finito de todos os possíveis estímulos considerados como válidos, de entrada para ADr, sendo que o estímulo vazio pertence a S^r ($\epsilon \in S^r$); w^r é a cadeia de entrada de estímulos:

$$w^r = w_1{}^r\, w_2{}^r\, w_3{}^r\, w_4{}^r \ldots\ldots w_n{}^r \quad (9)$$

Onde $w_n{}^r \in S^r$, para r=1,...,m e $n \geq 0$.

- $c^r_{0,0}$ pertence a $C^r_{0,0}$ e é a configuração inicial dos dispositivos DAC.

- A^r é o conjunto das configurações finais (de aceitação) dos dispositivos DAC, $A^r \subseteq C^r$ (para r=1,...,m).

- NA^r é o conjunto finito de símbolos de saída dos dispositivos r (para r=1,...,m)

- IARrkr,tc é o conjunto de regras, dado pela relação $IAR^r_{kr,tc} \subseteq IBA^r \times BA^r \times C^r \times S^r \times C^r \times NA^r \times AA^r \times IAA^r$.

As regras do conjunto $IAR^r_{0,0}$ (para r=1,...,m) definem o comportamento inicial do conjunto dos dispositivos DAC. As mensagens entre os dispositivos alteram o conjunto corrente de regras adaptativas do dispositivo, $IAR^r_{kr,tc}$, de DAC, convertendo-o em um novo conjunto de regras em $IAR^r_{kr,tc+1}$. As regras em $IAR^r_{kr,tc}$ (r=1, .., m) são da forma iar^r = (iba, ba^r, c_i^r, s^r, c_j^r, z^r, aa^r, iaa), significando que em resposta a algum estímulo $s^r \in S^r$ ela executa inicialmente o subgrupo de mensagens anterior iba, em seguida a regra adaptativa arr = (ba^r,c_i^r, s^r, c_j^r, z^r, aa^r), e por fim o subgrupo de mensagens posterior iaa. A regra adaptativa ar^r é executada da mesma forma definida em [1].

Para maiores detalhes relacionados à definição dos dispositivos adaptativos cooperantes recomenda-se os trabalhos [3] e [4].

V. EXEMPLO

Nesta seção é apresentado um exemplo de uma aplicação cuja modelagem pode ser composta de dois dispositivos, um "parser" e uma ontologia. Por meio de padrões sintático-semânticos, o "parser" identifica possíveis relações semânticas entre dois conceitos, gerando uma comunicação com o outro dispositivo, a ontologia, resultando na adição de regras, representando a relação entre os conceitos.

Os padrões sintático-semânticos, utilizados no dispositivo "parser" são baseados nos padrões apresentados nos trabalhos (HEARST, 1992) , tais como:

a) SN_0 tais como SN_1 {, SN_2, ...(e | ou) SN_i}

b) SN_0 {, SN,}* { , } e | ou SN_2

c) Tipos de SN: SN_1 {, SN_2, ,} (e | ou) SN_i

d) SN chamado de SN_2.

Trabalhos que apresentam a mesma técnica para construção de ontologia a partir de padrões sintáticos podem ser encontrados em (VELARDI, FARALLI e NAVIGLI, 2013) e (MORIN e CHRISTIAN, 2004).

Nas descrições apresentadas, "SN" são sintagmas nominais, "|" indica opções de escolha que podem ocorrer, "{}" indica uma lista de opções e "*" indica que a ocorrência é opcional.

O "parser" processa um "corpus", extraindo o texto que seja formado por um dos padrões apresentados, identificando as relações semânticas.

Quando as relações são identificadas, o "parser" envia uma mensagem à ontologia, que cria a respectiva regra para identificação da relação semântica.

Para ilustrar o exemplo, consideramos duas frases sobre copa do mundo.

Frase 1: *"Na copa do mundo, os visitantes viram vários jogadores de futebol tais como Messi, Iniesta e Cristiano Ronaldo".*

A frase 1 segue o padrão apresentado no item a, dessa forma o "parser" extrai as seguintes relações:

- hiperonímia("Messi","jogador de futebol"),
- hiperonímia("Iniesta","jogador de futebol") e
- hiperonímia("Cristiano Ronaldo","jogador de futebol").

Frase 2: *"Os jogadores de futebol podem sofrer os tipos de ferimentos: muscular, hematoma e osso quebrado".*

A frase 2 segue o padrão apresentado no item c, da qual são extraídas as relações:

- hiperonímia("muscular","ferimento"),
- hiperonímia("hematoma"," ferimento") e
- hiperonímia("osso quebrado"," ferimento").

A Figura 2 ilustra o exemplo apresentado, indicando as novas relações que são incluídas na ontologia, após a identificação de um padrão por parte do "parser" e a sua respectiva comunicação de cooperação.

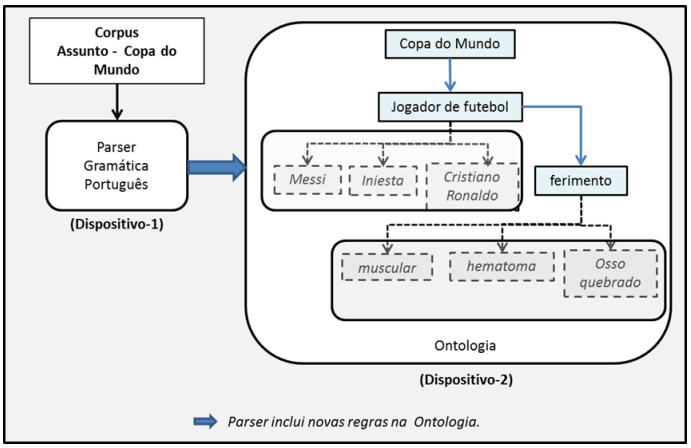


Fig 2 – Novas relações adicionadas na ontologia.

VI. CONSIDERAÇÕES FINAIS

Concluindo, com os DAC temos uma formulação teórica mais abrangente para utilização da adaptividade. Com essa formulação pode-se representar problemas complexos na modelagem de sistemas reativos, sistemas híbridos e sistemas concorrentes, nas mais diversas áreas como, Biologia, Medicina e Ciências Sociais.

Dentro da Engenharia da Computação, pode-se utilizar o modelo DAC na representação de aspectos da Robótica, Aprendizagem de Máquina, Processamento de Linguagem Natural e Linguagem de Programação.

Na Linguagem de Programação pode-se aplicar o modelo, considerando duas linguagens de programação voltadas à codificação de aplicações adaptativas, com uma camada de controle de supervisão, de modo que o comportamento de uma das linguagens possa afetar a outra adequadamente, mantendo sua integridade. Uma das linguagens nesse modelo poderia ser uma linguagem formal de especificação, enquanto a segunda poderia ser a linguagem de codificação.

Pode-se aplicar o modelo DAC também em duas gramáticas de linguagem natural, uma representando um núcleo comum e a outra representando as alterações regionais da língua, alterando algumas das suas regras. Assim podemos fatorar a diversidade regional de uma língua, representada em uma gramática, que quando aplicada influenciará a gramática na forma padrão, gerando uma nova gramática, representando seu caráter regional.

Outra possibilidade é o uso do DAC na tradução de línguas naturais, representando em cada dispositivo um reconhecedor de uma língua natural, efetuando a tradução entre elas por meio de uma comunicação para cada par de reconhecedores.

Resumindo, o modelo DAC abre o leque de aplicações que podem empregar de modo formal a tecnologia adaptativa, utilizando vários tipos de modelos ou dispositivos.

REFERÊNCIAS

[1] J. J. NETO, “Adaptative rule-driven devices – general formulation and a case study”, In: CIAA’2001 Sixth International Conference on Implementation and Application of Automata, pages 234–250, Pretoria, South Africa, July 2001.

[2] J. J. NETO, “Um levantamento da evolução da adaptatividade e da tecnologia adaptativa”. Revista IEEE América Latina, 5.7, 1548-0992, 2007.

[3] J. M. N. SANTOS, “Dispositivos adaptativos cooperantes: formulação e aplicação”, Tese de Doutorado, Escola Politécnica da USP, São Paulo, 2014.

[4] J. M. N. SANTOS and J. J. NETO, "Modeling Reactive Systems Using Cooperating Adaptive Devices", in the Third International Conference on Informatics Engineering and Information Science, 2014, Lodz. Informatics Engineering and Information Science, 2014.

José Maria Novaes dos Santos é bacharel em Matemática Aplicada pela Universidade de São Paulo (1989), possui mestrado em Engenharia da Computação pela Escola Politécnica da Universidade de São Paulo (1997) e doutorado em Engenharia da Computação pela Escola Politécnica da Universidade de São Paulo (2014). Suas áreas de interesse e pesquisa abrangem: teoria da computação, tecnologia adaptativa, estratégias para jogos gerais e processamento em linguagem natural.

AA4J: uma biblioteca para implementação de autômatos adaptativos

P. R. M. Cereda e J. José Neto

***Abstract*—Este artigo apresenta uma biblioteca para a implementação de autômatos adaptativos, utilizando a linguagem Java, de forma consistente e aderente à teoria. Aspectos técnicos do desenvolvimento da biblioteca são apresentados e discutidos, incluindo exemplos de implementação de autômatos para reconhecimento de linguagens regulares, livres de contexto e dependentes de contexto.**

***Palavras-chave:*—Autômato adaptativo, linguagens de programação, bibliotecas, adaptatividade.**

I. Introdução

Este artigo apresenta uma biblioteca chamada AA4J para a implementação de autômatos adaptativos, utilizando a linguagem Java (ou outras linguagens que rodam sobre a máquina virtual Java), de forma consistente e aderente à teoria original proposta por José Neto [1]. Espera-se que, através desta biblioteca, programas possam conter elementos adaptativos especificados em uma linguagem de alto nível.

A organização deste artigo é a seguinte: a Seção II apresenta uma breve revisão bibliográfica da teoria. A Seção III apresenta aspectos técnicos da biblioteca e discussões sobre implementação. Exemplos de implementação de autômatos utilizando AA4J são contemplados na Seção IV. As considerações finais são apresentadas na Seção V.

II. Conceitos iniciais

O *autômato adaptativo*, proposto por José Neto [1], é uma extensão do formalismo do autômato de pilha estruturado que permite o reconhecimento de linguagens do tipo 0, segundo a hierarquia de Chomsky. O termo *adaptativo*, neste contexto, pode ser definido como a capacidade de um dispositivo em alterar seu comportamento de forma espontânea. Logo, um autômato adaptativo tem como característica a possibilidade de provocar alterações em sua própria topologia durante o processo de reconhecimento de uma dada cadeia [2].

Essa capacidade de alteração do autômato faz-se possível através da utilização de ações adaptativas, que podem ser executadas antes ou depois de uma transição. A cada execução de uma ação adaptativa, o autômato tem sua topologia alterada, obtendo-se uma nova configuração. O objetivo de uma ação adaptativa é lidar com situações esperadas, mas ainda não consideradas, detectadas na cadeia submetida para reconhecimento pelo autômato [3]. Uma transição pode ter ações adaptativas associadas, que permitam a inclusão ou eliminação de estados e transições.

Ao executar uma transição que contém uma ação adaptativa associada, o autômato sofre mudanças, obtendo-se então uma nova configuração do autômato. Para a aceitação de uma determinada cadeia, o autômato percorrerá um caminho em um espaço de autômatos; em outras palavras, haverá um autômato E_0, que iniciará o reconhecimento de uma determinada cadeia; autômatos intermediários E_i, que serão criadas ao longo do reconhecimento; e um autômato final E_n, que corresponde ao final do reconhecimento da cadeia. Seja a cadeia $w = \alpha_0\alpha_1 \ldots \alpha_n$; então o autômato M descreverá um caminho de autômatos $\langle E_0, \alpha_0\rangle \rightarrow \langle E_1, \alpha_1\rangle \rightarrow \ldots \rightarrow \langle E_n, \alpha_n\rangle$, no qual E_i representa um autômato correspondente à aceitação da subcadeia α_i.

Definição 1 (autômato adaptativo)**.** Um *autômato adaptativo* M é definido por $M = (Q, S, \Sigma, \Gamma, P, q_0, Z_0, F)$, tal que Q é o conjunto finito de estados, $Q \subset Q^A$, Q^A é o conjunto de todos os estados possíveis, Q^A é enumerável, S é o conjunto finito de submáquinas, Σ é o alfabeto de entrada, $\Sigma \subset \Sigma^A$, Σ^A é o conjunto enumerável de todos os símbolos possíveis, Γ é o alfabeto da pilha, $\Gamma \subset \Gamma^A$, $\Gamma = Q \cup \{Z_0\}$, Γ^A é o conjunto enumerável de todos os símbolos possíveis da pilha, $\Gamma^A = Q^A \cup \{Z_0\}$, P é um mapeamento $P\colon Q^A \times \Sigma^A \times \Gamma^A \rightarrow Q^A \times (\Sigma^A \cup \{\epsilon\}) \times (\Gamma^A \cup \{\epsilon\}) \times H^0 \times H^0\}$, H^0 definido a seguir, $q_0 \in Q$ é o estado inicial, $Z_0 \in \Gamma$ é o símbolo inicial da pilha, $F \subset Q$ é o conjunto de estados finais. Os conjuntos enumeráveis Q^A, Σ^A e Γ^A (com A representando *All* – para todos) são convenientes porque as *funções adaptativas* podem *(a)* inserir novos estados q, $q \notin Q$ mas $q \in Q^A$, e *(b)* usar novos símbolos de pilha $\gamma \notin \Gamma$ mas $\gamma \in \Gamma^A$. Em resumo, as funções adaptativas podem modificar o autômato, mas os novos símbolos que elas introduzem estão todos nos conjuntos enumeráveis [4].

H^0 é o conjunto de todas as funções adaptativas no autômato adaptativo M. Define-se $H^0 = \{f \mid f\colon E \times G_1 \times G_2 \times \ldots \times G_k \rightarrow E\}$, tal que f é uma função, $k \in \mathbb{N}$ é o número de argumentos em f, e $G_i = Q^A \cup \Sigma^A \cup \Gamma^A$.

E é o conjunto de todos os autômatos adaptativos que têm o estado inicial q_0, o símbolo inicial de pilha Z_0 e o conjunto de estados finais F iguais aos do autômato adaptativo M. Define-se $E = \{N \mid N$ é um autômato adaptativo $N = (Q', \Sigma', \Gamma', P', q_0, Z_0, F)$, onde $Q' \subset Q^A$, $\Sigma' \subset \Sigma^A$, $\Gamma' \subset \Gamma^A$, $P'\colon Q^A \times \Sigma^A \times \Gamma^A \rightarrow Q^A \times (\Sigma^A \cup \{\epsilon\}) \times (\Gamma^A \cup \{\epsilon\}) \times H^0 \times H^0\}$. Observe que q_0, Z_0 e F são os mesmos em qualquer $N \in E$. □

Definição 2 (submáquina do autômato adaptativo)**.** O conjunto de todas as submáquinas do autômato adaptativo M é representado por S. Cada *submáquina* s_i é definida como $s_i = (Q_i, \Sigma_i, P_i, q_{i0}, F_i)$, tal que $Q_i \subseteq Q$ é o conjunto de estados da submáquina s_i, $\Sigma_i \subseteq \Sigma$ é o alfabeto de entrada da submáquina s_i, $P_i \subseteq P$ é o mapeamento da submáquina s_i,

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

$q_{i0} \in Q_i$ é o estado inicial da submáquina s_i, e $F_i \subseteq Q_i$ é o conjunto de estados finais da submáquina s_i. □

Uma transição do autômato adaptativo é uma relação da forma $(q, a, \beta) \vdash (q', a', \beta')$ para $P(q, a, \beta) \rightarrow (q', a', \beta', A, B)$, A e B são funções adaptativas, definidas a seguir, $A, B \in H^0$. Se $\tilde{q}$, $\tilde{a}$ ou $\tilde{\beta}$ não pertencem ao autômato corrente, então $P(\tilde{q}, \tilde{a}, \tilde{\beta}) \rightarrow (\tilde{q}, \tilde{a}, \tilde{\beta}, I, I)$, tal que I é a função identidade em E.

As chamadas de funções adaptativas associadas a uma transição são funções de $E \times G_1 \times G_2 \times \ldots \times G_k$ em E. Isto é conveniente porque uma função adaptativa pode ser utilizada em mais de uma transição, com argumentos correspondentes a G_i diferentes. Ao definir uma transição $P(q, a, \beta) \rightarrow (q', a', \beta', A, B)$, é necessário fornecer os argumentos correspondentes aos conjuntos G_i. Estes argumentos podem variar de transição a transição; assim, uma função adaptativa $A\colon E \times G_1 \times G_2 \times \ldots \times G_k \rightarrow E$ pode ser utilizada em diversas transições.

Definição 3 (função adaptativa)**.** Uma *função adaptativa* é qualquer função $A \in H$, tal que $H = \bigcup_{k \geq 0} \{f \mid f\colon E \times G_1 \times \ldots \times G_k \rightarrow E\}$, $G_i = Q^A \cup \Sigma^A \cup \Gamma^A$, $i, k \in \mathbb{N}$. Em outras palavras, H é o conjunto de todas as funções que tomam um autômato adaptativo mais um conjunto de parâmetros em G (qualquer número) e retornam um autômato adaptativo. □

As funções adaptativas podem ser definidas a partir de ações adaptativas elementares de consulta, remoção e inclusão, definidas informalmente a seguir.

As funções adaptativas utilizam-se de variáveis e geradores para executar as ações de edição no autômato. As variáveis são preenchidas uma única vez na execução da função adaptativa. Os geradores são tipos especiais de variáveis, usados para associar nomes unívocos a estados recém-criados; os valores definidos são unívocos e identificados pelo símbolo $*$, por exemplo, g_1^*, g_2^*.

Os autômatos adaptativos possuem três tipos de ações adaptativas elementares utilizadas para edição de seu conjunto de regras [1]. O trecho de autômato da Figura 1 será utilizado para exemplificar as ações adaptativas elementares.

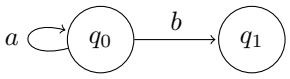


Figura 1. Trecho de um autômato utilizado para exemplificar as ações adaptativas elementares.

– *Ação adaptativa elementar de consulta*: realiza uma busca no autômato por produções cujos componentes sejam correspondentes aos valores passados a essa ação. É denotada por $?(q, a), A \rightarrow (q', a'), B$ ou $?(q, a, \beta), A \rightarrow (q', a', \beta'), B$, tal que q é o estado corrente, a é o símbolo a ser consumido, β é o topo da pilha, q' é o estado de destino, a' é o novo símbolo, podendo ser o mesmo símbolo a ser consumido ou vazio ($a' = a$ ou $a' = \epsilon$), β' é o novo topo da pilha, A é a função adaptativa a ser executada antes da transição, e B é a função adaptativa a ser executada após a transição.

As ações de consulta utilizam-se de variáveis para armazenar o resultado destas consultas. Assim, as variáveis armazenam um conjunto de estados ou transições que correspondem aos parâmetros consultados. Admitindo o trecho de autômato da Figura 1, e utilizando as variáveis $?x$ e $?y$, a Tabela I exemplifica as ações adaptativas elementares de consulta. Observe que, na terceira consulta, $?x$ e $?y$ contêm todos os símbolos possíveis.

Tabela I
EXEMPLOS DE AÇÕES DE CONSULTA.

Consulta	Significado	Resultado
$?(q_0, ?x) \rightarrow (q_1, \epsilon)$	Quais são os símbolos que, a partir do estado q_0, levam ao estado q_1?	$?x = \{b\}$
$?(q_0, b) \rightarrow (?x, \epsilon)$	A partir do estado q_0, qual é o estado de destino, consumindo o símbolo b?	$?x = \{q_1\}$
$?(q_0, ?x) \rightarrow (?y, \epsilon)$	A partir do estado inicial q_0, consumindo qualquer símbolo, quais estados de destino possíveis existem?	$?x = \{a, b\}$, $?y = \{q_0, q_1\}$

– *Ação adaptativa elementar de remoção*: remove uma produção de acordo com os valores passados a essa ação. É denotada por $-(q, a), A \rightarrow (q', a'), B$ ou $-(q, a, \beta), A \rightarrow (q', a', \beta'), B$, tal que q é o estado corrente, a é o símbolo a ser consumido, β é o topo da pilha, q' é o estado de destino, a' é o novo símbolo podendo ser o mesmo símbolo a ser consumido ou vazio ($a' = a$ ou $a' = \epsilon$), β' é o novo topo da pilha, A é a função adaptativa a ser executada antes da transição, e B é a função adaptativa a ser executada após a transição.

A ação adaptativa de remoção exclui elementos do autômato que correspondam aos parâmetros informados. Por exemplo, supondo o o trecho de autômato da Figura 1, a Tabela II exemplifica a execução da ação adaptativa de remoção.

Tabela II
EXEMPLO DE AÇÃO DE REMOÇÃO.

Remoção	Significado
$-[e, (q_0, a) \rightarrow (q_0, \epsilon)]$	Remova a transição que parte do estado q_0, consumindo o símbolo a, e que leva ao próprio estado q_0

O trecho de autômato modificado por essa ação de remoção é ilustrado na Figura 2.

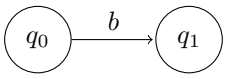


Figura 2. Trecho do autômato da Figura 1, modificado pela ação adaptativa elementar de remoção da Tabela II.

– *Ação adaptativa elementar de inclusão*: inclui uma produção de acordo com os valores passados a essa ação. É denotada por $+(q, a), A \rightarrow (q', a'), B$ ou $+(q, a, \beta), A \rightarrow (q', a', \beta'), B$, tal que q é o estado corrente, a é o símbolo a ser consumido, β é o topo

da pilha, q' é o estado de destino, a' é o novo símbolo podendo ser o mesmo símbolo a ser consumido ou vazio ($a' = a$ ou $a' = \epsilon$), β' é o novo topo da pilha, A é a função adaptativa a ser executada antes da transição, e B é a função adaptativa a ser executada após a transição.
A ação elementar de inclusão insere elementos novos no autômato, como transições e estados novos, de acordo com os argumentos fornecidos. Para a criação de um estado novo, utiliza-se um gerador, o qual receberá um identificador único que será o nome desse estado recém-criado. Por exemplo, supondo o trecho de autômato da Figura 1, e utilizando um gerador g_1^*, a Tabela III ilustra as execuções de ações adaptativas de inclusão.

Tabela III
EXEMPLO DE AÇÕES DE INCLUSÃO.

Inclusão	Significado
$+(q_1, b) \rightarrow (q_1, \epsilon)$	Insira uma transição que parta do estado q_1, consumindo o símbolo b, e que leve ao próprio estado q_1 (*loop*)
$+(q_1, c) \rightarrow (g_1^*, \epsilon)$	Insira uma transição que parta do estado q_1, consumindo o símbolo c, e que leve a um novo estado criado g_1^*

O trecho de autômato, modificado pelas ações de inclusão, é ilustrado na Figura 3.

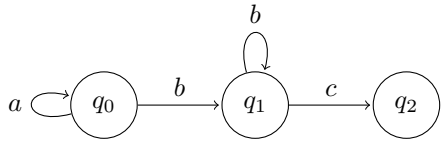


Figura 3. Trecho do autômato da Figura 1, modificado pelas ações adaptativas elementares de inclusão da Tabela III.

Graficamente, é possível representar se uma função adaptativa será executada antes ou depois da transição, através da notação apresentada na Figura 4. Em *(a)*, a função adaptativa A é executada antes da transição, enquanto que em *(b)* a função adaptativa B é executada após a transição. Em *(c)*, as duas funções são executadas, sendo A antes da transição, e B após.

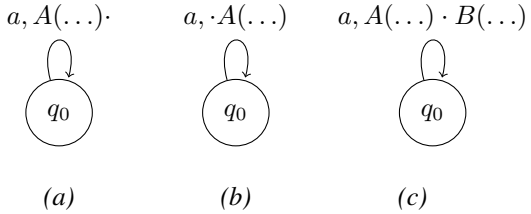


Figura 4. Notação gráfica para determinar o momento de execução das funções adaptativas.

A relação de transição $\vdash$ entre configurações do autômato adaptativo é similar à empregada no autômato de pilha estruturado, sendo que aqui deve ser considerada a presença das funções adaptativas. A linguagem aceita por um autômato adaptativo M é dada por $L(M) = \{w \in \Sigma^* \mid (q_0, w, Z_0) \vdash^* (q_f, \epsilon, Z_0)$, onde $q_f \in F\}$.

III. ASPECTOS TÉCNICOS E IMPLEMENTAÇÃO

A biblioteca `AA4J` permite a implementação de autômatos adaptativos, utilizando a linguagem Java, de acordo com a teoria apresentada na Seção II. Esta seção apresenta os aspectos técnicos e discussões sobre implementação.

A. Estados e símbolos

Estados e símbolos constituem os elementos básicos para a implementação de autômatos adaptativos utilizando a biblioteca `AA4J`. São disponibilizadas duas classes abstratas, `State` e `Symbol`, que devem ser estendidas pelo programador, incluindo os métodos abstratos apresentados na Figura 5. Optou-se pela obrigatoriedade da reescrita dos métodos de comparação, código *hash* e representação textual para garantir que os estados e símbolos possam ser manipulados corretamente durante o processo de reconhecimento (é comum que programadores não atentem-se à necessidade de reescrita de tais métodos).

```
public abstract boolean equals(Object o);
public abstract int hashCode();
public abstract String toString();
```

Figura 5. Métodos abstratos das classes `State` e `Symbol`. Estes métodos devem, obrigatoriamente, ser reescritos nas classes estendidas.

Estados e símbolos podem conter desde tipos primitivos até objetos complexos, proporcionando expressividade ao modelo. A seguir, são apresentados dois exemplos (Exemplos 1 e 2) de classes estendidas contendo diferentes tipos de dados (os códigos utilizam, para fins de simplificação, classes utilitárias da biblioteca Apache Commons Lang[1]).

Exemplo 1. Considere o autômato da Figura 6, no qual seus estados e símbolos são representados por cadeias de caracteres (equivalentes ao tipo de dado `String` em Java). A implementação de tais classes estendidas é apresentada nas Figuras 7 e 8.

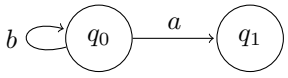


Figura 6. Estados e símbolos são representados por cadeias de caracteres.

A implementação dos estados e símbolos do autômato da Figura 6 utilizando as classes `StrState` e `StrSymbol` das Figuras 7 e 8 é apresentada na Figura 9.

É importante observar que o programador é livre para incluir funcionalidades adicionais às classes, de acordo com a necessidade. □

Exemplo 2. Considere o autômato da Figura 10, no qual seus estados são representados por números inteiros e seus símbolos são representados por valores reais (equivalentes aos tipos de dados `int` e `double` em Java, respectivamente). A implementação de tais classes estendidas é apresentada nas Figuras 11 e 12.

[1]Disponível em `https://commons.apache.org/proper/commons-lang/`.

```
public class StrState extends State {

  private String value;

  public StrState(String value) { this.value = value; }

  public boolean equals(Object object) {
     if (object == null) { return false; }
     else {
        if (!(object.getClass().equals(StrState.class))) {
           return false;
        }
        else {
           return new EqualsBuilder().append(this.getValue(),
                 ((StrState) object).getValue()).isEquals();
  }}}

  public String getValue() { return value; }

  public void setValue(String value) { this.value = value; }

  public int hashCode() { return new
       HashCodeBuilder().append(this.getValue()).hashCode(); }

  public String toString() { return value; }

}
```

Figura 7. Implementação da classe estendida `StrState`, cujo estado é representado por uma cadeia de caracteres.

```
public class StrSymbol extends Symbol {

  private String value;

  public StrSymbol(String value) { this.value = value; }

  public boolean equals(Object object) {
     if (object == null) { return false; }
     else {
        if (!(object.getClass().equals(StrSymbol.class))) {
           return false;
        }
        else {
           return new EqualsBuilder().append(this.getValue(),
                 ((StrSymbol) object).getValue()).isEquals();
  }}}

  public String getValue() { return value; }

  public void setValue(String value) { this.value = value; }

  public int hashCode() { return new
       HashCodeBuilder().append(this.getValue()).hashCode(); }

  public String toString() { return value; }

}
```

Figura 8. Implementação da classe estendida `StrSymbol`, cujo símbolo é representado por uma cadeia de caracteres.

```
State q0 = new StrState("q0");
State q1 = new StrState("q1");
Symbol a = new StrSymbol("a");
Symbol b = new StrSymbol("b");
```

Figura 9. Implementação dos estados e símbolos do autômato da Figura 6 utilizando as classes `StrState` e `StrSymbol` das Figuras 7 e 8.

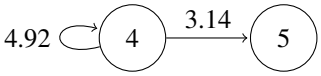


Figura 10. Estados são representados por números inteiros e símbolos são representados por números reais.

A implementação dos estados e símbolos do autômato da Figura 10 utilizando as classes `IntState` e `DoubleSymbol` das Figuras 11 e 12 é apresentada na Figura 13.

Classes mais complexas devem implementar os métodos de comparação, código *hash* e representação textual de acordo com a lógica do modelo. Uma classe que implementa *tokens*, por exemplo, pode comparar apenas as classes gramaticais e ignorar os valores propriamente ditos. □

B. Ações semânticas e adaptativas

A biblioteca `AA4J` disponibiliza uma classe abstrata `Action` para definição de ações. Em linhas gerais, as ações podem ser puramente semânticas (por exemplo, para geração de código ou verificação da existência de um símbolo na tabela de símbolos), adaptativas (definidas somente em termos de ações adaptativas elementares), ou híbridas, contemplando uma combinação de ambas. É necessário estender a classe `Action` e incluir o método abstrato apresentado na Figura 14.

De acordo com a Figura 14, o método abstrato `execute(...)` tem como parâmetros o mapeamento do autômato, a transição corrente que disparou a ação (para referência), e um vetor de objetos como parâmetros formais da ação. Este último permite que as ações sejam parametrizadas quando definidas no escopo de uma transição.

Cada ação deve receber um nome unívoco, que será seu identificador. O nome da ação permitirá que esta seja referenciada posteriormente, durante o processo de reconhecimento de uma cadeia de símbolos. O Exemplo 3 apresenta a implementação de duas ações semânticas.

Exemplo 3. Considere duas ações semânticas: a primeira imprime tão somente uma mensagem no terminal, e a segunda exibe o resultado de um cálculo matemático de acordo com os valores fornecidos como parâmetros. A Figura 15 apresenta tal implementação.

Observe que é necessário realizar a conversão de objetos quando a ação utiliza seus parâmetros formais. Na segunda ação semântica da Figura 15, os objetos devem ser convertidos para valores inteiros (utilizando o conceito de *unboxing*), de modo que o cálculo matemático especificado torne-se, portanto, válido. □

Ações adaptativas elementares são disponibilizadas através de uma classe utilitária chamada `ElementaryActions`. É

```
public class IntState extends State {

  private int value;

  public IntState(int value) { this.value = value; }

  public boolean equals(Object object) {
     if (object == null) { return false; }
     else {
        if (!(object.getClass().equals(IntState.class))) {
           return false;
        }
        else {
           return new EqualsBuilder().append(this.getValue(),
                 ((IntState) object).getValue()).isEquals();
  }}}

  public int getValue() { return value; }

  public void setValue(int value) { this.value = value; }

  public int hashCode() { return new
       HashCodeBuilder().append(this.getValue()).hashCode(); }

  public String toString() { return String.valueOf(value); }

}
```

Figura 11. Implementação da classe estendida `IntState`, cujo estado é representado por um número inteiro.

```
public class DoubleSymbol extends Symbol {

  private double value;

  public DoubleSymbol(double value) { this.value = value; }

  public boolean equals(Object object) {
     if (object == null) { return false; }
     else {
        if (!(object.getClass().equals(DoubleSymbol.class))) {
           return false;
        }
        else {
           return new EqualsBuilder().append(this.getValue(),
                 ((DoubleSymbol)
                      object).getValue()).isEquals();
  }}}

  public double getValue() { return value; }

  public void setValue(double value) { this.value = value; }

  public int hashCode() { return new
       HashCodeBuilder().append(this.getValue()).hashCode(); }

  public String toString() { return String.valueOf(value); }

}
```

Figura 12. Implementação da classe estendida `DoubleSymbol`, cujo símbolo é representado por um número real.

```
State s1 = new IntState(4);
State s2 = new IntState(5);
Symbol v1 = new DoubleSymbol(3.14);
Symbol v2 = new DoubleSymbol(4.92);
```

Figura 13. Implementação dos estados e símbolos do autômato da Figura 10 utilizando as classes `IntState` e `DoubleSymbol` das Figuras 11 e 12.

```
public abstract void execute(Mapping transitions, Transition
     transition, Object... parameters);
```

Figura 14. Método abstrato da classe `Action`. Este método deve, obrigatoriamente, ser reescrito nas classes estendidas.

necessário instanciar um objeto dessa classe, fornecendo o mapeamento do autômato como parâmetro ao construtor. Após a instanciação, é possível utilizar os métodos do objeto que implementam as ações adaptativas elementares (apresentados na Figura 16).

De acordo com a Figura 16, a classe `ElementaryActions` apresenta vários métodos que implementam as ações adaptativas elementares conforme a teoria apresentada na Seção II. Classes utilitárias são utilizadas como suporte para a execução dos métodos que implementam as ações adaptativas elementares:

- `Variable`: representa uma variável de consulta nas ações elementares e pode armazenar um conjunto de valores. Por definição, assim que a variável recebe um ou mais valores (no momento de sua instanciação ou através da aplicação de uma ação elementar de consulta), esta torna-se imutável, isto é, não é possível alterar seu conteúdo.
- `ActionQuery`: contempla um objeto de representação de ações. Uma ação pode ser identificada univocamente através de seu nome (construtor com apenas um parâmetro) ou através de seu nome seguido por seus parâmetros (construtor com múltiplos parâmetros).
- `SubmachineQuery`: contempla um objeto de representação de submáquinas. Uma submáquina é identificada univocamente através de seu nome.

```
Action message = new Action("message") {
  public void execute(Mapping transitions, Transition
       transition, Object... parameters) {
    System.out.println("Hello world!");
  }
};

Action add = new Action("add") {
  public void execute(Mapping transitions, Transition
       transition, Object... parameters) {
    int a = (int) parameters[0];
    int b = (int) parameters[1];
    System.out.println(a + b);
  }
};
```

Figura 15. Implementação de duas ações semânticas.

```
public void query(Variable source, Variable symbol, Variable target);
public void query(Variable source, SubmachineQuery submachine, Variable target);
public void query(Variable source, SubmachineQuery submachine, Variable target, ActionQuery postAction);
public void query(ActionQuery priorAction, Variable source, SubmachineQuery submachine, Variable target);
public void query(ActionQuery priorAction, Variable source, SubmachineQuery submachine, Variable target, ActionQuery
    postAction);
public void query(Variable source, Variable symbol, Variable target, ActionQuery postAction);
public void query(ActionQuery priorAction, Variable source, Variable symbol, Variable target);
public void query(ActionQuery priorAction, Variable source, Variable symbol, Variable target, ActionQuery postAction);
public void remove(Variable source, Variable symbol, Variable target);
public void remove(ActionQuery priorAction, Variable source, Variable symbol, Variable target);
public void remove(Variable source, Variable symbol, Variable target, ActionQuery postAction);
public void remove(ActionQuery priorAction, Variable source, Variable symbol, Variable target, ActionQuery postAction);
public void remove(Variable source, SubmachineQuery submachine, Variable target);
public void remove(Variable source, SubmachineQuery submachine, Variable target, ActionQuery postAction);
public void remove(ActionQuery priorAction, Variable source, SubmachineQuery submachine, Variable target);
public void remove(ActionQuery priorAction, Variable source, SubmachineQuery submachine, Variable target, ActionQuery
    postAction);
public void add(Variable source, Variable symbol, Variable target);
public void add(ActionQuery priorAction, Variable source, Variable symbol, Variable target);
public void add(Variable source, Variable symbol, Variable target, ActionQuery postAction);
public void add(Variable source, SubmachineQuery submachine, Variable target, ActionQuery postAction);
public void add(ActionQuery priorAction, Variable source, SubmachineQuery submachine, Variable target, ActionQuery
    postAction);
public void add(ActionQuery priorAction, Variable source, Variable symbol, Variable target, ActionQuery postAction);
public void add(ActionQuery priorAction, Variable source, SubmachineQuery submachine, Variable target);
public void add(Variable source, SubmachineQuery submachine, Variable target);
```

Figura 16. Métodos da classe `ElementaryActions` que implementam as ações adaptativas elementares.

As classes utilitárias permitem que as ações elementares sejam executadas de modo consistente. O Exemplo 4 apresenta a implementação de uma função adaptativa não-parametrizada utilizando a composição de ações adaptativas elementares.

Exemplo 4. Considere o trecho de autômato da Figura 17, no qual uma função adaptativa $\mathcal{A}$ (Algoritmo 1) realiza alterações em sua topologia, removendo a transição $(q_1, b) \rightarrow q_0$ e inserindo uma nova transição $(q_0, a) \rightarrow q_1$. A implementação de tal função adaptativa é apresentada na Figura 18 (admita a utilização das classes `StrState` e `StrSymbol` introduzidas nas Figuras 7 e 8, respectivamente).

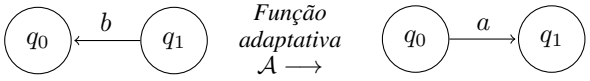


Figura 17. Trecho de autômato que ilustra uma função adaptativa removendo a transição $(q_1, b) \rightarrow q_0$ e inserindo uma nova transição $(q_0, a) \rightarrow q_1$.

Algoritmo 1 Função adaptativa $\mathcal{A}$

função adaptativa $\mathcal{A}$
$-(q_1, b) \rightarrow q_0$
$+(q_0, a) \rightarrow q_1$
fim da função adaptativa

□

É importante observar que a utilização de variáveis contendo múltiplos valores como parâmetros das ações adaptativas elementares resulta na aplicação da operação sobre o produto cartesiano dos parâmetros.

```
Action a = new Action("A") {
  public void execute(Mapping transitions, Transition
      transition, Object... parameters) {
    ElementaryActions ea = new ElementaryActions(transitions);
    State q0 = new StrState("q0");
    State q1 = new StrState("q1");
    Symbol a = new StrSymbol("a");
    Symbol b = new StrSymbol("b");
    ea.remove(new Variable(q1), new Variable(b), new
        Variable(q0));
    ea.add(new Variable(q0), new Variable(a), new
        Variable(q1));
  }
};
```

Figura 18. Implementação da função adaptativa $\mathcal{A}$ do Algoritmo 1.

C. Transições

Transições do autômato são implementadas através da classe `Transition`, que disponibiliza os métodos listados na Figura 19. É possível especificar transições contendo consumo de símbolos, em vazio e chamadas de submáquinas; adicionalmente, cada transição pode conter ações anteriores e posteriores, juntamente com seus parâmetros.

De acordo com a Figura 19, os métodos disponibilizados permitem uma especificação consistente do tipo de transição. Para definir uma transição em vazio, opta-se por utilizar a constante `EPSILON` ou a referência `null` como símbolo a ser consumido. O Exemplo 5 apresenta a implementação das transições do trecho de autômato da Figura 20.

Exemplo 5. Considere o trecho de autômato da Figura 20,

```
public void setSourceState(State sourceState);
public void setSymbol(Symbol symbol);
public void setTargetState(State targetState);
public void setSubmachineCall(String submachineCall);
public void setPriorActionCall(String priorActionCall);
public void setPriorActionArguments(Object[]
    priorActionArguments);
public void setPostActionCall(String postActionCall);
public void setPostActionArguments(Object[]
    postActionArguments);
public void setTransition(State sourceState, Symbol symbol,
    State targetState);
public void setSubmachineCall(State sourceState, String
    submachineCall, State targetState);
```

Figura 19. Métodos da classe `Transition`, que implementa as transições do autômato.

contendo transições com consumo de símbolos, em vazio e chamada de submáquina. A implementação de suas transições é apresentada na Figura 21 (admita a utilização das classes `StrState` e `StrSymbol` introduzidas nas Figuras 7 e 8, respectivamente).

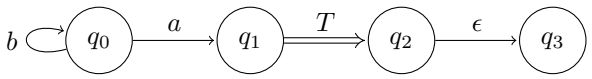


Figura 20. Trecho de autômato contendo transições com consumo de símbolos, em vazio e chamada de submáquina.

```
State q0 = new StrState("q0"); State q1 = new StrState("q1");
State q2 = new StrState("q2"); State q3 = new StrState("q3");
Symbol a = new StrSymbol("a"); Symbol b = new StrSymbol("b");

Transition t1 = new Transition();
t1.setTransition(q0, b, q0);
Transition t2 = new Transition();
t2.setTransition(q0, a, q1);
Transition t3 = new Transition();
t3.setSubmachineCall(q1, "T", q2);
Transition t4 = new Transition();
t4.setTransition(q2, EPSILON, q3);
```

Figura 21. Implementação das transições do trecho de autômato da Figura 20.

Observe que a implementação apresentada na Figura 21 pode ser reescrita utilizando outros métodos da classe `Transition` (Figura 19), o que confere liberdade ao programador para escolher a forma que melhor lhe agradar. □

Internamente, a biblioteca utiliza um tipo especial de transição que realiza o retorno de uma chamada de submáquina. Em linhas gerais, quando a submáquina termina seu reconhecimento em um estado de retorno, ocorre a remoção do estado de destino do topo da pilha e a execução prossegue a partir do novo endereço; a biblioteca realiza então uma transição de retorno, que segue em vazio a partir do estado de retorno da submáquina chamada até o estado que estava no topo da pilha.

D. Submáquinas

A biblioteca `AA4J` considera que todo autômato tem, ao menos, uma submáquina (por exemplo, um autômato finito M possui uma única submáquina, que é ele próprio). Submáquinas são disponibilizadas através da classe `Submachine`, cujo construtor é apresentado na Figura 22. Observe que os parâmetros do construtor são o nome da submáquina, que será utilizado para identificar univocamente a submáquina durante o processo de reconhecimento de uma cadeia de símbolos, o conjunto de estados que compõem a submáquina, o estado inicial (ou de entrada) da submáquina, e o conjunto de estados de aceitação (ou de retorno) da submáquina. O Exemplo 6 apresenta a implementação da especificação da submáquina T da Figura 23.

```
public Submachine(String name, Set<State> states, State
    initialState, Set<State> acceptingStates);
```

Figura 22. Construtor da classe `Submachine` que implementa uma submáquina.

Exemplo 6. Considere a submáquina T da Figura 23, tal que q_0 é o estado inicial e os estados q_1 e q_2 são de aceitação. A implementação da submáquina T é apresentada na Figura 24 (admita a utilização da classe `StrState` introduzida na Figura 7).

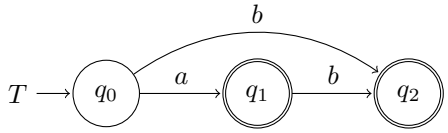


Figura 23. Submáquina T.

```
State q0 = new StrState("q0");
State q1 = new StrState("q1");
State q2 = new StrState("q2");

Set<State> Q = new HashSet<>();
Q.add(q0); Q.add(q1); Q.add(q2);

Set<State> F = new HashSet<>();
F.add(q1); F.add(q2);

Submachine T = new Submachine("T", Q, q0, F);
```

Figura 24. Implementação da especificação da da submáquina T da Figura 23 utilizando a classe `Submachine`.

É importante destacar que os estados da submáquina T são exclusivos a ela, isto é, outras submáquinas não poderão compartilhar estados entre si. □

E. Autômato adaptativo

Uma autômato adaptativo é implementado através da classe abstrata `AdaptiveAutomaton`, que deve ser estendida, incluindo seu método abstrato, para permitir a especificação dos símbolos, estados, transições, submáquinas e ações. O método a ser implementado na classe estendida é apresentado na Figura 25.

```
public abstract void setup();
```

Figura 25. Método da classe abstrata `AdaptiveAutomaton` a ser implementado na classe estendida para permitir a especificação dos símbolos, estados, transições, submáquinas e ações.

É necessário seguir uma seqüência de passos para a definição dos símbolos, estados, transições, submáquinas e ações no corpo do método `setup()`, de tal forma que o autômato seja configurado corretamente. Tais passos são descritos a seguir:

- definir inicialmente todos os símbolos e estados componentes do modelo do autômato, conforme ilustra o exemplo da Figura 9;
- definir todas as ações semânticas, adaptativas ou híbridas pertencentes ao modelo do autômato, conforme ilustram os exemplos das Figuras 15 e 18;
- inserir todas as ações definidas no passo anterior em uma lista de ações chamada `actions`, que é uma variável protegida da classe `AdaptiveAutomaton`; considerando `action` como uma ação definida, a sintaxe para inseri-la na lista é `actions.add(action);` no corpo do método;
- definir todas as submáquinas do modelo do autômato, conforme ilustra o exemplo da Figura 24.
- inserir todas as submáquinas definidas no passo anterior em uma lista de submáquinas chamada `submachines`, que é uma variável protegida da classe `AdaptiveAutomaton`; considerando `submachine` como uma submáquina definida, a sintaxe para inseri-la na lista é `submachines.add(submachine);` no corpo do método;
- definir todas as transições do modelo do autômato, conforme ilustra o exemplo da Figura 21.
- inserir todas as transições definidas no passo anterior em uma lista de transições chamada `transitions`, que é uma variável protegida da classe `AdaptiveAutomaton`; considerando `transition` como uma transição definida, a sintaxe para inseri-la na lista é `transitions.add(transition);` no corpo do método; e
- definir qual será a submáquina principal do autômato, através do método `setMainSubmachine(name)`, no qual `name` é o nome da submáquina principal.

A classe `AdaptiveAutomaton` disponibiliza os métodos concretos apresentados na Figura 26. O método `recognize(...)` é utilizado para realizar o processo de reconhecimento da lista de símbolos, retornando um valor lógico referente ao resultado do reconhecimento. O método `setStopAtFirstResult(...)` permite definir se autômato deve interromper o processo de reconhecimento caso já exista um resultado disponível (no caso de não-determinismo). Os métodos `getRecognitionPaths()` e `getRecognitionMap()` retornam uma lista e um mapa, respectivamente, contendo todos os caminhos de reconhecimento percorridos pelo autômato durante o processo de reconhecimento (em caso de um reconhecimento determinístico, haverá apenas um caminho).

```
public boolean recognize(List<Symbol> input);
public void setStopAtFirstResult(boolean flag);
public List<RecognitionPath> getRecognitionPaths();
public Map<Integer, RecognitionPath> getRecognitionMap();
```

Figura 26. Métodos concretos da classe `AdaptiveAutomaton`.

É importante destacar que os objetos da classe `AdaptiveAutomaton` sempre retornarão à sua configuração inicial após o processo de reconhecimento de uma cadeia de entrada (não há persistência de configurações). Exemplos completos da implementação de autômatos adaptativos utilizando a classe `AdaptiveAutomaton` são apresentados em detalhes na Seção IV.

F. Registro de eventos

A biblioteca `AA4J` registra todos os eventos de suas classes internas para cada chamada do processo de reconhecimento de uma cadeia de símbolos. Assim, é possível analisar a execução de forma incremental. A granularidade do registro está definida para capturar apenas eventos graves, mas é possível ajustar os valores através da edição do arquivo de configuração chamado `log4j2.xml`, utilizado pela biblioteca Apache Log4j[2], responsável pelo gerenciamento do registro.

O registro de eventos torna-se útil para automatizar o processo de validação de um modelo utilizando autômatos adaptativos. Adicionalmente, é possível utilizar os métodos `getRecognitionPaths()` e `getRecognitionMap()` diretamente no código-fonte para analisar o comportamento da execução (esta técnica, entretanto, não oferece detalhes complementares).

G. Não-determinismo

A biblioteca `AA4J` permite a definição e execução de modelos não-determinísticos, tal que todos os caminhos válidos sejam inspecionados e executados simultaneamente. Por padrão, a biblioteca aguarda o término de todas as ramificações do processo de reconhecimento de uma cadeia de símbolos, mas é possível alterar tal comportamento através do método `setStopAtFirstResult(...)`, que permite interromper o processamento quando um dos caminhos já encerrou-se.

Para determinar se o processo de reconhecimento de uma cadeia de símbolos foi determinístico, é suficiente verificar o tamanho da lista de caminhos de reconhecimento retornada pelo método `getRecognitionPaths()`; caso a lista possua apenas um caminho de reconhecimento, a execução foi determinística (é possível realizar o mesmo teste com o mapa retornado pelo método `getRecognitionMap()`, verificando o tamanho do conjunto de chaves).

H. Ciclos em vazio

A versão corrente da biblioteca `AA4J` não detecta ciclos em vazio potenciais (triviais ou não) no modelo do autômato adaptativo, podendo causar uma execução infinita; assim, é altamente recomendável que estes ciclos sejam detectados e removidos *a priori* para evitar problemas de execução.

[2]Disponível em `http://logging.apache.org/log4j/`.

IV. EXEMPLOS

Esta seção apresenta três exemplos de autômatos para ilustrar o funcionamento da biblioteca `AA4J`. Todas as implementações apresentadas utilizam as classes `StrState` e `StrSymbol` introduzidas nas Figuras 7 e 8, respectivamente. Considere também a existência de um método estático `List<Symbol> convert(final String text)` que converte um texto para uma lista de símbolos.

Exemplo 7. Considere um autômato finito M_1 que reconhece cadeias pertencentes à linguagem regular $L_1 = \{w \in \{a,b\}^* \mid w = (ab)^+\}$, de acordo com a Figura 27. A implementação de tal autômato é apresentada na Figura 28.

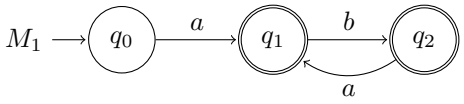


Figura 27. Autômato finito M_1 que reconhece cadeias pertencentes à linguagem regular $L = \{w \in \{a,b\}^* \mid w = (ab)^+\}$.

```
AdaptiveAutomaton aa = new AdaptiveAutomaton() {
 public void setup() {

   State q0 = new StrState("q0");
   State q1 = new StrState("q1");
   State q2 = new StrState("q2");

   Symbol a = new StrSymbol("a");
   Symbol b = new strSymbol("b");

   Set<State> Q = new HashSet<>();
   Q.add(q0); Q.add(q1); Q.add(q2);

   Set<State> F = new HashSet<>();
   F.add(q2);

   Submachine M = new Submachine("M", Q, q0, F);

   Transition t1 = new Transition();
   t1.setTransition(q0, a, q1);

   Transition t2 = new Transition();
   t2.setTransition(q1, b, q2);

   Transition t3 = new Transition();
   t3.setTransition(q2, a, q1);

   submachines.add(M);

   transitions.add(t1);
   transitions.add(t2);
   transitions.add(t3);

   setMainSubmachine("M");
 }
};

System.out.println(aa.recognize(convert("ab")));
System.out.println(aa.recognize(convert("abab")));
System.out.println(aa.recognize(convert("aba")));
```

Figura 28. Implementação do autômato finito M da Figura 27.

O resultado da execução do trecho de código-fonte apresentado na Figura 28 é: `true`, `true`, `false`. □

Exemplo 8. Considere um autômato de pilha estruturado M_2 que reconhece cadeias pertencentes à linguagem livre de contexto $L_2 = \{w \in \{a,b\}^* \mid w = a^n b^n, n \in \mathbb{Z}\}$, de acordo com a Figura 29. A implementação de tal autômato é apresentada na Figura 30.

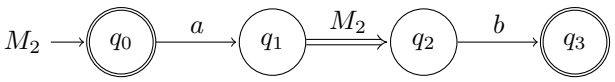


Figura 29. Autômato de pilha estruturado M_2 que reconhece cadeias pertencentes à linguagem livre de contexto $L_2 = \{w \in \{a,b\}^* \mid w = a^n b^n, n \in \mathbb{Z}\}$.

```
AdaptiveAutomaton aa = new AdaptiveAutomaton() {
 public void setup() {

   State q0 = new StrState("q0");
   State q1 = new StrState("q1");
   State q2 = new StrState("q2");
   State q3 = new StrState("q3");

   Symbol a = new StrSymbol("a");
   Symbol b = new StrSymbol("b");

   Set<State> Q = new HashSet<>();
   Q.add(q0); Q.add(q1); Q.add(q2); Q.add(q3);

   Set<State> F = new HashSet<>();
   F.add(q0); F.add(q3);

   Submachine M = new Submachine("M", Q, q0, F);

   Transition t1 = new Transition();
   t1.setTransition(q0, a, q1);

   Transition t2 = new Transition();
   t2.setSubmachineCall(q1, "M", q2);

   Transition t3 = new Transition();
   t3.setTransition(q2, b, q3);

   submachines.add(M);

   transitions.add(t1);
   transitions.add(t2);
   transitions.add(t3);

   setMainSubmachine("M");
 }
};

System.out.println(aa.recognize(convert("ab")));
System.out.println(aa.recognize(convert("aab")));
System.out.println(aa.recognize(convert("aabb")));
```

Figura 30. Implementação do autômato de pilha estruturado M_2 da Figura 29.

O resultado da execução do trecho de código-fonte apresentado na Figura 30 é: `true`, `false`, `true`. □

Exemplo 9. Considere um autômato adaptativo M_3 que reconhece cadeias pertencentes à linguagem dependente de contexto $L_3 = \{w \in \{a,b,c\}^* \mid w = a^n b^n c^n, n \in \mathbb{Z}, n \geq 1\}$, de acordo com a Figura 31. A função adaptativa $\mathcal{A}$ é apresentada no Algoritmo 2. A implementação de tal autômato é apresentada na Figura 32. Considere a existência de um método estático `generateState()` que retorna novos estados (através de um contador interno, por exemplo).

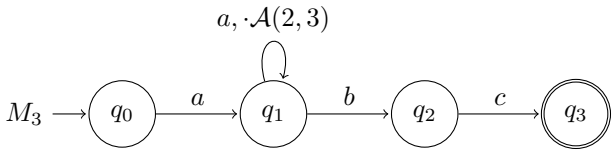


Figura 31. Autômato adaptativo M_3 que reconhece cadeias pertencentes à linguagem dependente de contexto $L_3 = \{w \in \{a,b,c\}^* \mid w = a^n b^n c^n, n \in \mathbb{Z}, n \geq 1\}$.

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

função adaptativa $\mathcal{A}(p_1, p_2)$
variáveis: $?x, ?y$
geradores: $g_1^\star, g_2^\star$
$?(?x, b) \rightarrow p_1$
$-(?x, b) \rightarrow p_1$
$?(?y, c) \rightarrow p_2$
$-(?y, c) \rightarrow p_2$
$-(q_1, a) \rightarrow q_1, \cdot\mathcal{A}(p_1, p_2)$
$+(?x, b) \rightarrow g_1^\star$
$+(g_1^\star, b) \rightarrow p_1$
$+(?y, c) \rightarrow g_2^\star$
$+(g_2^\star, c) \rightarrow p_2$
$+(q_1, a) \rightarrow q_1, \cdot\mathcal{A}(g_1^\star, g_2^\star)$
fim da função adaptativa

O resultado da execução do trecho de código-fonte apresentado na Figura 32 é: `false`, `true`, `true`. □

V. Considerações finais

O código-fonte da biblioteca `AA4J` está disponível para consulta e download em `https://github.com/cereda/aa`, sob a licença GPLv3[3]. O bytecode gerado é compatível com Java 7 ou versões superiores. Espera-se que esta contribuição possa proporcionar subsídios para a implementação de programas que apresentem características adaptativas.

A biblioteca apresentada neste artigo pode contribuir para a implementação de autômatos adaptativos utilizando a linguagem Java de forma consistente e aderentes à teoria. Adicionalmente, programas escritos em outras linguagens que rodam sobre a máquina virtual Java (tais como Groovy, Scala e Clojure) podem compartilhar o mesmo *bytecode* e utilizar as funcionalidades apresentadas.

[3]Disponível em `http://www.gnu.org/licenses/gpl-3.0.html`.

Referências

[1] J. José Neto, "Contribuições à metodologia de construção de compiladores," Tese de livre docência, Escola Politécnica da Universidade de São Paulo, São Paulo, 1993.
[2] ——, "Adaptive automata for context-sensitive languages," *SIGPLAN Notices*, vol. 29, no. 9, pp. 115–124, set 1994.
[3] ——, "Adaptive rule-driven devices: general formulation and case study," in *International Conference on Implementation and Application of Automata*, 2001.
[4] P. R. M. Cereda, "Modelo de controle de acesso adaptativo," Dissertação de mestrado, Departamento de Computação, Universidade Federal de São Carlos, 2008.

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.

```
AdaptiveAutomaton aa = new AdaptiveAutomaton() {
 public void setup() {

  State q0 = new StrState("q0"); State q1 = new StrState("q1");
  State q2 = new StrState("q2"); State q3 = new StrState("q3");

  Symbol a = new StrSymbol("a"); Symbol b = new StrSymbol("b"); Symbol c = new StrSymbol("c");

  Set<State> Q = new HashSet<>(); Q.add(q0); Q.add(q1); Q.add(q2); Q.add(q3);

  Set<State> F = new HashSet<>(); F.add(q3);

  Submachine M = new Submachine("M", Q, q0, F);

  submachines.add(M);

  Transition t1 = new Transition(); t1.setTransition(q0, a, q1);
  Transition t2 = new Transition(); t2.setTransition(q1, b, q2);
  Transition t3 = new Transition(); t3.setTransition(q2, c, q3);
  Transition t4 = new Transition(); t4.setTransition(q1, a, q1);

  t4.setPostActionCall("A");
  t4.setPostActionArguments(Variable.values(q2, q3));

  transitions.add(t1); transitions.add(t2); transitions.add(t3); transitions.add(t4);

  Action adapt = new Action("A") {
   public void execute(Mapping transitions, Transition transition, Object... parameters) {

    Symbol a = new StrSymbol("a"); Symbol b = new StrSymbol("b");
    Symbol c = new StrSymbol("c"); State q1 = new StrState("q1");

    ElementaryActions ea = new ElementaryActions(transitions);

    Variable p1 = new Variable(parameters[0]); Variable p2 = new Variable(parameters[1]);
    Variable g1 = new Variable(generateState()); Variable g2 = new Variable(generateState());
    Variable x = new Variable(); Variable y = new Variable();

    ea.query(x, new Variable(b), p1); ea.remove(x, new Variable(b), p1);
    ea.query(y, new Variable(c), p2); ea.remove(y, new Variable(c), p2);

    ea.remove(new Variable(q1), new Variable(a), new Variable(q1), new ActionQuery(new Variable("A"), p1, p2));

    ea.add(x, new Variable(b), g1); ea.add(g1, new Variable(b), p1);
    ea.add(y, new Variable(c), g2); ea.add(g2, new Variable(c), p2);
    ea.add(new Variable(q1), new Variable(a), new Variable(q1), new ActionQuery(new Variable("A"), g1, g2));
   }
  };

  actions.add(adapt);

  setMainSubmachine("M");
 }
};

System.out.println(aa.recognize(convert("aabbc")));
System.out.println(aa.recognize(convert("aabbcc")));
System.out.println(aa.recognize(convert("aaabbbccc")));
```

Figura 32. Implementação do autômato adaptativo M_3 da Figura 31.

O Reconhecedor Gramatical Linguístico: Avanços em Desambiguação Sintática e Semântica

A. T. Contier, D. Padovani e J. 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, um reconhecedor gramatical que utiliza autômatos e gramáticas adaptativas como tecnologia subjacente. O presente artigo é uma continuação de trabalhos anteriores em que foi incluída uma nova sessão para apresentar os avanços em desambiguação sintática e semântica.**

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

I. Autômatos Adaptativos

O AUTÔMATO adaptativo é uma máquina de estados à qual são impostas sucessivas alterações resultantes da aplicação de ações adaptativas associadas às regras de transições executadas pelo autômato [1]. Dessa maneira, estados e transições podem ser eliminados ou incorporados ao autômato em decorrência de cada um dos passos executados durante a análise da entrada. De maneira geral, pode-se dizer que o autômato adaptativo é formado por um dispositivo convencional, não adaptativo, e um conjunto de mecanismos adaptativos responsáveis pela auto modificação do sistema.

O dispositivo convencional pode ser uma gramática, um autômato, ou qualquer outro dispositivo que respeite um conjunto finito de regras estáticas. Este dispositivo possui uma coleção de regras, usualmente na forma de cláusulas if-then, que testam a situação corrente em relação a uma configuração específica e levam o dispositivo à sua próxima situação. Se nenhuma regra é aplicável, uma condição de erro é reportada e a operação do dispositivo, descontinuada. Se houver uma única regra aplicável à situação corrente, a próxima situação do dispositivo é determinada pela regra em questão. Se houver mais de uma regra aderente à situação corrente do dispositivo, as diversas possíveis situações seguintes são tratadas em paralelo e o dispositivo exibirá uma operação não determinística. Os mecanismos adaptativos são formados por três tipos de ações adaptativas elementares: consulta (inspeção do conjunto de regras que define o dispositivo), exclusão (remoção de alguma regra) e inclusão (adição de uma nova regra).

Autômatos adaptativos apresentam forte potencial de aplicação ao processamento de linguagens naturais, devido à facilidade com que permitem representar fenômenos linguísticos complexos tais como dependências de contexto. Adicionalmente, podem ser implementados como um formalismo de reconhecimento, o que permite seu uso no pré-processamento de textos para diversos usos, tais como: análise sintática, verificação de sintaxe, processamento para traduções automáticas, interpretação de texto, corretores gramaticais e base para construção de sistemas de busca semântica e de aprendizado de línguas auxiliados por computador.

Diversos trabalhos confirmam a viabilidade prática da utilização de autômatos adaptativos para processamento da linguagem natural. É o caso, por exemplo, de [2], que mostra a utilização de autômatos adaptativos na fase de análise sintática; [3] que apresenta um método de construção de um analisador morfológico e [4], que apresenta uma proposta de autômato adaptativo para reconhecimento de anáforas pronominais segundo algoritmo de Mitkov.

II. Processamento da Linguagem Natural: Revisão da Literatura

O processamento da linguagem natural requer o desenvolvimento de programas que sejam capazes de determinar e interpretar a estrutura das sentenças em muitos níveis de detalhe. As linguagens naturais exibem um intricado comportamento estrutural visto que são profusos os casos particulares a serem considerados. Uma vez que as linguagens naturais nunca são formalmente projetadas, suas regras sintáticas não são simples nem tampouco óbvias e tornam, portanto, complexo o seu processamento computacional. Muitos métodos são empregados em sistemas de processamento de linguagem natural, adotando diferentes paradigmas, tais como métodos exatos, aproximados, pré-definidos ou interativos, inteligentes ou algorítmicos [5]. Independentemente do método utilizado, o processamento da linguagem natural envolve as operações de análise léxico-morfológica, análise sintática, análise semântica e análise pragmática [6]. A análise léxico-morfológica procura atribuir uma classificação morfológica a cada palavra da sentença, a partir das informações armazenadas no léxico [7]. O léxico ou dicionário é a estrutura de dados contendo os itens lexicais e as informações correspondentes a estes itens. Entre as informações associadas aos itens lexicais, encontram-se a categoria gramatical do item, tais como substantivo, verbo e adjetivo, e os valores morfossintático-semânticos, tais como gênero, número, grau, pessoa, tempo, modo, regência verbal ou nominal. Na etapa de análise sintática, o analisador verifica se uma sequência de palavras constitui uma frase válida da língua, reconhecendo-a ou não. O analisador sintático faz uso de um léxico e de uma gramática, que define as regras de combinação dos itens na formação das frases. Nos casos nos quais há a necessidade de interpretar o significado de um texto, a análise léxico-morfológica e a análise sintática não são

suficientes, sendo necessário realizar um novo tipo de operação, denominada análise semântica [7]. Na análise semântica procura-se mapear a estrutura sintática para o domínio da aplicação, fazendo com que a estrutura ganhe um significado. O mapeamento é feito identificando as propriedades semânticas do léxico e o relacionamento semântico entre os itens que o compõe [8]. Já a análise pragmática procura reinterpretar a estrutura que representa o que foi dito para determinar o que realmente se quis dizer. Inserem-se nessa categoria as relações anafóricas, correferências, determinações, focos ou temas, dêiticos e elipses [9].

Em [10] são apresentados trabalhos de pesquisas em processamento de linguagem natural para a Língua Portuguesa tais como o desenvolvido pelo Núcleo Interinstitucional de Linguística Aplicada (NILC) no desenvolvimento de ferramentas para processamento de linguagem natural; o projeto VISL – Visual Interactive Syntax Learning, sediado na Universidade do Sul da Dinamarca, que engloba o desenvolvimento de analisadores morfossintáticos para diversas línguas, entre as quais o português; e o trabalho de resolução de anáforas desenvolvido pela Universidade de Santa Catarina. A tecnologia adaptativa também tem contribuído com trabalhos em processamento da linguagem natural. Em [11], são apresentadas algumas das pesquisas desenvolvidas pelo Laboratório de Linguagens e Tecnologia Adaptativa da Escola Politécnica da Universidade de São Paulo: um etiquetador morfológico, um estudo sobre processos de análise sintática, modelos para tratamento de não determinismos e ambiguidades, e um tradutor texto voz baseado em autômatos adaptativos.

III. RECONHECEDOR ADAPTATIVO: SUPORTE TEÓRICO LINGUÍSTICO

A Moderna Gramática Brasileira de Celso Luft [12] foi escolhida como suporte teórico linguístico do reconhecedor aqui proposto. A escolha foi feita em função da forma clara e precisa com que Luft categoriza os diversos tipos de sentenças de língua portuguesa, diferenciando-se das demais gramáticas que priorizam a descrição da língua em detrimento da análise estrutural da mesma. Luft diz que a oração é moldada por padrões frasais ou oracionais, compostos por elementos denominados sintagmas (Tabela 1).

TABELA 1.
ELEMENTOS FORMADORES DE SINTAGMAS

Sintagmas	
Substantivo	Quantitativos+Pronomes Adjetivos+ Sintagma Adjetivo1+Substantivo+ Sintagma Adjetivo2+ Sintagma Preposicional+ Oração Adjetiva
Verbal	Pré-verbais+ Verbo Auxiliar+ Verbo Principal

TABELA 1.
CONTINUAÇÃO

Sintagmas	
Adjetivo	Advérbio de Intensidade+ Adjetivo+ Sintagma Preposicional
Adverbial	Advérbio de Intensidade+ Adverbio+ Sintagma Preposicional
Preposicional	Preposição+ Sintagma Substantivo

Sintagma é qualquer constituinte imediato da oração, podendo exercer papel de sujeito, complemento (objeto direto e indireto), predicativo e adjunto adverbial. É composto por uma ou mais palavras, sendo que uma é classificada como núcleo e as demais como dependentes. As palavras dependentes podem estar localizadas à esquerda ou à direita do núcleo. Luft utiliza os seguintes nomes e abreviaturas:

1. Sintagma substantivo (SS): núcleo é um substantivo;
2. Sintagma verbal (SV): núcleo é um verbo;
3. Sintagma adjetivo (Sadj): núcleo é um adjetivo;
4. Sintagma adverbial (Sadv): núcleo é um advérbio;
5. Sintagma preposicional (SP): é formado por uma preposição (Prep) mais um SS.
6. Vlig: verbo de ligação
7. Vi: verbo intransitivo
8. Vtd: verbo transitivo direto
9. Vti: verbo transitivo indireto
10. Vtdi: verbo transitivo direto e indireto
11. Vt-pred: verbo transitivo predicativo

Um padrão oracional é determinado pelos tipos de sintagmas e pela sequência em que aparecem. Por exemplo, o padrão oracional SS Vlig SS, indica que a frase é composta por um sintagma substantivo, seguido de um verbo de ligação e de outro sintagma substantivo. Os padrões são classificados em cinco tipos (Tabela 2):

1. Padrões pessoais nominais: Neste caso, existe sujeito e o núcleo do predicado é um nome (substantivo, adjetivo, advérbio) ou um pronome (substantivo, adjetivo, advérbio). O verbo, nesses casos, é chamado de verbo de ligação (Vlig).
2. Padrões pessoais verbais: São aqueles nos quais existe o sujeito e o núcleo do predicado é um verbo. O verbo pode ser transitivo direto (Vtd), transitivo indireto (Vti), transitivo direto e indireto (Vtdi), e intransitivo (Vi). Se o verbo for transitivo direto (Vtd), o complemento será um objeto direto; se o verbo for transitivo indireto (Vti), o complemento será um objeto indireto; se o verbo for transitivo direto e indireto (Vtdi), o complemento será um objeto direto e um indireto; se o verbo for intransitivo (Vi), não há complemento.
3. Padrões Pessoais Verbo-Nominais: Neste caso, existe o sujeito e o núcleo do predicado é um verbo transitivo predicativo (Vt-pred), cujo complemento é um objeto direto e um predicativo do objeto.

4. Padrões Impessoais Nominais: Ocorrem quando não existe sujeito e o núcleo do predicado é um nome (substantivo, adjetivo, advérbio) ou um pronome (substantivo, adjetivo, advérbio).
5. Padrões Impessoais Verbais: Neste caso, não existe sujeito e o núcleo do predicado é um verbo.

TABELA 2
PADRÕES ORACIONAIS DE LUFT

Padrões Pessoais Nominais				
SS	Vlig	SS		
SS	Vlig	Sadj		
SS	Vlig	Sadv		
SS	Vlig	SP		
Padrões Pessoais Verbais				
SS	Vtd	SS		
SS	Vti	SP		
SS	Vti	Sadv		
SS	Vti	SP	SP	
SS	Vtdi	SS	SP	
SS	Vtdi	SS	Sadv	
SS	Vtdi	SS	SP	SP
SS	Vi			
Padrões Pessoais Verbo-Nominais				
SS	Vtpred	SS	SS	
SS	Vtpred	SS	Sadj	
SS	Vtpred	SS	SP	
SS	Vtpred	SS	Sadv	
SS	Vtpred	SS		
SS	Vtpred	Sadj		
SS	Vtpred	SP		
Padrões Impessoais Nominais				
	Vlig	SS		
	Vlig	Sadj		
	Vlig	Sadv		
	Vlig	SP		
Padrões Impessoais Verbais				
	Vtd	SS		
	Vti	SP		
	Vi			

Os sintagmas e os padrões oracionais de Luft foram mapeados em uma gramática para que pudessem ser processados computacionalmente, ficando da seguinte forma:

F → [Conec] [SS] SV [Conec]
Conec → F
SS → [Sadj] SS [Sadj | SP]
SS → [Quant | PrA] (Sc | Sp | PrPes)
SV→ [Neg] [Aux | PreV] (Vlig | Vtd| Vti | Vtdi| Vi)
[SS | Sadj | Sadv | SP] [SS| Sadj| Sadv| SP] [SP]
SP → Prep (SS | Sadj)
Sadj → Sadj [SP]
Sadj →[Adv] Adj
Sadv → Sadv [SP]
Sadv → [Adv] Adv
PrA → Ind | ArtDef | ArtInd| Dem| Pos

Sendo:
F – Frase
SS – Sintagma substantivo
SV – Sintagma verbal
SP – Sintagma preposicional
SN – Sintagma nominal
Sadv – Sintagma adverbial
Sadj – Sintagma adjetivo
Adv – adverbio
Adj – adjetivo
ArtDef – artigo definido
ArtInd – artigo indefinido
Aux – Partícula auxiliar (apassivadora ou pré-verbal)
Conec – Conector (conjunção ou pronome relativo)
Dem – pronome demonstrativo indefinido
Ind – pronome indefinido
Neg – partícula (negação)
PrA – pronome adjetivo
PrPes – pronome pessoal
Prep – preposição
Quant – numeral
Sc – substantivo comum
Sp – substantivo próprio
V– verbo
Vlig – verbo de ligação
Vi – verbo intransitivo
Vtd – verbo transitivo direto
Vti – verbo transitivo indireto
Vtdi – verbo transitivo direto e indireto

IV. PROPOSTA DE UM RECONHECEDOR GRAMATICAL

O Linguístico é uma proposta de reconhecedor gramatical composto de cinco módulos sequenciais que realizam cada qual um processamento especializado, enviando o resultado obtido para o módulo seguinte, tal como ocorre em uma linha de produção, até que o texto esteja completamente analisado (Fig.1).

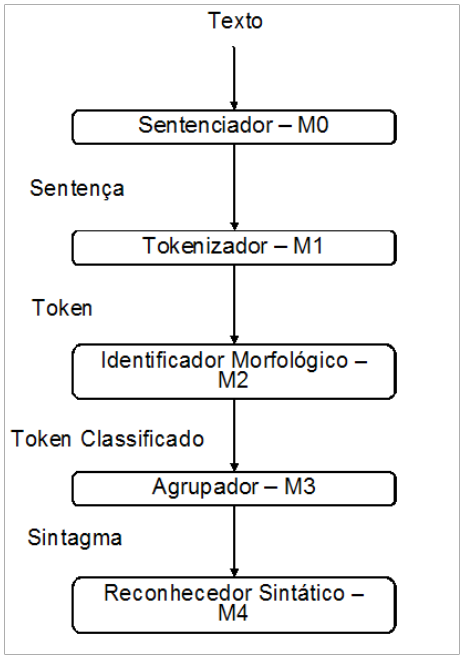


Figura 1. Estrutura do Linguístico.

O primeiro módulo, denominado Sentenciador, recebe um texto e realiza um pré-processamento, identificando os caracteres que possam indicar final de sentença, palavras abreviadas e palavras compostas, e eliminando aspas simples e duplas. Ao final, o Sentenciador divide o texto em supostas sentenças, para análise individual nas etapas seguintes.

O segundo módulo, denominado Tokenizador, recebe as sentenças identificadas na etapa anterior e as divide em *tokens*, considerando, neste processo, abreviaturas, valores monetários, horas e minutos, numerais arábicos e romanos, palavras compostas, nomes próprios, caracteres especiais e de pontuação final. Os *tokens* são armazenados em estruturas de dados (*arrays*) e enviados um a um para análise do módulo seguinte.

O terceiro módulo, chamado Identificador Morfológico, foi concebido para utilizar tecnologias adaptativas (Fig.2.1). O Identificador Morfológico é composto por um Autômato Mestre e um conjunto de submáquinas especialistas que acessam bases de dados com regras de acesso às bases de dados de classificações morfológicas. A prioridade é obter as classificações do corpus Bosque [13]; caso o termo procurado não seja encontrado, o Identificador Morfológico procura por substantivos, adjetivos e verbos, no formato finito e infinito (flexionados e não flexionados), através de uma submáquina de formação e identificação de palavras, que usa, como base, o vocabulário do TeP2.0 [14]; por fim, o Identificador Morfológico procura por termos invariáveis, ou seja, termos cuja classificação morfológica é considerada estável pelos linguistas, tais como, conjunções, preposições e pronomes, no caso, extraídos no léxico do Portal São Francisco [15].

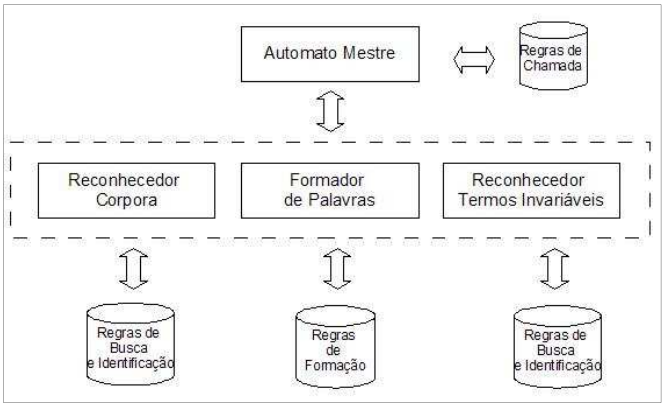


Figura 2.1. Arquitetura do Identificador Morfológico.

O Autômato Mestre (Fig.2.2) é responsável pelo sequenciamento das chamadas às submáquinas, de acordo com um conjunto de regras cadastradas em base de dados. Ele inicia o processamento recebendo o token da coleção de tokens criada pela classe Texto. Em seguida, antes de processá-lo, o autômato se modifica através de uma função adaptativa, criando uma submáquina de processamento (M1), uma transição entre o estado 1 e M1, e uma transição que aguarda o estado final da submáquina M1. O token é passado para M1 e armazenado em uma pilha. Quando M1 chega ao estado final, o autômato se modifica novamente, criando uma nova submáquina M2, o estado 2 e as transições correspondentes. Caso o estado final de M1 seja de aceitação, o processo é finalizado no estado 2, caso contrário a máquina M2 é chamada, passando o token armazenado na pilha.

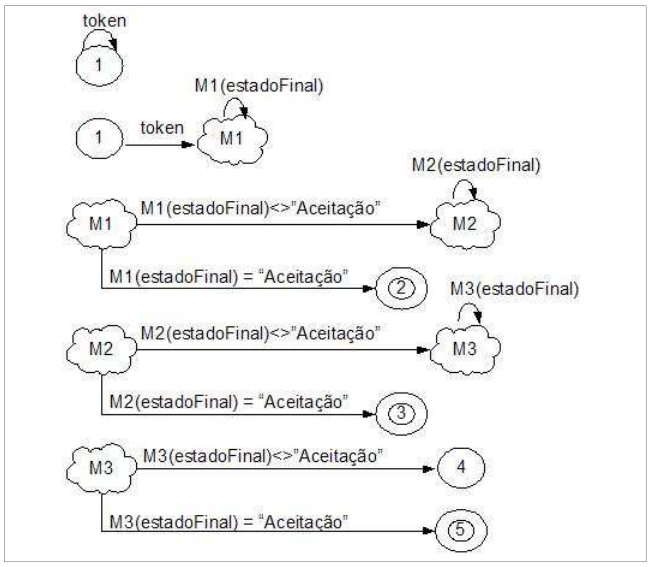


Figura 2.2. Estrutura Adaptativa do Autômato Mestre.

O processo se repete quando M2 chega ao final do processamento, com a criação da submáquina M3, do estado 3 e das transições correspondentes. Um novo ciclo se repete, e se o estado final de M3 é de aceitação, o autômato transiciona para o estado 5, de aceitação, caso contrário, ele vai para o estado 4 de não aceitação. M1, M2 e M3 representam, respectivamente, as submáquinas do Reconhecedor de Corpus, do Formador de Palavras e do Reconhecedor de Termos Invariáveis. Portanto, caso o Autômato Mestre encontre a classificação morfológica ao final de M1, ele não chama M2; caso encontre em M2, não chama M3 e, caso também não encontre em M3, ele informa aos demais módulos do Linguístico que não há classificação morfológica para o termo analisado.

As submáquinas M1, M2 e M3 também foram projetadas de acordo com a tecnologia adaptativa. A Máquina M1 usa um autômato adaptativo que se auto modifica de acordo com o tipo de token que está sendo analisado: palavras simples, palavras compostas, números, valores e símbolos. A Fig. 2.3 apresenta a estrutura adaptativa de M1.

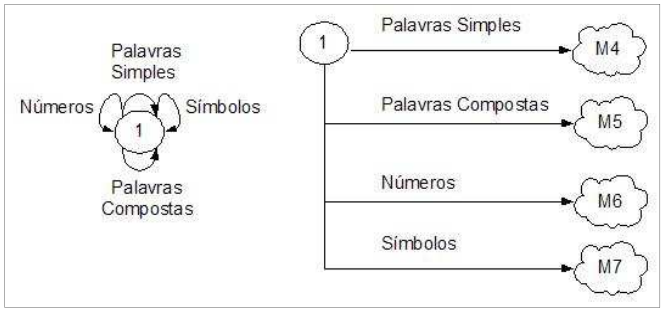


Figura 2.3. Estrutura Adaptativa do Reconhecedor de Corpus M1.

Inicialmente o autômato é composto por um único estado e por transições para ele mesmo (Fig.2.3, à esquerda). Ao identificar o tipo de token que será analisado (obtido no processo de tokenização), o autômato cria submáquinas e as transições correspondentes. As alternativas de configuração

são apresentadas na Fig.2.3, à direita. As submáquinas M4, M5, M6 e M7 reconhecem os tokens através de outro tipo de autômato que processa os tokens byte a byte (Fig. 2.4).

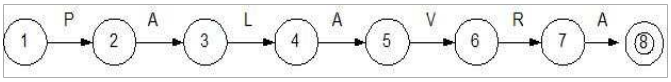


Figura 2.4. Reconhecedor de Palavras Simples.

No exemplo apresentado na Fig.2.4, o token “Palavra” é processado pela máquina M4; se o processamento terminar em um estado de aceitação, o token é reconhecido. As submáquinas M4, M5, M6 e M7 são criadas previamente por um programa que lê o Corpus e o converte em autômatos finitos determinísticos. A estrutura de identificação morfológica é composta por um par [chave, valor], no qual a chave é o estado de aceitação do elemento lexical e, o valor, sua classificação morfológica. No exemplo apresentado, a chave do item lexical “palavra” seria o estado ”8”, e, o valor, a classificação morfológica, H+n (substantivo, núcleo de sintagma nominal), proveniente do Corpus Bosque.

O Formador de Palavras, submáquina M2, também é montado previamente por um programa construtor, levando em consideração as regras de formação de palavras do Português do Brasil, descritas por Margarida Basílio em [16]. A submáquina M2 utiliza o vocabulário do TeP2.0 (formado por substantivos, adjetivos e verbos), como léxico de formas já feitas e o conjunto de regras de prefixação, sufixação e regressão verbal descrito pela autora para construir novas formas. No entanto, são necessários alguns cuidados para evitar a criação de estruturas que aceitem palavras inexistentes. A Fig. 2.5 apresenta um exemplo de autômato no qual são aplicados os prefixos “a” e “per”, e os sufixos “ecer” e “ar” (derivação parassintética) ao radical “noit” do substantivo noite, que faz parte do TeP2.0.

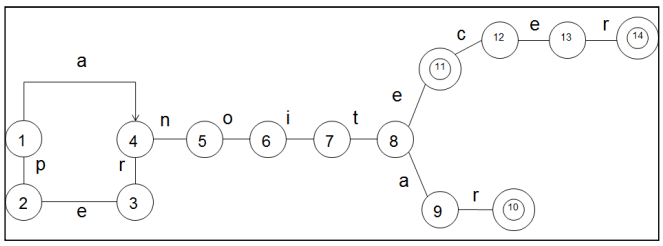


Figura 2.5. Autômato Formador de Palavras.

No exemplo apresentado, as palavras “anoitecer”, “pernoitar” e “anoitar” (sinônimo de “anoitecer”) existem no léxico do português do Brasil. Já pernoitecer é uma combinação que não existe na língua portuguesa. Margarida Basílio diz que algumas combinações não são aceitas simplesmente porque já existem outras construções consagradas pelo uso [17]. Para reduzir o risco de aceitar derivações inexistentes, o processo de construção do autômato restringe as possíveis formações, utilizando apenas as regras que a autora destaca como sendo mais prováveis. É o caso de nominalização de verbos com o uso dos sufixos –ção , –mento e – da. Em [16], Basílio cita que o sufixo –ção é responsável por 60% das formações regulares, enquanto o sufixo –mento é responsável por 20% destas formações. Já o sufixo –da é, via de regra, usado em nominalizações de verbos de movimento, tais como, entrada, saída, partida, vinda, etc.

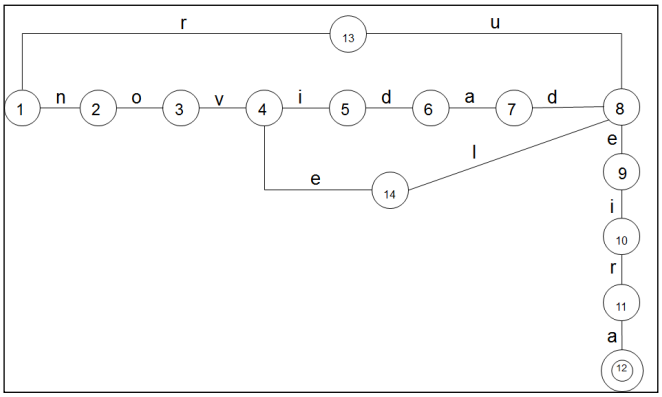


Figura 2.6. Autômato Formador de Palavras.

Outra característica do construtor do autômato é representar os prefixos e sufixos sempre pelo mesmo conjunto de estados e transições, evitando repetições que acarretariam o consumo desnecessário de recursos computacionais. A Fig. 2.6 apresenta exemplo de palavras formadas por reutilização de estados e transições na derivação das palavras rueira, novidadeira e noveleira. Os estados e transições usados para representar o sufixo “eira”, usado para designar são os mesmos nas três derivações.

A submáquina M2 também reconhece palavras flexionadas, obtidas, no caso de substantivos e adjetivos, através da aplicação de sufixos indicativos de gênero, número e grau aos radicais do vocabulário TeP2.0. A Fig. 2.7 apresenta um exemplo de autômato usado para a formação das formas flexionadas do substantivo menino. Foram adicionados ao radical “menin” as flexões “o(s)”, “a(s)”, “ão”, “ões”, “onona(s)”, “inho(s)” e“inha(s)”.

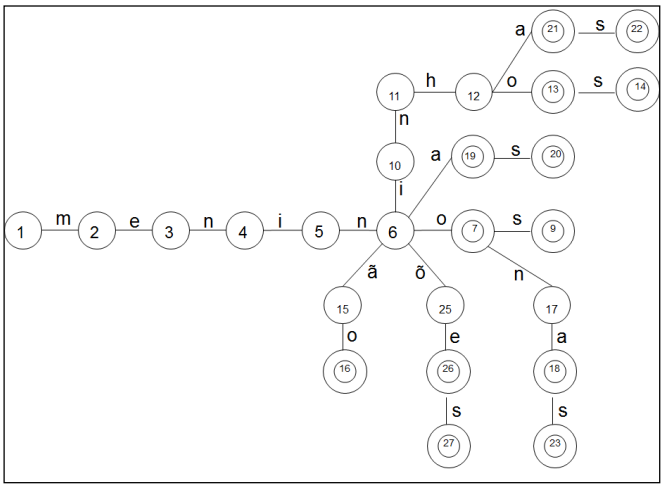


Figura 2.7. Autômato Formador de Flexões Nominais.

No caso de verbos, foi criada uma estrutura de estados e transições para representar as flexões de tempo, modo, voz e pessoa, obtendo-se, assim, as respectivas conjugações.

A Fig. 2.8 apresenta um exemplo de autômato usado para a formação das formas flexionadas do presente do indicativo do

verbo andar. Foram adicionados ao radical “and” as flexões “o”, “as”, “a”, “andamos”, “ais” e “andam”.

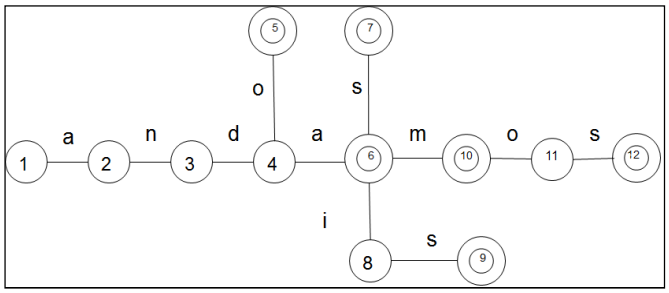


Figura 2.8. Autômato Formador de Flexões Verbais.

A estrutura de armazenamento da submáquina M2 também é composta por um par [chave, valor], no qual a chave é o estado de aceitação do elemento lexical e, o valor, sua classificação morfológica, acrescida da origem do termo, indicando que ele foi gerado pelo construtor. Por exemplo, o termo “novidadeira” seria classificado como H+n+C – substantivo, núcleo de sintagma nominal, gerado pelo construtor.

Já a submáquina M3 é um autômato que varia em função do tipo de termo (conjunções, preposições e pronomes) e utiliza uma estrutura arbórea similar a M4. A Fig. 2.9 apresenta um exemplo de autômato usado no reconhecimento de termos deste domínio.

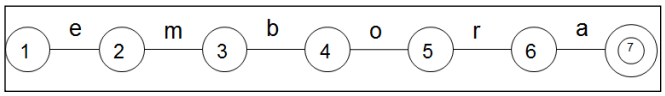


Figura 2.9. Autômato Reconhecedor de Conjunções, Preposições e Pronomes.

A estrutura de armazenamento da submáquina M3 é composta por um par [chave, valor], no qual a chave é o estado de aceitação do elemento lexical e, o valor, sua classificação morfológica, acrescida da origem do termo, indicando que ele faz parte da base de dados de apoio do Portal São Francisco. Por exemplo, o termo “embora” seria classificado como cj+Ba –conjunção da base de dados de apoio.

O Autômato Mestre também é responsável por desambiguar as classificações morfológicas e informar ao Agrupador apenas as que forem mais adequadas. Como as palavras podem ter mais do que uma classificação morfológica, é necessário utilizar uma técnica para selecionar aquela que é mais apropriada para o contexto analisado. Por exemplo, a palavra “casa” pode ser classificada como substantivo comum, feminino, singular ou como verbo flexionado na 3ª pessoa do singular no tempo presente, modo indicativo. No entanto, tendo em vista o contexto em que palavra se encontra, é possível escolher a classificação mais provável. Por exemplo, se a palavra “casa” vier precedida de um artigo definido, é mais provável que ela seja um substantivo; já se a palavra antecessora for um substantivo próprio, é mais provável que “casa” seja um verbo.

Collins [18] apresenta um modelo estatístico que leva em consideração 2 classificações anteriores à palavra analisada para fazer a desambiguação, criando distribuições nas quais a etiqueta de máxima probabilidade é o resultado da função:

P = max p(E,S), sendo:
E = etiquetas
S = palavras da sentença
p = probabilidade de E, dado S
max = máxima probabilidade

Por exemplo, para etiquetar a frase “O advogado entrevista a testemunha”, a função receberia as palavras da sentença e as sequências de etiquetas possíveis para elas (obtidas a partir de um Corpus de testes), calculando, então, a sequência de maior probabilidade. No entanto, Collins alerta que a complexidade para executar tal função inviabiliza sua utilização, pois ela cresce exponencialmente em função do número de palavras da sentença (Em uma sentença de “n” palavras e “K” possíveis classificações, existiriam $|K|^n$ possíveis etiquetas de sequencia).

Para evitar o problema da complexidade da execução, Collins propõe o uso do algoritmo Viterbi [19]. O algoritmo Viterbi é usado para encontrar a sequência mais provável de estados ocultos que resultam da observação de uma sequência de eventos. No caso da identificação das etiquetas morfológicas, os eventos são as palavras do texto analisado e os estados são as etiquetas de identificação morfológica. O algoritmo Viterbi pode ser implementado por meio de um autômato finito determinístico. No entanto, um autômato com estas características poderia ficar muito grande e, consequentemente, lento, devido ao tamanho do léxico usado para montá-lo. Uma alternativa para evitar este tipo de problema, é usar a tecnologia adaptativa. A Fig. 2.10 apresenta a estrutura do autômato que implementa o algoritmo Viterbi usado pelo desambiguador morfológico do Linguístico. O autômato recebe como parâmetro de entrada a palavra que está sendo analisada e monta dinamicamente os estados e transições de acordo com as possíveis etiquetas e as respectivas probabilidades, identificando, ao final do processamento, a etiqueta mais provável.

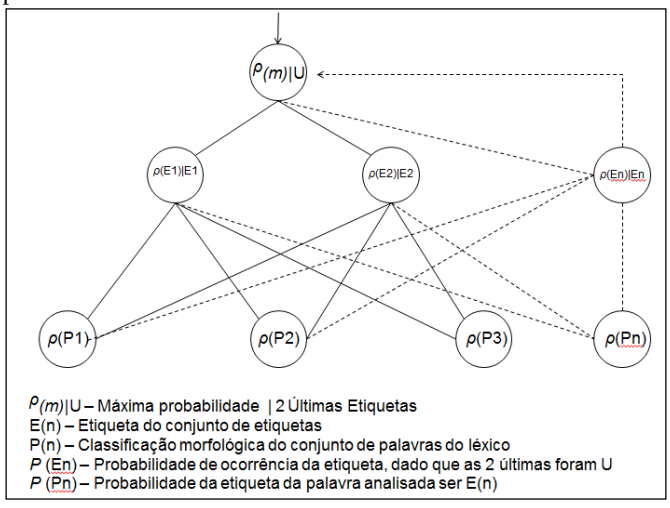


Figura 2.10. Autômato Adaptativo Viterbi.

O quarto módulo, denominado Agrupador é composto de um autômato, responsável pela montagem dos sintagmas a partir de símbolos terminais da gramática e um bigrama, responsável pela montagem dos sintagmas a partir de não-terminais (Fig.3). Inicialmente, o Agrupador recebe do Identificador as classificações morfológicas dos *tokens* e as agrupa em sintagmas de acordo com as regras de Luft. Neste processo são identificados sintagmas nominais, verbais, preposicionais, adjetivos e adverbiais Para isso, o Agrupador utiliza um

autômato adaptativo cuja configuração completa é definida da seguinte forma:
Estados = {1, 2, 3, 4, SS, SP, V, Sadj, Sadv, A},
Onde:
1,2,3 e 4 = Estados Intermediários
SS, SP, V Sadj, Sadv = Estados nos quais houve formação de sintagmas, sendo:
SS= Sintagma substantivo
SP = Sintagma preposicional
V = Verbo ou locução verbal
Sadj = Sintagma adjetivo
Sadv = Sintagma adverbial
A = Estado após o processamento de um ponto final
Tokens = {art, num, n, v, prp, pron, conj, adj, adv, rel, pFinal, sClass}, onde:
art = artigo, num = numeral
n = substantivo, v = verbo
prp = preposição, pron = pronome
conj = conjunção, adj = adjetivo
adv = advérbio, rel = pronome relativo
pFinal = ponto final, sClass = sem classificação
Estados de Aceitação = {SS, SP, V, Sadj, Sadv, A}
Estado Inicial = {1}
Função de Transição = {(Estado, *Token*)→Estado}, sendo:
{(1, art)→2, (2, art)→2, (3, art)→3
(1, num)→Sadv, (2, num)→2, (3, num)→3
(1, n)→SS, (2, n)→SS, (3, n)→SP
(1, v)→SV, (2, v)→ SV, (3, v)→SP
(1, prp)→3, (2, prp)→2, (3, prp)→3
(1, prop)→SS, (2, prop)→SS, (3, prop)→SP
(1, pron)→SS, (2, pron)→SS, (3, pron)→SP
(1, conj)→conj, (2, conj)→Ø, (3, conj)→ Ø
(1, adj)→Sadj, (2, adj)→Sadj, (3, adj)→3
(1, adv)→Sadv, (2, adv)→2, (3, adv)→3
(1, rel)→conj, (2, rel)→ Ø, (3, rel)→conj
(1, pFinal)→A, (2, pFinal)→ Ø, (3, pFinal)→ Ø}

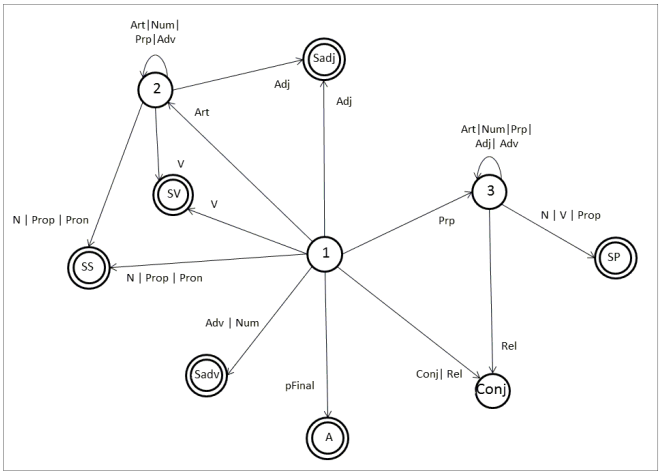


Figura 3. Configuração Completa do Autômato Construtor de Sintagmas.

Por exemplo, no caso de sintagmas substantivos teríamos:
SS → [Quant | PrA] (Sc | Sp | PrPes)
Pela regra acima, o conjunto de *tokens* "A" e "casa" formam um sintagma substantivo, da seguinte forma:
PrA = "A" (artigo definido)
Sc = "casa" (substantivo comum)
Da direita para esquerda, são realizadas as seguintes transições:
PrA Sc → SS; A Sc → SS; A casa → SS
Já o Agrupador recebe o *token* "A", identificado pelo Tokenizador como artigo definido, e se movimenta do estado 1 para o estado 2. Ao receber o *token* "casa", identificado como substantivo comum, ele se movimenta do estado 2 para o estado SS, que é um estado de aceitação. Neste momento o Agrupador armazena a cadeia "A casa" e o símbolo "SS" em uma pilha e reinicializa o autômato preparando-o para um novo reconhecimento. Em um passo seguinte, o Agrupador usa o bigrama para comparar um novo sintagma com o último sintagma formado, visando identificar elementos mais altos na hierarquia. Para isso ele usa a matriz apresentada na Tabela 3. A primeira coluna da matriz indica o último sintagma formado (US) e a primeira linha, o sintagma atual (SA). A célula resultante apresenta o novo nó na hierarquia.

TABELA 3.
MATRIZ DE AGRUPAMENTO DE SINTAGMAS

SA / US	SS	SP	V	Sadv	Sadj	Conj
SS	SS	SS	-	-	SS	-
SP	SP	-	-	-	-	-
V	-	-	V	-	-	-
Sadv	-	-	-	Sadv	Sadj	-
Sadj	SS	Sadj	-	-	Sadj	-
Conj	-	-	-	-	-	Conj

Esta técnica foi usada para tratar as regras gramaticais nas quais um sintagma é gerado a partir da combinação de outros, como é o caso da regra de formação de sintagmas substantivos: SS → [Sadj] SS [Sadj | SP]. Por esta regra, os sintagmas substantivos são formados por outros sintagmas substantivos precedidos de um sintagma adjetivo e seguidos de um sintagma adjetivo ou um sintagma preposicional. No exemplo anterior, supondo que os próximos 2 *tokens* fossem "de" e "madeira", após a passagem pelo autômato, o Agrupador formaria um sintagma SP. Considerando que na pilha ele tinha armazenado um SS, após a passagem pelo bigrama, e de acordo com a Tabela 3, o sintagma resultante seria um SS e o conteúdo que o compõe seria a combinação dos textos de cada sintagma que o originou. Caso não haja agrupamentos possíveis, o Agrupador envia o último sintagma formado para análise do Reconhecedor Sintático e movimenta o sintagma atual para a posição de último sintagma no bigrama, repetindo o processo com o próximo sintagma.

O quinto e último módulo, denominado Reconhecedor Sintático, recebe os sintagmas do módulo anterior e verifica se estão sintaticamente corretos de acordo com padrões oracionais de Luft. O Reconhecedor Sintático utiliza um autômato adaptativo que faz chamadas recursivas sempre que recebe conjunções ou pronomes relativos, armazenando, em uma estrutura de pilha, o estado e a cadeia de sintagmas reconhecidos até o momento da chamada. Caso o Reconhecedor Sintático não consiga se movimentar a partir do sintagma recebido, ele gera um erro e retorna o ponteiro para o

último sintagma reconhecido, finalizando a instância do autômato recursivo e retornando o processamento para aquela que a inicializou. Esta, por sua vez, retoma posição em que se encontrava antes da chamada e continua o processamento até o final da sentença ou até encontrar uma nova conjunção, situação na qual o processo se repete.

A configuração completa do autômato é definida da seguinte forma:

Estados = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27}

Tokens = {SS, SP, Vli, Vi, Vtd, Vti, Vtdi, Sadj, Sadv, Conj, A}

Estados de Aceitação = { 4, 5, 6, 9, 12, 13,14, 15, 17, 18, 19, 21, 22, 24, 25, 26, 27}

Estado Inicial = {1}

Função de Transição = {(Estado, *Token*)→Estado}, sendo:

{(1, SS)→2, (2, Vti)→3, (3, SP)→4, (4, SP)→4,
(3, Sadv)→5, (2, Vi)→6, (2, Vtdi)→7
(7, SS)→8, (8, SP)→9, (9, SP)→9, (8, Sadv)→10,
(2, Vlig)→11, (11, SP)→12, (11, Sadv)→13,
(11, Sadj)→ 14, (11, SS)→15, (2, Vtd)→16, (16, SS)→17,
(2, Vtpred)→18, (18, SP)→19, (18, Sadj)→20
(18, SS)→ 21, (21, SS)→22, (21, Sadj)→23, (21, Sadv)→24,
(21, SP)→25}

Pilha = {[Texto, Sintagma, Estado]}

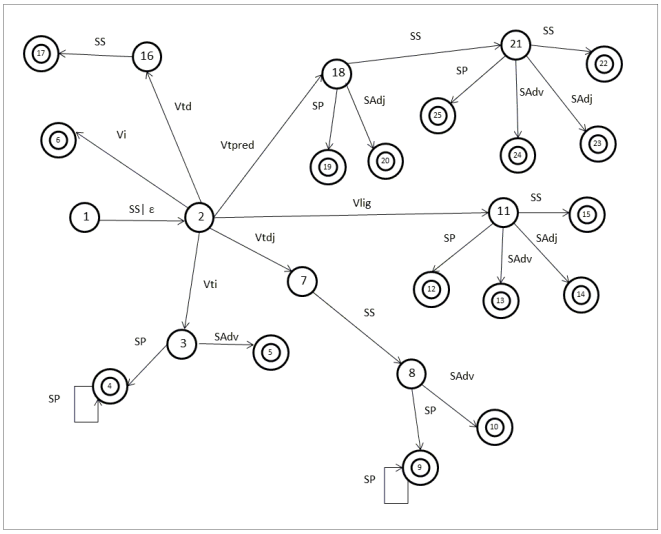


Figura 4.1. Configuração Completa do Reconhecedor Sintático.

No entanto, para que a análise sintática seja feita, não são necessárias todas as ramificações da configuração completa do autômato (Fig. 4.1). Por exemplo, quando se transita um verbo de ligação a partir do estado 2, o autômato vai para o estado 11 e todas as demais ramificações que partem deste estado para os estados 3, 7, 16 e 18, não são usadas. Com a tecnologia adaptativa, é possível criar dinamicamente os estados e transições do autômato em função dos tipos de verbos, evitando manter ramificações que não são usadas.

A Fig. 4.2 apresenta a configuração inicial do autômato adaptativo equivalente ao autômato de pilha apresentado anteriormente. No estado 1, o autômato recebe os *tokens* e transita para o estado 2 quando processa um sintagma substantivo (SS) ou quando transita em vazio. No estado 2, o autômato transita para si mesmo quando recebe qualquer tipo de verbo: Vi, Vtd, Vlig, Vtpred, Vtdi e Vti. Todas as outras ramificações são criadas por meio de funções adaptativas chamadas em função do tipo de verbo processado.

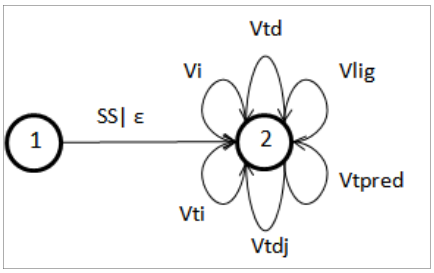


Figura 4.2. Configuração Inicial do Reconhecedor Gramatical.

Por exemplo, se o verbo é de ligação (Vlig), o autômato utiliza as funções adaptativas α (j) e β (o), definidas da seguinte forma:

α (j): { o* :
- [(j, Vlig)]
+[(j, Vlig) :→ o, β (o)]
}

β (o): { t*u*v*x*:
+ [(o, SP) :→ t]
+ [(o, Sadv) :→ u]
+ [(o, Sadj) :→ v]
+ [(o, SS) :→ x]
}

A função adaptativa α (j) é chamada pelo autômato antes de processar o *token*, criando o estado 11 e a produção que leva o autômato do estado 2 ao novo estado criado. Em seguida, o autômato chama a função β (o), criando os estados 12, 13, 14 e 15 e as produções que interligam o estado 11 aos novos estados. A Fig. 4.3 mostra a configuração do autômato após o processamento do verbo de ligação. Neste exemplo, o autômato criou apenas os estados 11, 12, 13, 14 e 15 e as respectivas transições, evitando alocar recursos que seriam necessários para criar o autômato completo, conforme apresentado na Fig. 4.1.

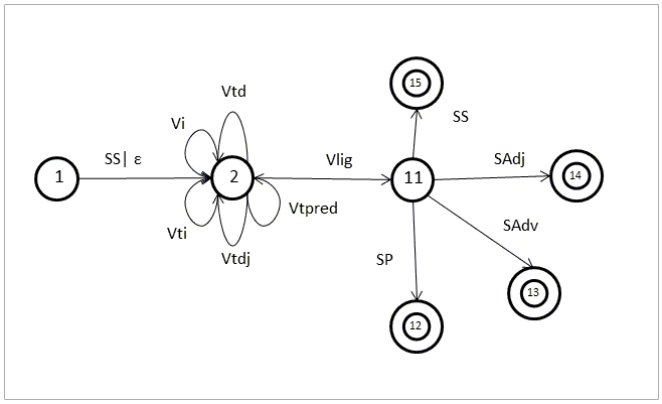


Figura 4.3. Configuração do autômato após o processamento do verbo de ligação.

V. DESAMBIGUAÇÃO SINTÁTICA E SEMÂNTICA

Iwai [20] apresenta um formalismo, denominado Gramáticas Adaptativas, no qual as regras gramaticais são criadas dinamicamente a partir do processamento da cadeia de entrada e de informações relacionadas ao contexto em que se apresentam, permitindo a identificação e resolução de ambiguidades semânticas. A autora demonstra, ainda, a

equivalência computacional entre as Gramáticas Adaptativas e os autômatos adaptativos, o que permite que frases da linguagem gerada por uma gramática adaptativa sejam reconhecidas por autômatos adaptativos.

O Linguístico incorpora o formalismo proposto pela autora, através do mapeamento dos sintagmas e padrões oracionais de Luft para uma gramática adaptativa. As ações adaptativas são usadas para modificar as regras de produção em função do contexto definido pelo tipo do verbo encontrado no texto e pela presença de conjunções ou pronomes relativos. O resultado final é o seguinte:

$G = (G^0, T, R^0)$, onde:

$G^0 = (VN^0, VT, VC, PL^0, PD^0, S)$

R^0 = relação que associa regras de produção às ações adaptativas.

VN0 = { S, con, SS, SV, SS, Sadj, SP,num, PrA, Sc, Sp, PrPes, Neg, Aux, PreV,V, Vlig ,Vtda,Vtdna , Vtdi, Vtpred ,Vti ,Vi, Sadv, Prep, Ind, ArtDef, ArtInd, Dem, Pos, PrRel, Adv, Adj, pont }

VT = {{léxico},{,}, ; , . ,:,!,?,1, 2, 3, 4, 5, 6, 7,8, 9, 0} V_C = ∅, P_{D0} = ∅

PL0 = { S → [con] [PrRel] [SS] SV pont [S]

SS → [Sadj] SS [Sadj | SP] [S] | S

SS → (([num | PrA] Sc) | Sp | PrPes)

SV→ [Neg] [Aux | PreV] V [SS | Sadj | Sadv | SP] | ((SS SP) | (SS Sadj) | (SP SP) | (SS SS) | (SS Sadv) | (SS SP SP))

V → (Vlig | Vtda | Vtdna | Vtdi | Vtpred |Vti |Vi)

SP → Prep (SS | Sadj)

Sadj → Sadj [SP] [S] | S

Sadj → [Adv] Adj [S]

Sadv → Sadv [SP] [S] | S

Sadv → [Adv] Adv [S]

PrA → [PrA] (Ind | ArtDef | ArtInd| Dem| Pos)

con → (aditiva | adversativa | alternativa | conclusiva| explicativa | integrante | causal | comparativa | concessiva | condicional | conformativa | consecutiva | final | proporcional | temporal)

num→ (0 |1 |2 | 3 |4 |5 |6 |7 |8 |9)| (cardinal| ordinal| multiplicativo| fracionario)

pont → (,| ; | . |: |! |?)}

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

A análise semântica das sentenças é feita de acordo com os termos processados, que podem gerar novas regras de produção e exclusão de regras existentes. Por exemplo, quando o Linguístico encontra um verbo transitivo indireto, ele dispara uma ação adaptativa A (Vti) que elimina regras iniciais definidas na gramática G^0 e cria outras, relacionadas aos complementos oracionais exigidos por este tipo de verbo. A representação destas alterações é feita da seguinte forma:

$$A\ (t = Vti) = L\ (t) = \{ + [R: {}_G{}^0 P_{26\text{-}66,\ 69,77\text{-}78,\ 87,\ 93,\ 101\text{-}102,\ 113\text{-}114}: {}_G{}^1 V \leftarrow \chi \backslash V] + [{}_{G1}\chi V \rightarrow Vti] - [R: {}_G{}^0 P_{67\text{-}68,70\text{-}76,79\text{-}86,\ 88\text{-}100,103\text{-}112,115\text{-}120}: {}_G{}^1 \varnothing]$$

A sintaxe da instrução R é a seguinte:

R: $<{}_G{}^n> <P_{I\text{-}F}> : <{}_G{}^{n+1}> P'$ (4)

onde:

R: replace

${}_G{}^n$: gramática anterior à atualização

$P_{I\text{-}F}$: regras de produção para atualização, sendo I = Inicio de intervalo e F= fim de intervalo. Acrescenta-se mais de um grupo de regras usando a vírgula para separação.

${}_G{}^{n+1}$: gramática posterior à atualização

P': nova regra de produção

∅ : símbolo que indica que as regras assinaladas em $_{PI\text{-}F}$ serão removidas

O formalismo apresentado por Iwai é bastante prático e fornece suporte para incluir características adicionais que o parser pode utilizar para resolver problemas em que o contexto seja importante para definição das regras de produção. Por exemplo, o parser pode verificar se a frase analisada está correta ou resolver possíveis ambiguidades caso tenha à disposição informações sobre o tipo do verbo, tempo, modo e pessoa do verbo analisado. No exemplo abaixo, a ação adaptativa A utiliza como parâmetros de entrada a identificação do tipo do verbo, modo e pessoa para definição das funções adaptativas.

T = { A (verbo=Vlig, tempo = Presente, modo = Singular, pessoa = Primeira) = F (verbo, tempo, modo, pessoa) =
{ + [R: ${}_G{}^n P_{inicio\text{-}fim}$: ${}_G{}^n$+1V ← χV]
+ [${}_G{}^{n+1}$χV → Vlig, Presente, Singular, Primeira]
+ [R: ${}_G{}^n P_{inicio\text{-}fim}$: ${}_G{}^{n+1}$ ∅] }

Analogamente, é possível incluir regras para análise da concordância nominal ou qualquer outro tipo de conteúdo semântico, sem necessidade de usar qualquer outro elemento

fora do formalismo. As gramáticas adaptativas também podem ser usadas para representar o uso de informações probabilísticas na seleção das regras de produção. Tal técnica é usada quando existe mais de uma regra de produção aplicável e não existem informações sintáticas e semânticas suficientes para desambiguá-las. Neste caso, é possível usar a probabilidade de ocorrência das regras como fator de escolha e o formalismo de Iwai pode ser usado representar este tipo de informação contextual. No exemplo abaixo, a ação adaptativa A utiliza como parâmetro de entrada a probabilidade de ocorrência das regras de produção avaliadas e uma indicação para usar a regra de máxima probabilidade.

SVprob → V SP 10%
SVprob → V (SS SP) 90%

T = { A (t=SV, u = prob, v=max) = F (SV, prob, max) =
{ + [R: $_G^n$ $P_{inicio-fim}$: $_G^{n+1}$SV ← χSVprob]
+ [$_G^{n+1}$ $\chi SV_{prob} \rightarrow \chi SV_{max}$]
? [$_G^{n+1}$ χSV_{max}]
+ [$_G^{n+1}$ χSV_{max} → V (SS SP)]
+ [R: $_G^n$ $P_{inicio-fim}$: $_G^{n+1}$ ∅] }

VI. CONSIDERAÇÕES FINAIS

Este artigo apresentou uma revisão dos conceitos de Tecnologia Adaptativa e de Processamento da Linguagem Natural. Em seguida, foi apresentado o Linguístico, uma proposta de reconhecedor gramatical que utiliza autômatos adaptativos como tecnologia subjacente. Por fim, procurou-se apresentar cenários em que a gramática utilizada pelo Linguístico é estendida, incorporando critérios semânticos para auxiliar na resolução de ambiguidades.

O Linguístico está fase de desenvolvimento e testes preliminares confirmaram a tecnologia adaptativa como uma alternativa válida para o processamento da linguagem natural, proporcionando economia de recursos computacionais e flexibilidade para implantação de novas regras, o que nos motiva a aprofundar a pesquisa, provendo embasamento quantitativo para chegar a respostas mais conclusivas.

REFERÊNCIAS

[1] http://www.pcs.usp.br/~lta/
[2] C. Taniwaki. Formalismos adaptativos na análise sintática de Linguagem Natural . Dissertação de Mestrado, EPUSP, São Paulo, 2001.
[3] C. E. Menezes. Um método para a construção de analisadores morfológicos, aplicado à língua portuguesa, baseado em autômatos adaptativos. Dissertação de Mestrado, Escola Politécnica da Universidade de São Paulo, 2000.
[4] D. Padovani. Uma proposta de autômato adaptativo para reconhecimento de anáforas pronominais segundo algoritmo de Mitkov. Workshop de Tecnologias Adaptativas –WTA 2009, 2009.
[5] M. Moraes. Alguns aspectos de tratamento sintático de dependência de contexto em linguagem natural empregando tecnologia adaptativa, Tese de Doutorado, Escola Politécnica da Universidade de São Paulo, 2006.
[6] E. Rich and K. Knight. Inteligência Artificial, 2. Ed. São Paulo: Makron Books, 1993.
[7] R. Vieira and V. Lima. Linguística computacional: princípios e aplicações. IX Escola de Informática da SBC-Sul, 2001.
[8] C. Fuchs and P. Le Goffic. Les Linguistiques Contemporaines.
[9] M. G. V.Nunes et al. Introdução ao Processamento das Línguas Naturais. Notas didáticas do ICMC Nº 38, São Carlos, 88p, 1999. Paris, Hachette, 1992. 158p.
[10] T. B. Sardinha, A Língua Portuguesa no Computador. 295p. Mercado de Letras, 2005.
[11] R.L.A. Rocha. Tecnologia Adaptativa Aplicada ao Processamento Computacional de Língua Natural. Workshop de Tecnologias Adaptativas – WTA 2007, 2007.
[12] C. Luft. Moderna Gramática Brasileira. 2ª. Edição Revista e Atualizada. 265p. Editora Globo, 2002.
[13] http://www.linguateca.pt/
[14] http://www.nilc.icmc.usp.br/tep2/
[15] http://www.portalsaofrancisco.com.br/alfa/materias/index-lingua-portuguesa.php/
[16] M. Basilio. Formação e Classes de Palavras no Português do Brasil. Ed.Contexto, 2004.
[17] M. Basilio. Teoria Lexical. Ed.Atica, 1987.
[18] http://www.cs.columbia.edu/~mcollins/hmms-spring2013.pdf/
[19] G.D. J. Forney. IEEE. Proceedings of the IEEE, Volume 61 , Issue 3,1973.
[20] M., Iwai: Um Formalismo Gramatical para Linguagens Dependentes de Contexto. São Paulo, 2000. Escola Politécnica da Universidade de São Paulo. Tese de Doutorado (2000).

Ana Teresa Contier formou-se em Letras-Português pela Universidade de São Paulo (2001) e em publicidade pela PUC-SP (2002). Em 2007 obteve o título de mestre pela Poli-USP com a dissertação: "Um modelo de extração de propriedades de textos usando pensamento narrativo e paradigmático".

Djalma Padovani formou-se em administração de empresas pela Faculdade de Economia e Administração da Universidade de São Paulo, em 1987 e obteve o mestrado em engenharia de software pelo Instituto de Pesquisas Tecnológicas de São Paulo - IPT, em 2008. Trabalhou em diversas empresas nas áreas de desenvolvimento de software e tecnologia de informação e atualmente é atua no Laboratório de Dados da Serasa Experían.

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

Análise Semântica de Sentimentos Utilizando Árvores de Decisão Adaptativa

A. M. Silva & R. L. A. Rocha & J. José Neto

***Resumo*— Devido ao crescimento das redes sociais na Web, análise de sentimentos e mineração de opiniões são temas de pesquisas que cada vez mais tem se tornado frequente. O objetivo deste trabalho de pesquisa é aprofundar o estudo de análise de sentimentos, e propor um modelo prático da Roda das Emoções de Plutick juntamente com a utilização de Árvores de Decisão Adaptativa para classificação automática de sentimentos, de menções extraídas de textos da rede social Twitter. Neste sentido a classificação de sentimentos pode ser de grande ajuda, realizando a triagem de conteúdos de mensagens relevantes, descartando a informação desnecessária, vinculando mensagens com mesmo tema em comum, e até gerando métricas para classificação das emoções. Esta pesquisa é um exercício exploratório, com base no conjunto de Tweets de uma determinada ontologia, busca-se caracterizar indicadores linguísticos capazes de automaticamente, classificar e hierarquizar as emoções contidas nos sentimentos pelo método teórico Roda de Emoções de Plutchik utilizando os modelos explicitamente teóricos propostos: MFA – Modelo de Filtragem Adaptativa e o MCS – Modelo de Classificação de Sentimentos ambos dotados de ações adaptativas.**

***Keywords*— Análise de Sentimentos, Árvores de Decisão Adaptativas; Análise Semântica, Linguística de Corpus, Redes Sociais, Adaptatividade.**

I. INTRODUÇÃO

O crescimento de informações colocadas na Web por empresas de notícias, e usuários que compartilham suas ideias, opiniões pessoais e criticas, é evidente nas últimas décadas. A divulgação dessas informações se dá em blogs pessoais, microblogs, sites de relacionamento e principalmente em redes sociais como *Facebook* e o *Twitter*. Com esse crescimento de informações inseridas na Web pelos usuários, de forma que a informação divulgada é de opinião pessoal, isso despertou uma nova área de pesquisa chamada de análise de sentimentos ou mineração de opinião, que vem acompanhada da área de PLN - Processamento de Linguagem Natural e LC - Linguística de Corpus. Para extração, processamento e armazenamento das informações originadas nas redes sociais, utiliza-se de técnicas já empregadas dentro das áreas de PLN e LC.

O desafio consiste em descobrir padrões que possam resultar em uma consciência coletiva dos usuários de internet. E não simplesmente subjulgar um ou outro usuário por determinada opinião divulgada. Mas sim realizar análise semântica dos sentimentos e classificá-los em um nível de assertividade.

Motivação

A pesquisa interdisciplinar da comunicação e análise de sentimentos em uma determinada ontologia é motivada não só pela importância e efeito de causalidade da informação, mas também pela percepção de que o resultado das mensagens divulgadas na rede social gera um problema complexo de se lidar. O modo como à informação é distribuída afeta outros indivíduos, que por sua vez, são seguidores do indivíduo que iniciou a primeira divulgação na rede social. Os indivíduos influenciados podem ou não compartilhar da mesma opinião da mensagem inicial. Avaliar e extrair medidores deste tweets são desafios deste trabalho de pesquisa. Porém, a dimensão de informações disponíveis nas redes sociais, dificulta a análise e classificação das informações nelas contidas, que graças à pessoalidade, não existe uma uniformidade, um padrão entre as mensagens de diversos indivíduos, aos mais variados assuntos postados. Devido a isso, tem-se a necessidade de encontrar métodos que possam utilizar essas informações e prover resultados eficazes.

Objetivo

A partir de sentimentos exibidos nos tweets, o objetivo principal é concluir se a população achou um fato positivo ou negativo, e também a classificação das emoções, tomando como base a Roda das Emoções de Plutchik, contidas nos sentimentos pré-classificados.

Objetivos Específicos

Com base no objetivo central do trabalho, os objetivos específicos são:

- selecionar tweets referentes à ontologia em questão;
- estabelecer um critério computacional, de modo que as mensagens sejam de relevância dentro da ontologia, descartando informações desnecessárias;
- caracterizar indicadores lingüísticos (léxico-semânticos) capazes de reconhecer se o tweet possui sentimentos/emoção dentro do domínio ontológico;
- descrever recursos computacionais adaptativos capazes de classificar os sentimentos e suas emoções de acordo com a Roda das Emoções de Plutchik;

Organização do Trabalho

A pesquisa será dividida em quatro grandes etapas, que serão desenvolvidas hierarquicamente. A primeira refere-se ao estudo da Análise de Sentimentos e das Ontologias. A segunda refere-se às tecnologias adaptativas que já foram empregadas em outros trabalhos de pesquisas e explorará a tecnologia adaptativa de árvores de decisão, e como ela pode ajudar no problema de classificação de sentimentos. A terceira refere-se à proposta de dois modelos encapsulados de ações adaptativas, o primeiro MFA – Modelo de Filtragem Adaptativa dotado se um automoto adaptativo simples e o segundo MCS – Modelo de Classificação de Sentimentos dotado de árvores de decisão

adaptativas. E a quarta etapa que irá explanar a conclusão deste trabalho de pesquisa.

Hipóteses

- Qual o tipo de emoção que o indivíduo mencionou ao deixar o seu tweet escrito?
- Como reconhecer automaticamente sentimentos de textos extraídos dos tweets?
- Como extrair automaticamente sentimentos de textos extraídos dos tweets?
- Como associar automaticamente se um tweet é positivo ou negativo?
- Como associar automaticamente os sentimentos contidos em um tweet na Roda das Emoções?

II. ANÁLISE DE SENTIMENTO

O foco da análise de sentimentos é identificar o sentimento que o usuário menciona a respeito de uma marca, uma entidade, um assunto de interesse, um lugar, um produto. Essa opinião explícita em formas de mensagens textuais divulgadas na internet pode ter sido formada a partir de conteúdos noticiosos também encontrados na Web, por *cookies* relacionados ao perfil do usuário, ou não. O usuário pode formar sua opinião a partir de conteúdos que buscou de forma não digital. Mas a maior parte das vezes, o usuário teve acesso à informação por meio digital, mas a continuidade e divulgação de forma liberal de sua opinião serão concebidas de forma digital, e estudos mostram que cada vez mais os indivíduos utilizam as redes sociais.

Grupos, empresas e pesquisadores querem saber a opinião coletiva de determinado assunto. Por exemplo, o lançamento de um novo produto por uma empresa, posteriormente essa empresa irá acompanhar as opiniões dos usuários e relacionar se aquele novo produto teve aceitação ou não, e qual foi o nível dessa aceitação. Será preciso monitorar essas opiniões divulgadas, realizar filtros específicos para encontrar exatamente as menções relacionadas aquele produto em especifico. A análise de sentimentos é também uma forma de mineração de dados que irá prover essa demanda.

Para realização da análise de sentimentos, é preciso também realizar o uso de ontologias, valores numéricos estatísticos que ajudarão a quantificar o quanto uma palavra indica ou não determinado sentimento. Em análise de sentimentos, não serão analisados o conteúdo expresso no texto como um todo, mas sim a classificação da emoção predominante no assunto de pesquisa emergente, que poderá ser positivo, negativo.

Análise de sentimentos utilizando algoritmos de aprendizagem de máquina e redes bayesianas utilizando textos extraídos do Twitter em língua espanhola foi proposto por Grigori et al em 2012 [1]; Em 2014, KANAVOS et al. [2] propôs um modelo de análise de sentimentos que mede a influencia do usuário na rede com os demais nós dessa rede; Em 2013, PORSHNEV et al. [3] propõem um modelo de previsão do estado psicológico dos usuários a partir dos textos do Twitter, e classifica as previsões utilizando o algoritmo DJIA e classifica-as em oito emoções.

Dificuldades Encontradas no Estudo de Análise de Sentimentos

Dentro da área de PLN, já é sabido que existem diversos problemas lexicais e sintáticos. Esses problemas que se referem à forma da língua natural não serão tratados neste momento, serão considerados como exceções. De forma que possamos focar no principal desafio, que é a análise semântica de sentimentos utilizando árvores adaptativas. Abaixo segue uma lista de dificuldades encontradas:

- textos escritos sintaticamente incorretos, dificultando as filtragens iniciais sólidas;
- distinção de textos opinativos, de textos factícios e de textos informativos;
- identificar dentro de um texto informativo, se existe opiniões embutidas;
- escrita com sarcasmos ou ironias;
- textos com um ou mais assuntos de interesse, e as várias opiniões descritivas no mesmo texto;
- escrita informal e abreviativa;
- conteúdo mentiroso, (ex: um "blogueiro" é pago para divulgar informações positivas de uma empresa e "contaminar" a rede social).

Aplicações Práticas

Abaixo cita-se alguns exemplos de aplicações práticas em que um analisador de sentimentos pode ser empregado:

Análise sobre pessoas: o analisador de sentimentos pode ser utilizado para verificar a opinião dos usuários sobre determinada pessoa, principalmente de pessoas que estão em destaque na mídia publicitária. Por exemplo, a aceitação ou rejeição de determinado político durante a época de eleições, e ofertando a oportunidade do próprio político utilizar as informações para alterar suas estratégias de campanha.

Análise de um produto: Essa análise é muito utilizada por empresas, para verificar as taxas de aceitação e rejeição no lançamento de um determinado produto, ou para acompanhar a curva de vendas de um produto carro chefe da linha.

Bolsa de Valores: Informações de determinada empresa que possui ações abertas na bolsa de valores são avaliadas para auxiliar na tomada de decisão dos próximos passos para investimentos, principalmente das empresas que estão em destaque na mídia.

Mecanismos de Busca

Abaixo cita-se alguns mecanismos de busca utilizados em extração de dados para análise de sentimentos:

Sites de Busca: é possível utilizar o Google ou o Bing, lembrando que essa busca se estenderia a qualquer site, o que torna difícil a padronização e extração dos textos, principalmente na indexação do corpus.

Facebook: Segundo SOCIAL TIMES (2014) [5], em dezembro de 2014 possuía 1,35 bilhões de usuários. Nesta rede social, os usuários podem compartilhar links e "likes" e demonstrar suas opiniões através de mensagens. Embora essa rede possa não ser tão trivial como o Twitter, pois existe uma barreira na obtenção dos dados.

Twitter: considerado um ótimo local, para obter informações sobre determinado assunto. Segundo o SOCIAL

TIMES (2014) [5], em dezembro de 2014 possuía 284 milhões de usuários. Os tweets em sua maioria são públicos, fornecendo uma base rica em informação e de fácil acesso. Uma extensão do Twitter, um site chamado Sentiment140 (http://www.sentiment140.com) [6], obtém informações de tweets recentes sobre determinado assunto e monta um gráfico analisando as opiniões de usuários. Na figura 1, é realizada a busca da palavra "dengue" na língua espanhola, o site classifica em 75% de menções negativas e 25% de menções positivas. Na figura 2, é realizada a busca da palavra "terrorism" na língua inglesa, o site classifica em 65% de menções negativas e 35% de menções positivas. Na figura 3, é realizada a busca da palavra "coca-cola" com letra minúscula na língua inglesa, o site classifica em 4% de menções negativas e 94% de menções positivas. Na figura 4, é realizada a busca da palavra "Coca-Cola" com letra maiúscula na língua inglesa, o site classifica em 16% de menções negativas e 84% de menções positivas.

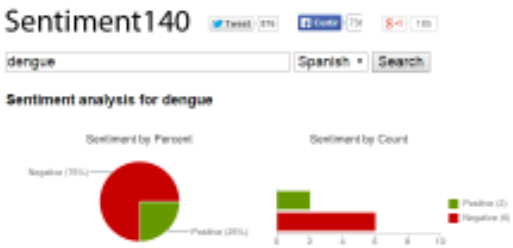


Figura 1: Análise de menções em língua espanhola com a palavra "dengue" (Sentiment140, 2015)

Figura 2: Análise de menções em língua inglesa com a palavra "terrorism" (Sentiment140, 2015)

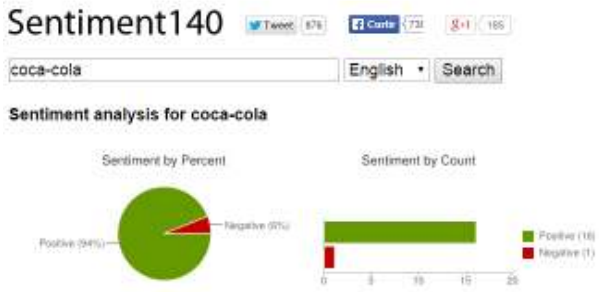


Figura 3: Análise de menções em língua inglesa com a palavra "coca-cola" em letra minúscula (Sentiment140, 2015)

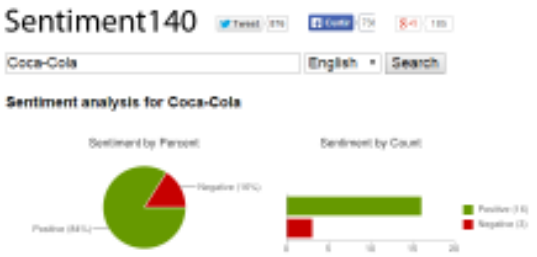


Figura 4: Análise de menções em língua inglesa com a palavra "Coca-Cola" em letra maiúscula (Sentiment140, 2015)

KWAK et al. [7], avalia em seu trabalho o potencial do Twitter como uma rede de informação, em seu estudo mostra que 85% dos tweets criados são relacionados a manchetes de jornais e notícias divulgadas na mídia. Também se pode considerar o fato do Twitter permitir em tempo real, o compartilhamento das mensagens, podendo então captar o sentimento dos indivíduos no momento em que ele passa a conhecer o assunto, a notícia, o anúncio em questão, o que faz com que o indivíduo demonstre a sua "emoção primária" antes mesmo que outros fatores possam vir a influencia-lo, causando até mesmo uma diminuição do sentimento gerado [8].

Roda das Emoções de Plutchik

Uma emoção não é simplesmente um estado de sentimento. Emoção é uma cadeia de eventos frouxamente ligados que começam com um estímulo e incluem sentimentos, alterações psicológicas, impulso para a ação e comportamento específico, dirigido por objetivos [4]. O efeito do estado emocional é criar uma interação entre o indivíduo e o evento ou estímulo que precipitou a emoção. Analisar o que os indivíduos escrevem nas redes sociais é então uma forma de descobrir que existe sim um evento, algo que tenha estimulado a exteriorização do seu pensamento, de suas emoções, de forma textual utilizando uma rede social. As emoções não são simplesmente eventos lineares, mas são processos de *feedback* que se dá internamente dentro do indivíduo. A função da emoção é restaurar o indivíduo a um estado de equilíbrio quando eventos inesperados criam as disfunções.

PLUTCHIK [4] menciona que existem centenas de palavras para classificar as emoções, e essas palavras tendem a criar grupos/famílias com mesma similaridade semântica. PLUTCHIK [4] descobriu que as emoções primárias podem ser conceituadas de forma análoga a uma roda de cores, colocando as emoções semelhantes juntas ou opostas, como se fosse cores complementares. Ele afirma que existem emoções que são misturas de duas emoções primárias, assim como algumas cores, que são misturas de cores primárias. Ele propõe a análise das emoções com mais uma terceira dimensão que representa a intensidade daquela emoção, o modelo teria então uma forma de um cone.

O *modelo simplístico* consiste em fazer a análise lexical das frases e classificá-las em dois tipos: Positivos e negativos. Essa análise é bastante simples de ser realizada, pois há apenas duas classificações para cada palavra, variando apenas em intensidade. Porém, esse modelo apresenta também resultados muito simplísticos. Apesar de útil para uma análise inicial,

perde a sua utilidade quando são buscadas informações mais precisas quanto às frases. Além disso, esse modelo já usado no âmbito das próprias redes sociais e também como visto nos exemplos do site Sentiment140 [6].

O *modelo de Plutchik* aplica a roda das emoções de Plutchik, mostrada abaixo. Esse modelo apresenta oito sentimentos básicos em formato circular e sentimentos e emoções mais complexas à medida que se afasta do centro da roda. Esse modelo, apesar de apresentar uma complexidade maior para ser trabalhado, se corretamente realizado pode informar com precisão o tom geral da mensagem. Para isso, pode ser usada uma classificação vetorial de cada palavra e esses vetores seriam somados, retornando no sentido da frase.

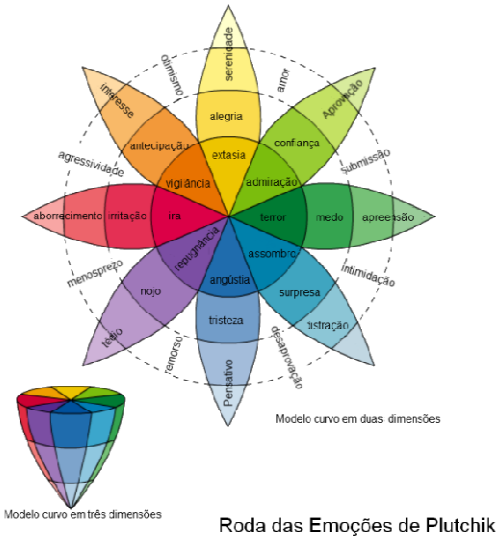


Figura 5: Roda das Emoções de Plutchik (PLUTCHIK, 2001)

III. TECNOLOGIAS ADAPTATIVAS

"O termo "tecnologia" denota o emprego de métodos e conhecimentos científicos com finalidades práticas. Assim, pode-se dizer que se entende por "Tecnologia Adaptativa" o conjunto das aplicações práticas do conceito fundamental da Adaptatividade, sempre que esta for utilizada como instrumento na resolução de problemas oriundos das mais variadas áreas de interesse" [9].

Destacam-se alguns Dispositivos Adaptativos: Dispositivos de reconhecimento, da classe dos autômatos, baseado na sucessão de mudanças de estado; Dispositivos para a representação de sistemas assíncronos, tais como statecharts, que incorporam mecanismos responsáveis pela representação dos fenômenos de sincronização; Dispositivos estocásticos, como as redes de Markov, capazes de representar fenômenos de caráter aleatório; Dispositivos de auxílio à tomada de decisões, representados principalmente pelas tabelas de decisão e pelas árvores de decisão.

Os dispositivos adaptativos são formados de duas camadas. Contêm um dispositivo convencional não adaptativo e uma camada externa envolvente, que leva consigo o mecanismo de adaptatividade. A camada externa irá atuar na camada interna modificando a estrutura, o comportamento e principalmente o conjunto de regras. A maior vantagem deste dispositivo é a facilidade em poder utilizar dispositivos não adaptativos com funções adaptativas, ocorrendo então à inclusão de camada adaptativa.

"Non-Adaptive" ou dispositivo orientado a regras é qualquer máquina formal, cujo comportamento depende de um conjunto de regras já conhecidos, essas regras devem ser finitas e sequenciais de modo que se possam mapear as próximas ações. Esses dispositivos podem ser determinísticos ou não determinísticos. Um dispositivo é chamado de adaptativo quando ocorre uma ação adaptativa que altere seu conjunto de regras, e somente se a ação adaptativa for não nula. Os dispositivos adaptativos devem ser capazes de reconhecer todos os mecanismo ao longo de toda sua extensão, e saber detectar quais são as possíveis situações em pode ocorrer efetivamente à mudança e de reagir adequadamente. No modelo proposto por JOSÉ NETO [10], ele enfatiza que o uso de regras em sistemas "Non-Adaptive" já evoluiu muito no que se diz respeito, a notação, sintaxe e semântica. É preciso utilizar-se dos sistemas não adaptativos já projetados e bem documentados, se familiarizar com os dispositivos adaptativos e começar a produzir novas notações se preocupando apenas com a parte que realmente exigirá do projetista a adaptabilidade.

Diversidade do conceito de adaptatividade

Para demonstrar que o conceito de adaptatividade pode ser empregado em diversos dispositivos computacionais, utilizaremos como exemplo a dissertação de mestrado de BASSETO [11], onde demonstra a inserção de sistemas computacionais na composição musical automatizada, modelando o conhecimento musical através de gramáticas sensíveis ao contexto. BASSETO [11] propõem em sua dissertação um modelo de formalismo adaptativo, mediante uma variação dos modelos não determinísticos em Rede de Markov.

A música é um fenômeno contínuo que se constitui de uma sequencia temporal de eventos discretizada no tempo. Os seus elementos que a compõem, o conjunto de símbolos nos remete a noção de linguagem, onde pode ser empregado o conceito de tradução musical, para uma linguagem assim chamada de partitura. Dado o conceito de que musica pode ser representada por uma linguagem, pode-se então manipular essa linguagem utilizando sistemas computacionais.

O conceito de composição musical se difere completamente do conceito de caos, e sim a uma semântica musical. Comumente um computador não pode ter a sensibilidade de criação, de compor músicas. Porém guiados por programas pré-definidos, poderiam auxiliar na criação de novas combinações, novas composições musicais. Se os computadores forem sujeitados a criação a partir de escolhas aleatórias, seria praticamente submetê-los a geração de ruídos, não haveria um som harmonioso. Porém se programados a partir de regras, de uma linguagem com estruturas sintáticas. "A possibilidade de codificação desses símbolos gráficos em

elementos computacionais pode ser inferida pela existência de um conjunto finito de símbolos e de sua ordenação lógica. A existência de denominações para cada um dos símbolos da partitura já é um ponto de partida para este tipo de codificação" [11].

A música é tratada como uma linguagem musical, e a forma da escrita são transpostas para uma linguagem computacional, ou seja, conversão de regras, onde ele chama de Representação em Listas de Eventos; mais adiante BASSETO [11] trata a questão da interpretação da partitura musical como uma Representação Procedimental, onde um algoritmo, passo a passo, irá interpretar os símbolos, assim como um músico quando lê a partitura. Essas duas representações são semelhantes à forma de representação musical tradicional, ou seja, um processo computacional simplório de execução imediata do material musical. A evolução do trabalho de BASSETO [11] propõem mais duas representações que incluem elementos semânticos sofisticados, onde a proposta seja uma aplicação amigável, mais inteligente e compostas por mais recursos computacionais, onde seja possível através a concepção do usuário realizarmos a representação de modelos musicais semânticos. A Representação Estruturada que engloba o conceito de contexto no aninhamento de estruturas e também o conceito de abstração dos dados associados a elementos musicais; A Representação Semântica, que é a mais interessante para esse projeto de pesquisa, capaz de adaptar modelos mentais e efetuar processamentos muito mais específicos sobre o material musical. Nesta representação os eventos musicais não são tratados de maneira isolada. "A implementação da semântica de um dado objeto musical consiste normalmente na definição de propriedades ou atributos, específicos para este dado objeto musical, bem como através do estabelecimento de relações determinadas entre os objetos musicais que foram definidos" [11].

Segundo PISTORI [12], as árvores de decisão são uma forma de representar hierarquicamente funções discretas sobre múltiplas variáveis. Combinam as características discretas de um dispositivo adaptativo, cujo mecanismo subjacente é uma arvore de decisão, com estratégias para tratamento de valores contínuos, incorretos e inconsistentes.

"Uma arvore de decisão não-determinística é uma arvore de decisão que incorpora o conceito de não-determinismo, bastante estudado na teoria dos autômatos. A introdução do não-determinismo possibilita um tratamento elegante de problemas relacionados com informação incompleta ou ausente, além de propiciar uma distinção conceitual entre decisões exatas e decisões aproximadas" [12].

IV. CLASSIFICANDO SENTIMENTOS COM O APOIO DE ÁRVORES ADAPTATIVAS

Para o emprego da classificação de sentimentos de textos extraídos de uma rede social, neste caso o Twitter, é preciso que exista uma base de conhecimento que primeiramente reconheça se o texto, no caso do Twitter, se o tweet (menção) pertence à ontologia a ser estudada. Sabe-se que o Twitter é uma rede social pública e disponível para qualquer pessoa que queira minerar dados de sua base. O único padrão que o Twitter obrigatoriamente limita aos seus usuários, é a publicação de tweets com no máximo 140 caracteres. Além dessa limitação, não existe nenhuma outra exigência, de escrita, na forma da escrita, do uso de caractere especial, espaçamentos, fotos e links. Ou seja, os textos podem ser lexicalmente e sintaticamente corretos, como podem ser textos incorretos. A partir dessa observação tornam-se como premissa do projeto proposto, que não serão resolvidos problemas léxicos e sintáticos da língua a ser estudada, mas que o principal problema é o reconhecimento de uma palavra que semanticamente representa um sentimento, uma emoção. Mais adiante será mostrado como o coletor de "sentimentos" tratará cada palavra que representa um sentimento como um único símbolo dentro do autômato adaptativo.

A primeira decisão prática a ser tomada é: que língua será utilizada e parametrizada para ser utilizada como filtro inicial da base de dados que se deseja extrair? Para o modelo proposto optou-se pela Língua Portuguesa, por se tratar de uma língua que muito tem a contribuir em PLN – Processamento de Linguagem Natural. A segunda decisão a ser tomada refere-se a qual banco de dados deve-se utilizar? Facebook? Um jornal regional? Transcrever áudios de sessões entre um paciente e um terapeuta? Para o modelo proposto optou-se pela rede social Twitter, utilizada mundialmente, que desperta grande interesse coletivo sobre assuntos temporais. A terceira decisão é: quais mensagens enquadram-se na Ontologia desejada? Para o modelo proposto a Ontologia pode variar a qualquer momento, pois são sentimentos ligados a Ontologia escolhida que será a saída, indicadores e classificadores que de certa forma medirão o cunho de emoções de um assunto dentro da Ontologia analisada. Pode-se optar também por escolher mais de uma ou várias Ontologias para treinamento inicial e filtragem dos sentimentos que estão na figura 9, representados pelo retângulo "Filtragem *Top Down*". Nesta etapa que serão reconhecidos os sentimentos, ou seja, quais palavras contidas nos textos extraídos do Twitter devem ser mineradas? Na ausência de um corpus notado em língua portuguesa, que diferencie uma palavra e que a reconheça e a categorize como um sentimento (não importando neste momento qual seja ele) é preciso ensinar a máquina a aprender esse reconhecimento. Nesta fase optou-se por um modelo de automoto adaptativo que reconheça o léxico e o classifique como um sentimento.

V. RESULTADOS

Modelo Proposto 1 – MFA – Modelo de Filtragem Adaptativa

Na figura 6 é proposto um modelo de filtragem adaptativo que irá auxiliar no reprocessamento e enriquecimento da base de dados dos sentimentos que serão utilizados como entrada no MCS – Modelo de Classificação de Sentimentos da figura 9.

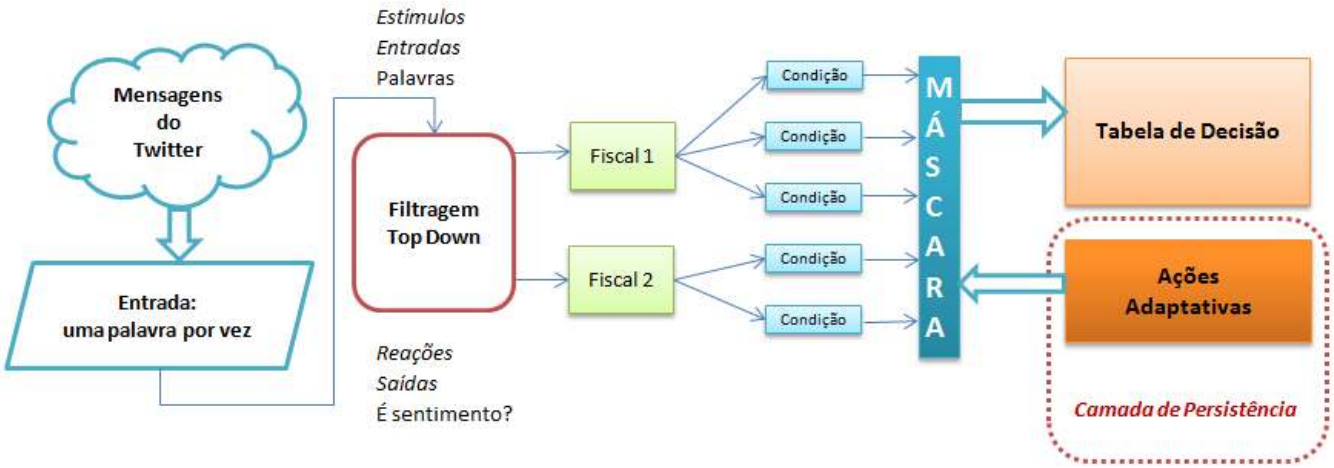


Figura 6: MFA - Modelo de Filtragem Adaptativa

Segundo CEREDA [13], Persistência é o nome dado para o intervalo de disponibilidade do conteúdo de um determinado objeto. Em soluções computacionais essa característica é fundamental em diversos problemas. Ao longo do tempo as informações coletadas necessitam de uma área de preservação do estado, de atualização ou até mesmo uma eventual remoção. "A persistência é uma funcionalidade que pode contribuir para a extensão dos dispositivos adaptativos, conferindo-lhes a capacidade de preservar estados internos e recupera-los conforme a necessidade [13]".

O MFA possui os seguintes componentes:

- Nuvem extratora de informações do Twitter;
- Módulo de entrada: onde cada palavra é processada individualmente umas das outras;
- Módulo de Filtragem Top Down: onde será enriquecido a cada novo processamento para que se tenha um conhecimento pré-existente (isso é imprescindível para a alimentação do MCS), este módulo receberá os estímulos/entradas que são as palavras uma a uma de cada texto e enviará as saídas que são chamadas de reações, ao próximo módulo.
- Módulos Fiscais: que tem a função de monitorar/vigiar o módulo anterior e diagnosticá-lo;
- Condições: onde as saídas serão analisadas e validadas dando subsidio na tomada de decisão;
- Máscara: que tem a função de interfacear às condições no devido preenchimento da tabela de decisão. A máscara também terá o papel de incorporar as mensagens de retorno das ações adaptativas e incrementar a tabela de decisão.
- Tabela de Decisão: avalia os diagnósticos do bloco anterior, e atua sobre todos os modelos do sistema para adequá-los.
- Ações Adaptativas: que possui a capacidade de aprender e incorporar novos conhecimentos, associando a cada conhecimento adquirido, a informação da procedência da nova informação, e retornando essa informação para a Máscara. A manipulação das informações dentro do módulo de Ações Adaptativas se dá devida a capacidade dos dispositivos adaptativos poderem interpretar a informação e manipulá-la. Este módulo é responsável por todo tipo de decisão tomada, e também se responsabilidade pela mudança no entorno do modelo a cada vez que uma nova informação é acrescentada.

No MFA o módulo de Ações Adaptativas é composto de uma camada de persistência para facilitar na manutenção e atualização da base de conhecimento que deve ser produzida. A persistência ajuda na configuração de ações léxico-sintáticas em dispositivos adaptativos. A estrutura propostas em camadas, onde a persistência esta implementada em uma camada léxico-sintática, permite, que se caso for preciso realizar alguma alteração nas regras não seria preciso excluir, e treinar a minha base de informações já armazenadas, bastaria apenas excluir e consertar o item que apresenta a inconsistência. Isso mostra que não preciso modificar a estrutura completa de um dispositivo adaptativo e sim envolvê-lo com essa camada de persistência (regras e conhecimento).

Abaixo temos dois exemplos, o primeiro constando a seguinte menção "eu to bem triste com esse negócio da redução da maioridade penal ter sido aprovada q odio" e o segundo com a seguinte menção "Feliz por terem aprovado a maioridade penal ☺". Ambos os exemplos foram extraídos do Twitter no dia 02 de Jul. de 2015, e ambos referem-se ao tema de importância nacional "redução da maioridade penal".

O MFA processa cada palavra por vez, e interpreta os símbolos e a composição deles. No primeiro processamento ele aprende, incrementa, e atualiza a base de conhecimento. Nos exemplos acima a base seria incrementada com os seguintes sentimentos: triste / ódio / feliz. Nos processamentos seguintes a comunicação entre a máscara e as condições também serão atualizados com as novas regras, e caso entre uma palavra como, por exemplo, "tristeza" os dispositivos adaptativos podem analisar a terminologia da palavra e já classifica-la também como um sentimento. É importante destacar que no MFA não esta sendo analisada a proximidade semântica dos sentimentos. O principal objetivo do MFA é dizer se a palavra é ou não é um sentimento e enriquecer a base a casa vez que chega um novo sentimento na escrita textual em língua portuguesa.

O MFA é de extrema importância, pois nesse módulo já

será realizado todo o descarte de informações não relevantes, permanecendo apenas as palavras categorizadas como

Figura 7 - Exemplo de mensagem "feliz" (extraído do TweetDeck em 02 de Jul. de 2015)

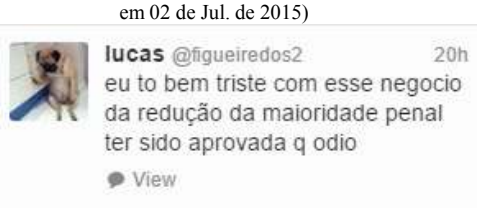


Figura 8 - Exemplo de mensagem "ódio/triste"(extraído do TweetDeck em 02 de Jul. de 2015)

sentimentos, formando então uma base enriquecida, que por sua vez servirá de entrada para o MCS na sessão seguinte. Recomenda-se que primeiro seja treinada uma quantidade razoável de dados oriundos de diversas Ontologias separadamente do funcionamento do MCS. E somente depois de treinado o MFA deve acopla-lo ao MCS. Isso ajuda na performance e evita o processamento desnecessário de informações. Pois o tempo que se gasta esperando o MFA processar cada palavra a mencionar se é ou não um sentimento, em fase de aprendizagem, e mandar essa informação para o MCS, é desnecessário. Então primeiramente deve ocorrer o treinamento do MFA e depois acopla-lo no MCS. E quando entrar um sentimento novo é permissível o tempo de espera do MCS pelo MFA trabalhando em conjunto. Lembrando que as ações adaptativas do MFA realizam modificações somente em seu próprio modelo. Em seguida será explorado o objetivo do MCS e como ocorrem suas ações adaptativas.

Modelo Proposto 2 – MCS – Modelo de Classificação de Sentimentos

Abaixo é proposto o Modelo de Classificação de Sentimentos utilizando o conceito de árvores adaptativas, para realizar a classificação por proximidade semântica do sentimento mais próximo ao sentimento primário encontrado na Roda das Emoções de Plutchik [4].

Na sessão anterior foi proposto o MFA – Modelo de Filtragem Adaptativa que está ilustrado no MCS – Modelo de Classificação de Sentimentos, pelo quarto componente no fluxo acima proposto, o retângulo "Filtragem Top Down" com uma linha pontilhada em vermelho, onde indica que este é um módulo que possui recurso de adaptatividade e que executa sua função independente dos outros componentes do MCS.

As saídas que foram geradas no MFA serão as entradas e estímulos do MCS. Neste passo os sentimentos são apenas palavras, cada um com seu significado, porém o modelo ainda não é capaz de diferenciá-las em sua forma semântica.

Na figura 10 estão ilustradas as oito emoções primárias que compõem a Roda das Emoções de Plutchik. A partir das emoções primárias foram ilustradas as demais emoções utilizando a notação gráfica de árvores de decisão.

Abaixo esta ilustrada um exemplo da árvore de decisão que pode compor a emoção primária "Ira". A classificação foi feita a partir de sinônimos de palavras e por proximidade semântica. Na Roda das Emoções "Ira" esta representada na cor rosa, e vem acompanhada das emoções: "irritação" e "aborrecimento". Essas duas emoções estão representadas na árvore de decisão cada uma por um nó, mas isso não necessariamente significa que elas são opostas ou contrárias, pelo contrário, são emoções sintaticamente escritas de forma diferentes, porém com proximidade semântica. As que são aninhadas de forma hierárquica pai e filho, são as emoções semanticamente mais próximas, exemplo as emoções fúria (pai) e furor (filho); aborrecimento (pai) e abatimento (filho); irritação (pai) e raiva (filho). A relação de proximidade semântica neste exercício se dá pelo significado de cada palavra, e não pela geração de uma emoção primária, que por sua vez gera suas emoções secundárias e assim por adiante. E emoções como impaciência (filho de ansiedade) e desânimo (filho de abatimento) não possuem grande proximidade semântica, mas são originadas da mesma emoção primária. Devida a essa sinergia entre emoções semanticamente

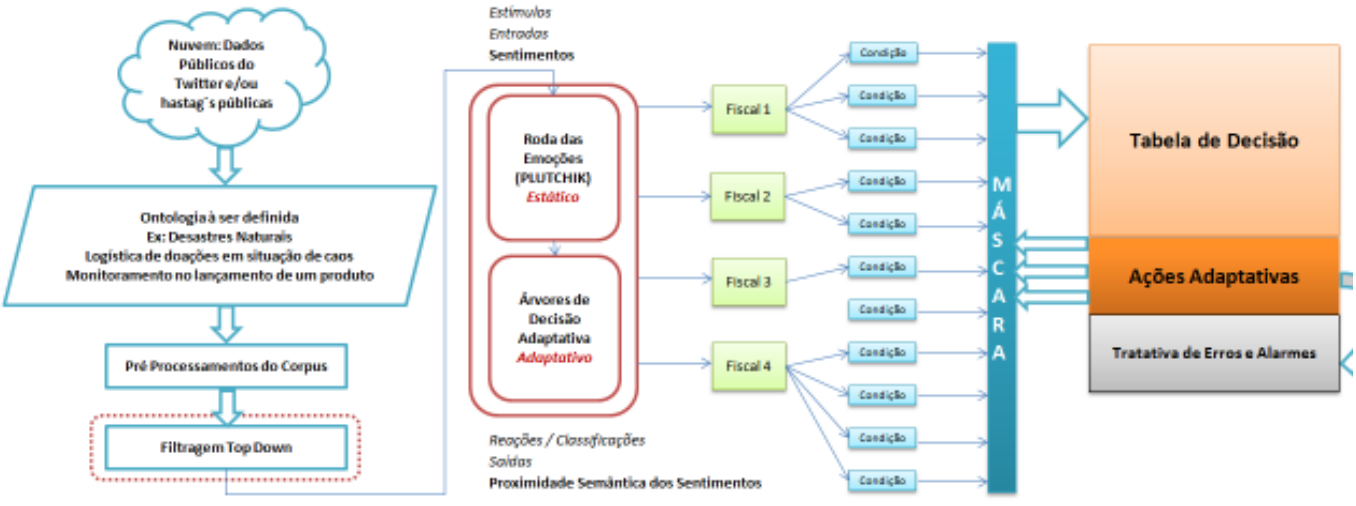


Figura 9 - MCS - Modelo de Classificação de Sentimentos

próximas e não próximas justifica-se a utilização do mecanismo de árvore de decisão.

Figura 10 - Emoções Primárias

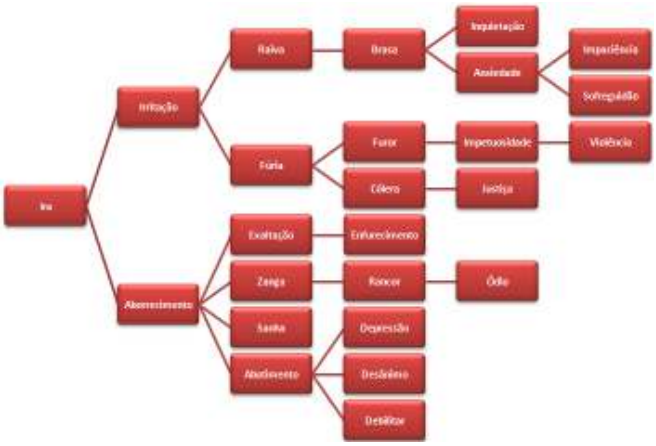


Figura 11 - Árvore de Decisão da Emoção "Ira"

A árvore de decisão apresentada com a emoção primaria "ira" é um exercício prático de como seria o comportamento do componete modelo representado pelo retangulo em vermelho na figura 9. Esse componete é formado pela união do quadrado "Roda das Emoções" mais o quadrado "Árvore de Decisão Adaptativa", sendo o primeiro uma pré decisão da entrada do quadrado seguinte, ambos trabalham em conjunto, o primeiro fazendo a seleção de uma emoção e verificando se a mesma já existe na Roda das Emoções, caso contrário, a emoção é enviada para o segundo onde será realizada uma classificação a partir da árvore de decisão adaptativa. Por exemplo, se a árvore de decisão recebe-se a palavra "Petulância", como ilustrada na figura 12, a árvore realizaria os testes iniciando da emoção mais primária "admiração", em seguida faria um teste de sinomino e varredura histórica, para verificar em qual dos filhos ela melhor se encaixa, passando pelas emoções "confiança", em seguida "fidúcia", em seguida "atrevimento", e finalmente seria aninhada a emoção "ousadia". Esta classificação resulta em um desdobramento de 5 nós para realizar essa classificação.

Figura 12 - Árvore de Decisão da Palavra "Admiração"

A figura 12, também representa um exercício de como seria o comportamento da Árvore de Decisão Adaptativa caso recebe-se como entrada as emoções nela contida. Caso esse componente receba como entrada uma emoção ainda não mapeada, ou seja uma emoção nova, automaticamente seriam acionados os outros módulos do sistema. Primeiramente os "fiscais" com o objetivo de vigiar e diagnosticar o módulo anterior caso manifeste algum comportamento diferente, neste caso, o módulo anterior ainda não saberia identificar a nova emoção. Em seguida, passaria pelas componentes de "condição", que analisam, filtram e dão os subsídios necessários para a tomada de decisão. Na tabela de decisão então é avaliado os diagnósticos do bloco anterior, e atuará sobre todos os modelos do sistema para adequá-los. Caso necessário então ações adaptativas operam transformando as regras e modificando o modelo atual, principalmente na criação de uma nova categoria para a nova emoção inputada no modelo de classificação de sentimentos. O componente de ações adaptativas retroalimenta o modelo. É importante destacar também os componentes de erro e alarme que registram o comportamento do modelo, caso não exista nenhum caminho a seguir ou nenhuma decisão a ser tomada.

Problemas em Aberto

No exercício praticado para compor a árvore de decisão adaptativa dotada de emoções, secundárias, terciárias, quartanárias e assim por diante, foi utilizado um raciocínio a partir dos sinônimos das palavras que já vinham mapeadas como sentimentos no MFA – Modelo de Filtragem Adaptativa. Um dos problemas que fica em aberto neste trabalho para desenvolvimento futuro é a relação automática para aninhar os sentimentos no nó mais próximo semanticamente de sua emoção primária. Poder-se-ia explorar mais a fundo as técnicas de frequência, probabilidade e estática, e principalmente de semântica latente.

Como um ponto de atenção para estudos futuros é a exploração de qual emoção gera outra emoção no âmbito psicológico de formação dessa emoção, propriamente dita, dentro do ser humano. E se pode haver uma relação dessa

formação natural de emoções e passagens por cada uma dessas emoções com a forma de classificação adaptativa utilizando árvore de decisão para classificar as emoções a partir de um modelo raiz, que é a Roda das Emoções.

VI. CONCLUSÃO

Nos modelos teóricos propostos neste trabalho de pesquisa o MFA e o MCS, foi possível esboçar o uso de funções Adaptativas que podem auxiliam no tempo de processamento, na tomada de decisão e principalmente no tamanho do caminho que o sistema irá percorrer para chegar a uma solução coerente, e certamente correta. O uso da camada de persistência pode se mostrar eficaz no MFA, comunicando-se diretamente com a Máscara que por sua vez, retroalimenta o sistema e as gravações nas tabelas de decisões. Tendo os sentimentos já separados pelo MFA, e trabalhando em conjunto com o MCS pode-se perceber a eficiência dos modelos em funcionamento paralelo. O uso de recursos adaptativos no MCS pode se mostrar eficaz e coerente para classificação de sentimentos fundamentada nas oito emoções primárias da Roda das Emoções.

O mapeamento deste trabalho de pesquisa foi fundamental para realizar o detalhamento das atividades de próximos passos. Classificação de sentimentos é uma área multidisciplinar que envolve o conhecimento em ferramentas de processamento de linguagem natural, e o uso de ferramentas adaptativas foi fundamental no processo de aprendizagem do sistema de classificação de sentimentos.

AGRADECIMENTOS

Ao CNPQ, aos colegas de trabalho do LTA – Laboratório e Linguagens e Técnicas Adaptativas, aos organizadores deste Workshop e ao programa de Pós-Graduação em Engenharia da Computação.

REFERÊNCIAS

[1] GRIGORI, S. et al. **Empirical study of machine learning based approach for opinion mining in tweets**. In *Proceedings of the 11th Mexican international conference on Advances in Artificial Intelligence,* 2012. Springer-Verlag, Berlin, Heidelberg, 1-14

[2] KANAVOS, A. et al. **Conversation Emotional Modeling in Social Networks**, Tools with Artificial Intelligence (ICTAI), 2014 IEEE 26th International Conference on , vol., no., pp.478,484, 10-12 Nov. 2014

[3] PORSHNEV, A.; REDKIN, I.; SHEVCHENKO, A., "**Machine Learning in Prediction of Stock Market Indicators Based on Historical Data and Data from Twitter Sentiment Analysis**," Data Mining Workshops (ICDMW), 2013 IEEE 13th International Conference on , vol., no., pp.440,444, 7-10 Dec. 2013.

[4] PLUTCHIK, R; **The Nature of Emotions**. In: American Scientist Magazine, vol. 89, number 4, pp. 344-350, 2001.

[5] SOCIAL TIMES. **The 10 Biggest Social Networks Worldwide**. Disponível em: <http://www.ad week.com/socialtimes/largest-social-networks-worldwide/504044>. Acesso em 19 abr. 2015.

[6] SENTIMENT 140. Disponível em: <http://www.sentiment140.com >. Acesso em 19 abr. 2015.

[7] KWAK, H., LEE, C., PARK, H., and MOON, S. (2010). "**What is Twitter, a social network or a news media?**", Proceedings of the 19th International Conference on World Wide Web, (New York, NY, USA: ACM), pp. 591–600.

[8] SLOMAN, A., CHRISLEY, R., SCHEUTZ, M. (2005). "**The architectural basis of affective states and processes**", Who Needs Emotions?: The Brain Meets the Machine, v. 3, pp. 203–244.

[9] JOSÉ NETO, J. **Um glossário sobre adaptatividade. In: Terceiro Workshop de Tecnologia Adaptativa** - WTA 2009. EPUSP, 2009.

[10] JOSÉ NETO, J. **Adaptive Rule-Driven Devices - General Formulation and Case Study**., in Bruce W. Watson & Derick Wood, ed., 'CIAA', Springer, pp. 234-250, 2001.

[11] BASSETO, B. A. Um **Sistema de Composição Musical Automatizada, Baseado em Gramáticas Sensíveis ao Contexto, Implementado com Formalismos Adaptativos**. São Paulo 2000, 143p. Dissertação de Mestrado. Escola Politécnica, Universidade de São Paulo.

[12] PISTORI, H. **Tecnologia em Engenharia da Computação: Estado da Arte e Aplicações**. Tese de Doutorado, Escola Politécnica da Universidade de São Paulo, 2003.

[13] CEREDA, Paulo Roberto Massa, JOSÉ NETO, João. **Persistência em dispositivos adaptativos**. Em: Memórias do VIII Workshop de Tecnologia Adaptativa – WTA 2014. EPUSP, São Paulo, ISBN: 978-85-86686-76-4, pp. 120-125. 06 e 07 de Fevereiro de 2014.

Ariana Moura da Silva, graduada em Sistemas de Informação pela FAENAC – Faculdade Editora Nacional e Mestre em Engenharia da Informação pela UFABC – Universidade Federal do ABC. Atualmente, é doutoranda 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 aluna pesquisadora no LTA – Laboratório de Linguagens e Técnicas Adaptativas – do PCS. Tem experiência na área de Processamento de Linguagem Natural, com ênfase em Análise Semântica, atuando principalmente nos seguintes temas: classificação de emoções, mineração de dados de redes sociais, redes de comunicação em desastres naturais.
Curriculum Lattes: http://lattes.cnpq.br/2930741552082859.

Ricardo Luis de Azevedo da Rocha, 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.
Curriculum Lattes: http://lattes.cnpq.br/5660360751410581.

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 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.
Curriculum Lattes: http://lattes.cnpq.br/4091709928353457.

An Adaptive Middleware Solution for Complex-Event Processing

A. L. Gonzalez, R. L. A. Rocha and J. J. Neto

***Abstract*— The occurrence of events is growing because of the increasing rate of use of services made available via internet, such as data storage. On that basis, however, there is a growing number of threats to the information, requiring approaches capable of promptly responding to unknown threats. The objective of this research is to create an approach that is able to adapt itself from an initial configuration based on historical data threats, in order to obtain the means to respond to new and unknown threats. For this, an approach involving complex-event processing (CEP) and ontologies, to coordinate the CEP, is proposed. Thus, it is expected that it is possible to treat a wide variety of threats, known or unknown, in a effective and efficient manner. Thus, one can ensure greater security to the information storage services located in clouds.**

***Keywords*— Complex-Event Processing (CEP); Adaptivity; Middleware; Ontology.**

I. INTRODUÇÃO

O TRABALHO apresentado por este artigo refere-se a um trabalho de pesquisa em andamento. Este trabalho está sendo desenvolvido no contexto do Programa de Pós-Graduação em Engenharia de Computação da POLI/USP. Por se tratar de um trabalho em andamento, alguns pontos ainda estão indefinidos e é necessária maior pesquisa para fazer as escolhas da forma mais embasada possível. Assim, este artigo apresenta o projeto de pesquisa, os pontos principais e alguns tópicos ainda a serem definidos.

A pesquisa envolve temas relacionados à segurança cibernética, processamento de eventos, técnicas adaptativas e representação de conhecimento. A área de segurança cibernética foi escolhida com o intuito de delimitar o escopo da abordagem a ser proposta, mas a aplicação da mesma poderá ser expandida, caso seja notada esta necessidade.

A utilização de técnicas adaptativas foi imaginada no decorrer da disciplina PCS5004 – Fundamentos e Aplicações da Tecnologia Adaptativa, da POLI/USP.

O crescente volume de dados gerados no monitoramento e supervisão dos recursos disponibilizados pelos sistemas de informação, armazenados em bases de dados de séries temporais dificultam a adoção bem fundamentada de ações de controle e supervisão de um operador humano no contexto de segurança cibernética.

A utilização de ontologias e técnicas adaptativas para representação computacional de conhecimento e sua utilização em uma solução automática e de tempo-real para classificação de eventos primários e lançamento de eventos complexos para aumentar a consciência situacional do estado de operação do sistema e de seu contexto de operação, torna mais fácil a adoção de ações de controle e supervisão de um operador humano no contexto de segurança cibernética.

A Figura 1 apresenta o resumo gráfico da abordagem proposta.

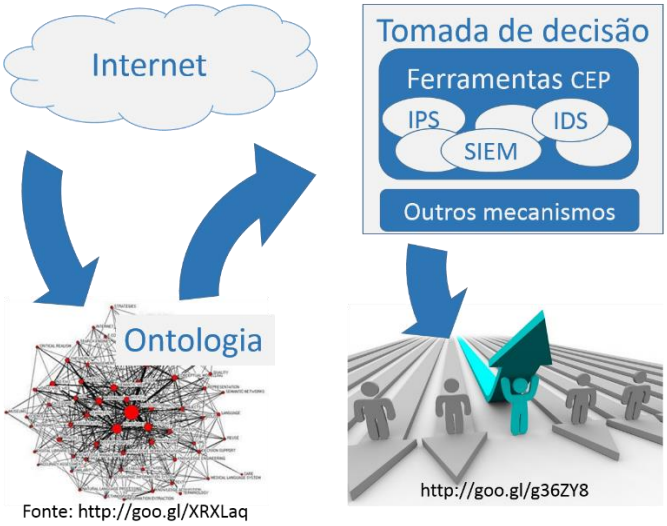


Figura 1- Resumo da abordagem proposta

A. *Problema de Pesquisa*

O problema de pesquisa a ser abordado neste trabalho visa atender ao tema de pesquisa do trabalho de pós-graduação do autor. Este tema foi levantado nas pesquisas iniciais, realizadas previamente ao início dos trabalhos, e aprimorado no contexto da disciplina PCS5012 – Metodologia de Pesquisa Científica para Engenharia de Computação, da POLI/USP. Este trabalho de pesquisa visa encontrar uma resposta para o problema de pesquisa que considera a dificuldade em adotar ações para controle e supervisão de eventos, levando-se em conta o contexto de segurança cibernética.

B. *Hipótese*

Para responder ao problema de pesquisa levantado, imagina-se que a utilização de ontologias e técnicas adaptativas para representação computacional de conhecimento proporciona um aumento da consciência situacional do estado de operação do sistema e de seu contexto de operação e, além disso, a utilização desta forma de representação de conhecimento, em uma solução automática e de tempo-real para classificação de eventos primários e lançamento de eventos complexos, torna mais fácil a adoção de ações de controle e supervisão de um operador humano no contexto de segurança cibernética.

C. *Justificativa*

As hipóteses, são importantes pois direcionam o trabalho em desenvolvimento. Esta hipótese será o rumo que o trabalho deverá seguir e sua aceitação, ou rejeição, deverá ser discutido e apresentado como resultado do trabalho. A rejeição da hipótese não significa que o trabalho não tem valor, pois pode-se gerar conhecimento novo no processo de investigação para a rejeição da mesma.

D. *Objetivos*

Com o intuito de desenvolver este trabalho e verificar seus resultados, foi delineado um objetivo geral, que deverá ser validado ao término do trabalho. A partir deste objetivo geral, foram traçados objetivos específicos, que serão verificados durante o trabalho, de forma similar a checkpoints do processo de desenvolvimento do trabalho.

Este trabalho propõe a construção adaptativa de ontologias para representação computacional de conhecimento extraído de análises realizadas no crescente volume de dados gerados no monitoramento e supervisão dos recursos disponibilizados pelos sistemas de informação, armazenados em bases de dados de séries temporais e sua utilização em uma solução automática e de tempo-real para classificação de eventos primários e lançamento de eventos complexos para aumentar a consciência situacional do estado de operação do sistema e de seu contexto de operação, suportando a adoção bem fundamentada de ações de controle e supervisão de um operador humano no contexto de segurança cibernética.

A partir do objetivo geral apresentado acima, podem-se traçar alguns objetivos específicos, que serão abordados no decorrer do desenvolvimento do trabalho. Os objetivos específicos são:

- Estudo e escolha de ferramentas CEP;
- Integração da solução às ferramentas CEP escolhidas;
- Escrita de artigos para compartilhamento do conhecimento.

E. *Resultados Esperados*

A partir do objetivo do trabalho, podem-se definir alguns resultados esperados deste trabalho de pesquisa. Os resultados esperados são:

- Definição de uma abordagem para CEP na área de cybersecurity que se utilize dos benefícios das ontologias para aumentar a segurança da informação;
- A verificação da aplicabilidade da abordagem proposta também faz parte dos resultados esperados.

F. *Organização do Trabalho*

Este trabalho está organizado de forma que a Seção II apresenta uma revisão dos principais trabalhos relacionados, fornecendo o embasamento teórico necessário para a discussão da escolha relacionada ao dispositivo adaptativo. A Seção III descreve a solução adaptativa para processamento de eventos complexos proposta neste trabalho. A Seção IV apresenta as conclusões do trabalho. Por fim, são apresentadas as referências consultadas para a concepção deste trabalho.

II. CONCEITOS GERAIS

A área de tecnologia da informação (TI) está passando por um período de grande crescimento do fluxo de informação. Fortes pressões na forma de utilização de sistemas de informação, com usuários utilizando seus próprios dispositivos (BYOD, do inglês, *Bring-Your-Own-Device*) para acessar sistemas antes disponíveis apenas dentro do ambiente corporativo e a alta capilaridade na aquisição de dados de fontes diversas (IoT, do inglês, *Internet-of-Things*) tornam possível a disponibilização em tempo-real de informação contextualizada e distribuída para um usuário tomar decisões bem fundamentadas.

Além dos desafios impostos a demandas de conectividade e capacidade de armazenamento de dados, os sistemas de informação passam a ser mais valiosos, do ponto de vista de disponibilidade de informação mais detalhada de cada usuário, como hábitos e contexto de utilização de informação e mais expostos, do ponto de vista de segurança, uma vez que se tornam acessíveis de praticamente qualquer lugar através de qualquer dispositivo.

A expansão da conectividade, o armazenamento e coleta de um volume cada vez maior de dados, o aumento da capacidade de análise e a obtenção da informação a partir dos dados tornam a proteção destes sistemas uma tarefa desafiadora, de forma que abordagens tradicionais de segurança de rede continuam necessárias, mas estão longe de serem suficientes num contexto em que as consequências do roubo de informação se tornam cada vez mais abrangentes.

Soluções de segurança do tipo caixa-forte, com forte separação entre ambiente interno e externo, não são mais aplicáveis em muitos cenários, exigindo uma consciência situacional maior para identificação e monitoramento de atividades de cada usuário.

Nesta abordagem, além da capacidade de trabalhar sobre grande volume de dados em tempo-real, há a dificuldade de se lidar com a imprevisibilidade e não sincronismo das ações e intenções de um usuário. A consciência da necessidade de acompanhar as ações de usuários específicos dentro de um sistema levou à construção de soluções SIEM, que vão além do estabelecimento de uma barreira de acesso e buscam a identificação e rastreamento de ações que ocorrem dentro do sistema em tempo-real baseando-se no processamento de eventos de segurança provenientes de dispositivos de controle de acesso, IDS, IPS e sobre eventos gerados por sistemas como o SPLUNK, que monitoram logs textuais de diversos sistemas e aplicam regras pré-definidas.

Existem, hoje, inúmeros sistemas capturando, recebendo e gerando informação de e para o mundo real. Uma dificuldade crescente está em dar significado a toda esta informação existente, de forma que a qualidade da informação para uso em diversos outros fins seja aumentada e que possam ser obtidas respostas mais eficientes a partir delas [6]. No contexto de segurança da informação, tentativas de intrusão não autorizada a redes e sistemas de armazenamento deixam rastros de informações que podem ser transformados em eventos. Estes podem ser coletados, classificados, processados

e utilizados para decisão de quais medidas apropriadas serão adotadas para evitar ou minimizar as consequências de eventos indesejáveis. [4] define evento como algo notável que acontece. Desta forma, pode-se considerar este fluxo de informação como um conjunto de eventos observados em um intervalo de tempo, sob uma determinada óptica. Sistemas orientados a eventos permitem que eventos sejam, automaticamente, interpretados e correlacionados.

A. *Adaptatividade*

A adaptabilidade oferece ao usuário de um sistema a possibilidade de alterar as configurações, a partir de uma quantidade limitada de opções previstas no processo de desenvolvimento e pré-programadas. A adaptatividade, por outro lado, permite que um sistema se autoconfigure, em tempo de execução, sem a interferência de um agente externo.

Pela utilização de técnicas adaptativas, o sistema é capaz de se adaptar a situações novas ao se confrontar com situações imprevistas. Estas alterações ocorrem de maneira automática pelo sistema, e de forma transparente ao usuário [13] [14]. Pela utilização da adaptatividade, pode ser impossível reconhecer um software após algum tempo em execução.

B. *Dispositivos Adaptativos*

Dispositivos orientados a regras são quaisquer dispositivos que sejam dependentes de sequências finitas de regras pré-definidas.

A adaptatividade de um dispositivo se dá a partir da existência de funções que permitam que tal dispositivo altere as suas próprias regras de construção.

Em [15] é proposto um modelo híbrido, que permite a adoção de técnicas adaptativas sobre dispositivos orientados a regras já existentes. Neste trabalho, os dispositivos adaptativos são formados por duas camadas, sendo uma delas não adaptativa (dispositivo convencional) e a outra responsável por implementar as funções de adaptatividade. Desta forma, a camada adaptativa tem a capacidade de modificar a camada convencional, mudando seu conjunto de regras.

C. *Statecharts Adaptativos*

Nos artigos [13] e [14], os autores apresentam os conceitos e a formulação geral acerca das técnicas adaptativas.

No ano seguinte à publicação do segundo trabalho, foi publicado [1]. Este trabalho surge como uma continuação dos primeiros, no qual é apresentada uma proposta de utilização das técnicas adaptativas de maneira gráfica.

A abordagem desenvolvida faz uso de elementos conhecidos como statecharts. Statecharts são dispositivos hierárquicos que visam descrever o comportamento reativo de sistemas de maneira gráfica.

Os statecharts aparecem como uma evolução das máquinas de estados finitos e dos diagramas de estado. Este tipo de diagrama permite que a dinamicidade dos sistemas que estão descrevendo.

Statecharts são formados por retângulos com bordas arredondadas, representando os estados. Além disso, existem também linhas que conectam os retângulos. Estas linhas representam as transições entre os estados [1].

O "funcionamento" dos statecharts se dá de maneira semelhante ao das máquinas de estados finitos. As transições dependem do estado atual em que a máquina se encontra e de um evento (entrada). Na ocorrência de eventos previstos, a transição é disparada e a máquina passa para o próximo estado.

Porém, statecharts ampliam as máquinas de estados finitos por permitirem a existência de transições ortogonais, ou seja, simultâneas.

A utilização de statecharts para a modelagem do comportamento reativo permitiu um aprofundamento e consequente evolução das abordagens adaptativas utilizadas até este momento. Os statecharts adaptativos são construídos a partir de statecharts convencionais existentes. Seu funcionamento se dará da mesma maneira, a não ser quando da existência de funções adaptativas associadas às transições.

Estas funções adaptativas, quando encontradas em uma transição entre estados, permite que o statechart execute uma transformação sobre seus estados e transições, ou seja, uma auto-reconfiguração.

As funções adaptativas permitem que novos estados ou transições sejam incluídos no statechart, ou que elementos existentes sejam eliminados.

A partir de sua utilização e após algum tempo de execução, pode tornar-se impossível reconhecer o statechart adaptativo inicial.

Statecharts são capazes de representar o comportamento dinâmico dos sistemas que descrevem. Os estados podem ser encadeados e ligados, de forma a representar os sistemas em seus comportamentos sequencial ou concorrente [1].

As ligações entre os estados são chamadas de transições, e a cada transição podem ser associadas ações que serão executadas quando a transição ocorrer. Para que uma transição ocorra, é necessário que uma condição seja satisfeita.

As ações associadas às transições dos statecharts podem ser ações adaptativas. Estas ações, quando ocorrem, podem modificar o conjunto de estados e/ou as ligações entre os estados do statechart. Estados e transições podem ser criados e eliminados dos statecharts pelas ações adaptativas.

A Figura 2 apresenta um exemplo de um statechart, com uma ação adaptativa em destaque. A ação adaptativa está representada em vermelho, com a identificação $\mathcal{A}()$.

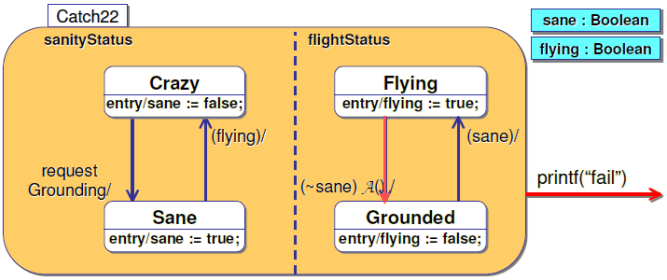


Figura 2 - Exemplo de Statechart Adaptativo (Fonte: Material de aula PCS5004)

D. Gramáticas Adaptativas

Gramáticas são formalismos baseados em regras de substituição [9]. Sua operação se baseia na aplicação sucessiva de regras, até que restem apenas terminais. Estas gramáticas permitem a auto-modificação de suas regras durante sua utilização.

E. Redes de Markov

Redes de Markov garantem que a probabilidade de um dado estado ser atingido depende exclusivamente do estado corrente da rede. Em [2], prova-se que a Máquina de Markov é equivalente ao Autômato Finito.

A ocorrência de uma transição entre estados faz com que seja executado um conjunto de ações adaptativas. Em Sistemas de Markov Adaptativos, as ações adaptativas de uma máquina podem atuar, também, nos conjuntos de regras de outras máquinas pertencentes ao sistema.

F. Tabelas de Decisão Adaptativas

Tabelas de decisão adaptativas são definidas a partir das tabelas de decisão tradicionais (não-adaptativas). Estas tabelas consistem de um conjunto de regras representadas por condições e ações correspondentes, que serão executadas caso a condição seja satisfeita. As regras são apresentadas nas colunas, enquanto as condições são apresentadas nas linhas. As ações, assim como as condições, são apresentadas nas linhas [16].

As tabelas adaptativas apresentam estrutura similar às tradicionais, porém, com linhas adicionais após as ações, que consistem das ações adaptativas. Estas linhas de ações adaptativas são divididas em ações anteriores e ações posteriores.

Quando uma ação adaptativa é executada, seu conjunto de regras poderá ser alterado. Desta forma, as colunas da tabela poderão ser modificadas, de maneira que novas colunas podem ser adicionadas, ou colunas existentes podem ser eliminadas.

A Figura 3 apresenta um exemplo de uma tabela de decisões adaptativa. Nesta tabela de decisões, as áreas 2, 3 e 4 apresentam as as funções adaptativas, suas chamadas e as ações correspondentes.

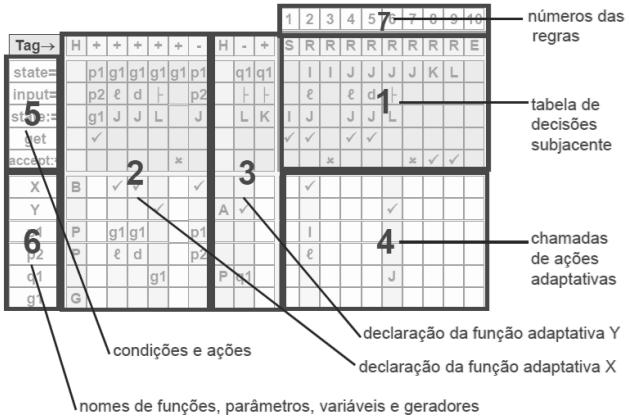


Figura 3- Exemplo de Tabela de Decisões Adaptativas (Fonte: Material de aula PCS5004)

G. Árvores de Decisão Não-Determinísticas Adaptativas

Árvores de Decisão Não-Determinísticas (*NDD-tree*) representam as decisões como folhas da árvore e os testes discretos em seus nós intermediários. Os resultados dos testes discretos indicam o caminho percorrido, desde o nó-raiz, para decidir um dado problema.

Quando buscando uma solução para determinado problema, a NDD-tree executa de maneira não-determinística, testando todos os caminhos possíveis. É possível ter mais de uma resposta para um problema tratado por NDD-tree.

[17] define a Árvore de Decisão Não-Determinística Adaptativa (ANDD-tree). Esta árvore utiliza NDD-tree como o mecanismo subjacente. A camada adaptativa permite que a NDD-tree altere sua estrutura. Operações de inspeção, adição e remoção de regras (nós de decisão) são possíveis neste tipo de dispositivo.

H. Autômatos de Estados Finitos Adaptativos

Os autômatos de estados finitos adaptativos são uma evolução, e simplificação, dos autômatos de pilha adaptativos [3]. Este dispositivo utiliza o autômato de estados finitos como mecanismo subjacente.

Neste tipo de dispositivo, o passo de operação muda juntamente com as modificações no conjunto de regras do autômato subjacente. O novo conjunto de regras associa a cada regra não-adaptativa um par de ações adaptativas (anterior e posterior).

Estes autômatos apresentam o mesmo poder computacional da Máquina de Turing, além de aceitar linguagens dependentes de contexto.

III. PROCESSAMENTO DE EVENTOS COMPLEXOS ADAPTATIVO PARA SEGURANÇA DA INFORMAÇÃO

O processamento de eventos (EP) consiste de um conjunto de técnicas e ferramentas que auxiliam na compreensão e controle de sistemas orientados a eventos [11]. A ideia principal é explorar relacionamentos causais, temporais e semânticos entre eventos, de forma a dar sentido a eles em tempo hábil de resposta [6]. Este comportamento permite identificar oportunidades e riscos tão logo um evento ocorra. Também possibilita que os eventos sejam diagnosticados e decisões sejam tomadas em um curto intervalo de tempo.

A partir do correlacionamento entre dois ou mais eventos, é possível identificar novos eventos. Eventos gerados a partir destas relações, e que podem ocorrer somente pela existência dos eventos relacionados, são conhecidos como eventos complexos, de forma que eventos complexos são abstrações de mais alto nível que representam uma situação, um evento composto, inferido pela ocorrência de outros eventos mais elementares e diretamente observados ou identificados [5]. Porém, é necessário que exista um contexto para que eventos complexos sejam identificados, não sendo suficiente a simples correlação entre os eventos. Ou seja, a combinação entre dois ou mais eventos necessita de significado semântico e

conhecimento situacional para gerar novos eventos complexos.

Os eventos, por si só, não carregam valor semântico que permita que conhecimento seja obtido a partir de sua análise ou da análise de seu comportamento. Uma forma de agregar significado aos eventos é pela utilização de ontologias. Estas ontologias permitem que as informações referentes aos eventos sejam estruturadas adequadamente e analisadas automaticamente em ambiente computacional, possibilitando a realização de previsões e a tomada das ações devidas. O processamento de eventos complexos (CEP) baseia-se na observação de que, em muitos casos, ações são disparadas não pela ocorrência de um evento isolado, mas pela composição de diversos eventos, que podem ocorrer em momentos distintos (temporalidade) e sob contextos diferentes (causalidade e semântica).

A. Middleware Adaptativo

A segurança da informação é uma área na qual utilizam-se técnicas de processamento de eventos. Esta área trata da proteção da informação quanto ao acesso, alteração e destruição não autorizados. Segundo [12], falhas de segurança em sistemas são exploradas por atacantes por 229 dias até serem detectadas. Após a detecção, há ainda um período de aproximadamente 30 dias até que os agressores sejam banidos da rede da vítima [10]. Segundo boletim [7], obteve-se um índice indicando que mais de 97% dos sistemas contém falhas de segurança, independentemente das camadas de segurança adotadas.

Devido à natureza das ameaças, a imprevisibilidade e a criticidade do tempo de resposta são características importantes. Desta forma, é necessário um monitoramento constante das ações para que, no momento em que um evento de ameaça à segurança seja identificado, contramedidas sejam disparadas, proporcionando o menor atraso possível para que sejam reduzidos os riscos e danos causados pelos eventos. Identificar eventos, ou séries de eventos, e reagir a eles de maneira apropriada e com a menor latência são os principais objetivos de sistemas nesta área. Em outras palavras, quanto antes uma ameaça for identificada, mais cedo uma ação poderá ser disparada em resposta, mitigando seus efeitos.

As abordagens tradicionais de processamento de eventos de segurança levam em conta apenas eventos considerados de segurança, de forma que os sistemas identificam como ameaças eventos previamente indicados como sendo relacionado à área de segurança. Ou seja, caso ocorra um evento que não conste da lista de eventos de segurança dos sistemas de detecção, este evento é desconsiderado no momento da verificação, não sendo submetido a qualquer tipo de análise. Outro problema das abordagens tradicionais é a falta de integração entre os eventos provenientes de fontes heterogêneas e distribuídas. Existem diversas ferramentas voltadas para a captura de eventos em diferentes áreas de cybersecurity, como SIEM, IDS/IPS, DLP, APT, entre outras. Porém, não existem ferramentas que correlacionem estes eventos, de forma a identificar padrões de diferentes ameaças que possam ter alguma relação entre si.

A associação de ontologias com o processamento de eventos complexos na área de segurança permite que questões como o apoio a processos de análise da causa raiz de conjuntos de eventos de segurança sejam passíveis de ações preventivas e/ou reativas de forma automática, a partir da definição do contexto de cada evento e a regra a ser aplicada em cada caso. Segundo [8], ontologias são capazes de inferir novo conhecimento a partir de informações existentes, de forma a possibilitar uma maior automatização dos sistemas, que podem reagir de forma automática a determinadas ações.

Com o intuito de obter melhores resultados, quanto aos tempos de detecção eventos ou identificação de eventos complexos e tempo de reação do sistema de segurança, é proposta uma abordagem adaptativa baseada em ontologias para a classificação de eventos de segurança. Esta abordagem visa correlacionar eventos provenientes de diversas fontes e adicionar semântica a estes eventos, possibilitando a identificação de eventos complexos a partir do monitoramento contínuo de eventos.

Pela utilização desta abordagem, espera-se obter uma maior consciência situacional, possibilitando tomadas de decisão baseadas no significado dos eventos. Adicionalmente, torna-se possível a adaptação automática e online da ontologia utilizada através da realimentação de eventos previamente identificados de forma a evolui-la dinamicamente. Esta realimentação se daria a partir da análise de dados históricos, por exemplo, que possibilitariam a atualização da ontologia. A capacidade de adaptação possibilitaria antecipar as conclusões a respeito do espaço cibernético de contexto e, consequentemente, diminuir o tempo de reação a uma ameaça.

B. CEP com Middleware Adaptativo

A solução proposta é composta por duas partes. A primeira parte corresponde à um conjunto de ferramentas de processamento de eventos complexos já existentes no mercado. Estas ferramentas são responsáveis por tratar os eventos capturados e, se possível, disparar as ações necessárias para tratá-los.

Caso os eventos sejam desconhecidos pelo conjunto de ferramentas, os eventos são passados, na forma de estímulos, para a segunda parte da abordagem proposta. Esta segunda parte corresponde à parte adaptativa da proposta. A parte adaptativa é composta por uma máscara, responsável por verificar um conjunto condições, a partir dos estímulos. Estas condições servirão de entrada para a Tabela de Decisão.

Na Tabela de Decisão estão configuradas ações adaptativas que permitem alterar a ontologia, de forma que próximos conjuntos de estímulos (eventos complexos) possam ser tratados prontamente pela solução.

Além disso, existem ações adaptativas que alteram a própria Tabela de Decisão, de forma a refletir as alterações realizadas na ontologia.

Por fim, a Tabela de Decisão contém conjuntos definidos de ações, que permitem disparar as respostas adequadas aos conjuntos de estímulos, por meio das ferramentas CEP.

A abordagem proposta, descrita acima, é apresentada, graficamente, abaixo, por meio da Figura 4.

Esta abordagem foi proposta, inicialmente, para tratar eventos relacionados à área de cybersecurity. Por isso, a ontologia utilizada descreve o universo relacionado à esta área.

Porém, durante o desenvolvimento da mesma, foi observado que, ao se alterar a ontologia, pode-se tratar eventos de quaisquer áreas.

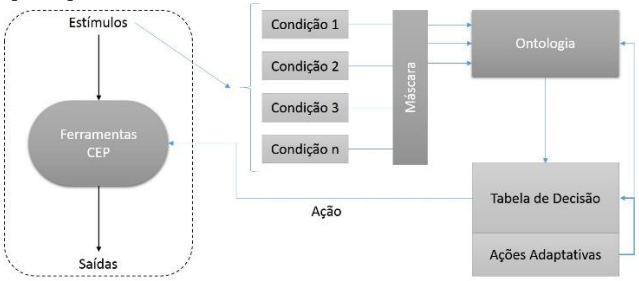


Figura 4 - Abordagem Proposta

IV. CONCLUSÕES

O trabalho apresentado possui uma vertente na análise e processamento automatizado de eventos no contexto de segurança cibernética. O desenvolvimento de uma abordagem para CEP visa auxiliar na tomada de decisão e reação de sistemas frente a identificação de ameaças à sistemas de informação.

A partir do objetivo do trabalho, podem-se definir alguns resultados esperados deste trabalho, como a definição de uma abordagem para CEP na área de cybersecurity que se utilize dos benefícios das ontologias para aumentar o significado dos eventos. Desta forma, espera-se um aumento nos níveis de segurança da informação.

A definição de formas para a verificação da aplicabilidade da abordagem proposta também faz parte dos resultados esperados.

Os resultados obtidos durante o trabalho serão analisados e comparados com resultados obtidos a partir dos mesmos conjuntos de entrada em sistemas de código aberto já existentes e utilizados pela comunidade. Desta forma, poderão ser avaliados a eficiência, eficácia e grau de melhoria obtidos pela proposta.

Como forma de avaliação, definida de maneira preliminar, será desenvolvido um software que utilizará uma abordagem iterativo-incremental, com foco nas disciplinas de Requisitos, Análise e Projeto, Casos de Uso, Implementação, Testes, Implantação e Gerenciamento de Configuração. Este software terá seu desempenho analisado e os resultados obtidos a partir de sua execução comparados aos resultados de softwares de código aberto com melhores resultados existentes atualmente.

REFERÊNCIAS

[1] JR Almeida. STAD-Uma Ferramenta para Representação e Simulação de Sistemas. Através de Statecharts Adaptativos. PhD thesis, Tese de Doutorado, Escola Politécnica, Universidade de São Paulo, São Paulo, 1995.

[2] Bruno Arantes BASSETO. Um sistema de composição musical automatizada, baseado em gramáticas sensíveis ao contexto, implementado com formalismos adaptativos. PhD thesis, Dissertação (mestrado), EPUSP, São Paulo, 2000.

[3] Amaury Antônio de CASTRO JUNIOR. Aspectos de projeto e implementação de linguagens para codificação de programas adaptativos. PhD thesis, Universidade de São Paulo, 2009.

[4] K Mani Chandy and W Roy Schulte. Event Processing: Designing IT Systems for Agile Companies. McGraw Hill New York, 2010.

[5] Gianpaolo Cugola and Alessandro Margara. Processing flows of information: From data stream to complex event processing. ACM Computing Surveys (CSUR), 44(3):15, 2012.

[6] Opher Etzion et al. The event processing manifesto. In Dagstuhl Seminar Proceedings (10201), pages 1–60, 2011.

[7] Inc. FireEye. Cybersecurity's Maginot Line: A Real-world Assessment of the Defense-in-Depth Model. https://www2.fireeye.com/real-world-assessment.html, 2013 (Acessado em 07/05/2015).

[8] A. L. Gonzalez, R.Willrich, D. Izidoro, and C. A. S. Santos. Representação aberta e semântica de anotações de incidentes em mapas web. Simpósio Brasileiro de Sistemas da Informação, 2013.

[9] Margarete Keiko IWAI. Um formalismo gramatical adaptativo para linguagens dependents de contexto. Departamento de Computação e Sistemas Digitais (PCS)-Escola Politécnica, Tese de Doutorado, Escola Politécnica, Universidade de São Paulo (USP), São Paulo, SP (in portuguese), 2000.

[10] S Jayson. Cost of cyber crime study: Global report. Technical report, Technical Report October, Ponemon Institute, 2013.

[11] David Luckham. The power of events, volume 204. Addison-Wesley Reading, 2002.

[12] Mandiant. M-trends: Beyond the breach (2014 threat report). Technical report, 2014.

[13] João José Neto. Contribuições à metodologia de construção de compiladores. São Paulo: Tese de Livre Docência, USP, 1993.

[14] João José Neto. Adaptive automata for context-dependent languages. ACM Sigplan Notices, 29(9):115–124, 1994.

[15] João José Neto. Adaptive rule-driven devices-general formulation and case study. In Implementation and Application of Automata, pages 234–250. Springer, 2001.

[16] João José NETO. Adaptive rule-driven devices-general formulation and case study. In Implementation and Application of Automata, pages 234–250. Springer, 2002.

[17] Hemerson PISTORI, João José NETO, and Mauro Conti PEREIRA. Adaptive nondeterministic decision trees: general formulation and case study. INFOCOMP Journal of Computer Science, 5(1):35–40, 2006.

Personalização, "Customização", Adaptabilidade e Adaptatividade

R. Caya, J.Neto

***Abstract*— This paper presents a review about common terminology used in the field of computer technology to refer to adaptive systems and the difference between each one. The main objective is to establish a clear guide to use several forms of accurately describing to behavior of dynamic systems in order to achieve not only a proper categorization, but also a wider visibility in the academic field. The impact of the use of the precise term or group of terms are explicitly point out by the cases in which searches into academic databases come out with much unexpected results.**

***Keywords*— Terminology, Adaptivity, Adaptability, Personalization, Customization, Glossary.**

I. INTRODUÇÃO

ATUALMENTE, dentro do escopo tecnológico existe uma grande diversidade de termos que são associados com tecnologias que admitem algum tipo de mudança nas suas configurações padrão. De maneira particular aquelas tecnologias que manifestam um mecanismo que permite a elas mesmas serem responsáveis pela decisão de aplicar uma mudança na sua própria configuração, tecnologias que implementam adaptatividade, tem sido o foco dos trabalhos de pesquisa no Laboratório de Tecnologias Adaptativas (LTA) da Universidade de São Paulo. Existem cada vez mais estudos sobre sistemas e dispositivos que possuem comportamento dinâmico, mas nem todas as publicações que os descrevem costumam empregar uma terminologia uniforme e consistente para designar os fenômenos e as atividades estudadas [1]. A falta de um consenso sobre a definição do que é a adaptatividade, e a proliferação de termos para nomear esses sistemas têm dificultado a categorização de pesquisas e a análise do desenvolvimento nesta área, por exemplo através de um explorador automático para repositórios de pesquisa. Assim, muitos trabalhos que não necessariamente implementam todas as características da adaptatividade utilizam este termo como guarda-chuva e não com o significado específico que ele realmente tem. Da mesma maneira, existem trabalhos na literatura que para evitar um uso incorreto colocam termos correlatos que de alguma maneira procuram expor o comportamento das tecnologias que desenvolvem. Existe assim uma necessidade de esclarecer as definições dos termos mais utilizados para descrever este tipo de tecnologias em busca de uma padronização da terminologia, uma base conceitual concreta para a uniformização do uso de algumas das palavras e expressões mais utilizadas que possa servir de guia tanto para a elaboração de buscas de pesquisas neste campo quanto para o uso adequado da terminologia em novos projetos de pesquisa.

O uso adequado da terminologia é um fator importante dentro da elaboração dos projetos de pesquisa pois permite não só caracterizar o trabalho desenvolvido, mas também influencia a visibilidade de dito trabalho. Esse ponto é especialmente sensível quando o foco da pesquisa se refere a um meio e não a um fim, isto é, quando o interesse está concentrado principalmente na maneira em como foi resolvido um problema no lugar do problema particular. Esse é o caso das tecnologias adaptativas onde a principal contribuição está no mecanismo, a maneira, que suporta a resolução deum problema, e que, portanto, pode ser aplicado em diferentes tipos de problemas, em diferentes áreas de conhecimento, com diferentes níveis de complexidade e para diferentes finalidades. Assim é importante estabelecer uma base sobre a qual os pesquisadores possam identificar trabalhos que implementam estas tecnologias, assim como também oportunidades de pesquisa em novas áreas, e até permitir a elaboração de indicadores sobre o crescimento da pesquisa na área. De maneira particular, as pesquisas relacionadas com tecnologias que permitem gerenciar mudanças nas suas configurações têm recebido importante atenção como consequência das novas necessidades expostas pelos usuários na chamada Sociedade da Informação e devido a isso é impostergável a elaboração de uma ferramenta que permita distinguir as características associadas aos termos mais usados: personalização, customização, adaptabilidade e adaptatividade.

A motivação deste trabalho é incorporarintríseca clareza no entendimento e uso dos termos apresentados, não só dentro da área tecnológica, mas também dentro de outras áreas que eventualmente possam precisar da descrição deste tipo de comportamento. Com essa finalidade são apresentadas, em primeiro lugar, as definições gerais de cada um dos termos desde o ponto de vista linguístico para logo incorporar o seu significado em termos de ferramentas tecnológicas.

O objetivo deste trabalho é estudar as definições de cada um desses termos, tanto de maneira geral quanto específica no escopo tecnológico, identificar com clareza as características que cada uma implica, expor as relações e diferenças entre elas e oferecer alguns exemplos de sistemas que implementam as técnicas correspondentes.

As próximas seções deste artigo estarão organizadas da seguinte maneira: na seção 2 será abordada a necessidade de adaptação como parte intrínseca ao ser humano, começando pela necessidade de adaptação para sobrevivência da espécie

R. Caya, Universidade de São Paulo (USP), São Paulo, São Paulo, Brasil, rosalia.caya@usp.br.com

J. J. Neto, Universidade de São Paulo (USP), São Paulo, São Paulo, Brasil, joao.jose@poli.usp.br

até a necessidade de adaptação para necessidades individuais. Na seção 3 será apresentada a evolução da demanda de adaptações por parte dos usuários dentro da área das tecnologias computacionais, as principais razões e as consequências que isso traz para o desenvolvimento de novas tecnologias. A seção 4 apresentará as definições dos quatro termos mais utilizados para a representação de tecnologias que implementam diferentes tipos de adaptações e finalmente a relação que existe entre elas. As principais características e diferenças entre tecnologias que implementam mecanismos de personalização, customização, adaptabilidade e adaptatividade são expostas na seção 5. A seção 6 apresentará os termos correlatos, mas frequentemente encontrados na literatura para descrever tecnologias que implementam alguns desses comportamentos, mas que optaram por utilizar outra nomenclatura. Por fim, na seção 7 estarão as considerações finais.

II. A ADAPTAÇÃO COMO NECESSIDADE DO SER HUMANO

Os quatro termos que são objeto de estudo neste trabalho possuem como base a mesma raiz semântica: a capacidade de ajustar alguma coisa ou comportamento a uma nova condição, a capacidade de assimilar uma mudança. Tal capacidade é uma das características principais dos organismos vivos, particularmente é a responsável pela evolução e sobrevivência das espécies. A presença de mecanismos adaptativos nos seres vivos é colocada em evidência em processos de diferente natureza e complexidade, desde a regulação que faz a pupila da quantidade de luz que penetra no olho em diferentes situações de luminosidade até as lentas e usualmente inconscientes modificações das atividades sociais em ajuste para o entorno cultural.

O termo "adaptar" é definido no Dicionário Aurélio [26] da Língua Portuguesa como tornar apto, fazer que uma coisa se combine convenientemente com outra. Já os dicionários de Língua Inglesa fornecem as seguintes definições: [14]mudar alguma coisa de maneira que ela funcione melhor ou seja mais adequada para algum propósito, [22] mudar gradualmente seu comportamento e atitudes com a finalidade de ter sucesso numa nova situação. De maneira geral, pode ser dito que o termo "adaptar" descreve a mudança aplicada em alguma entidade com a finalidade de torná-la mais adequada para desempenhar uma tarefa, eventualmente trata-se de uma nova tarefa ou ela é desempenhada em novas circunstancias. A Figura 1 apresenta a etimologia da palavra "adapt" que significa literalmente "ajustar" para alguma condição ou propósito.

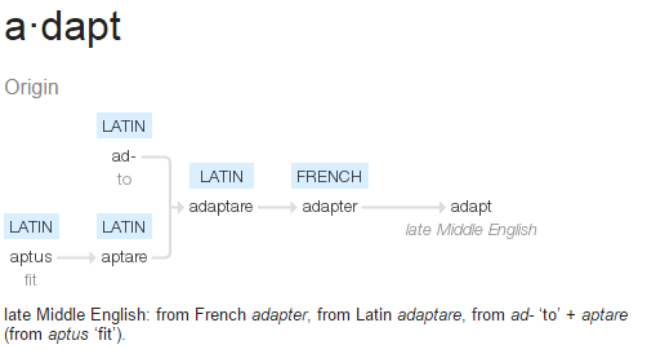


Figura 1. Etimología da palavra "adapt". Gerada pelo Google como resultado da busca com as palavras-chave "adapt origin"

Particularmente, os seres humanos têm manifestado sua necessidade por adaptações como uma caraterística da sua própria história. De acordo com [2] os seres humanos têm respondido de diferentes formas às mudanças no nosso ambiente ao longo da história: respostas biológicas (ajustes no nosso genoma, no nosso desenvolvimento e aclimatação) e respostas em termos de práticas culturais e tecnologias. De fato, as pesquisas indicam [3] que o ser humano sofreu uma adaptação exuberante nos últimos duzentos mil anos. Nos parágrafos seguintes desenvolveremos este último tipo de resposta por ser ela o foco deste trabalho.

As adaptações manifestas através de práticas culturais e tecnológicas são as responsáveis pela transformação não só das nossas sociedades, construindo complexas instituições culturais, mas também da mudança no nosso entorno físico para fazê-lo cada vez mais adequado para o desenvolvimento das nossas atividades cotidianas.

Assim, não só nós, humanos, ajustamo-nos às condições do entorno, mas também exercemos uma força de mudança nesse mesmo entorno. De maneira geral, pode se dizer que este é um primeiro nível de ajuste do entorno ao ser humano, não como entidade pessoal, mas sim como a presença da espécie. Com o passar do tempo surge a necessidade por mais um nível de ajuste, dessa vez focado nas necessidades de comunidade. Assim, por exemplo as necessidades que algumas minorias dentro da sociedade manifestam: mulheres, crianças, velhos, portadores de deficiência, entre outros. Finalmente, um último nível de ajuste é aquele no qual o foco de dita mudança é o indivíduo. Neste último estágio as mudanças são feitas para a satisfação expressa de um único indivíduo, e procurando satisfazer suas necessidades levando em conta suas preferências.

Um dos primeiros exemplos desses níveis de ajuste ou modificações está na construção de moradias. Num primeiro momento o objetivo era a construção, num entorno inóspito, de moradias que permitiam a sobrevivência da espécie. Tempos depois, essas moradias foram evoluindo, incorporando melhorias: novos materiais, novas técnicas de construção, novos formatos de moradias, maior resistência as condições meteorológicas, entre outros.

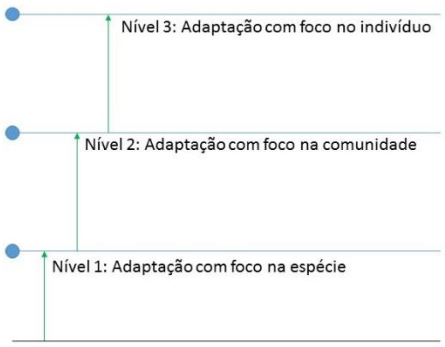


Figura 2. Representação dos diferentes níves de adaptação na história da humanidade.

Na metade do século vinte um novo nível de ajuste surgiu: a preocupação por fornecer dentro das construções alternativas acessíveis para pessoas portadoras de deficiência física, assim novos ajustes foram incorporados para satisfazer as necessidades desta comunidade. Finalmente, na atualidade existem construções que são realizadas para cobrir as expectativas, desejos, preferências e requisitos particulares. Um exemplo desses casos são as construções que, incorporam complexos sistemas de domótica e automação, assim como segurança e controle, atingindo o último nível de ajuste.

III. EVOLUÇÃO NA DEMANDA POR ADAPTAÇÕES COMPUTACIONAIS

Desde a disponibilização do computador pessoal o uso das tecnologias tem experimentado um crescimento exorbitante. Na atualidade, o uso de dispositivos e ferramentas tecnológicas para a realização de tarefas cotidianas é comum e frequente. Essa proximidade entre humano e máquina tem exposto uma nova necessidade a ser analisada e avaliada: a necessidade de interação de qualidade entre esses dois agentes. Para entender o processo de interação entre humano e máquina é preciso entender as características do que chamamos interação humana, pois é esse o parâmetro que fixará os requisitos para nossa interação com a tecnologia.

O processo de interação entre seres humanos é uma atividade altamente complexa e, de fato, uma experiência única para cada participante. Isso quer dizer que cada um dos envolvidos possui um conjunto de características no momento da interação que permite-lhe interpretar corretamente a mensagem sendo transmitida. Dita interpretação tem base não só no conteúdo explícito da mensagem, mas também em outro tipo de informações prévias, experiências, dos participantes, por exemplo: o conhecimento sobre o entorno onde é efetuada a interação (o assunto, a conjuntura sociocultural, o ambiente físico, entre outros) e suas próprias preferências. Essas informações permitem-lhe preencher os possíveis vazios, resolver ambiguidades, e inclusive resgatar informações implícitas e subjacentes. Esta riqueza de informações presentes no processo de interação entre humanos é o que permite a transmissão efetiva de ideias, abstratas e complexas, que tem permitido, em grande medida, a evolução da raça humana até o ponto de governar o planeta no qual moramos.

A tecnologia, por sua parte, tem sido criada com o objetivo de assistir aos seres humanos na execução de diversas tarefas. Desde aquelas que implicam o uso de força física até processos de diferentes complexidades mentais, alguns altamente sofisticados e outros parte do dia a dia. É assim que na atualidade nossas atividades cotidianas encontram-se cada vez mais na necessidade de interagir com sistemas computacionais em alguma de suas diversas formas: *pc*, *smartphone*, *tablet*, *netbook*, *notebook*, sistemas dedicados (bancos, bibliotecas, sistemas privados, entre outros). Nessas circunstâncias ficou cada vez mais evidente a necessidade de enriquecer a interação entre estes dois agentes, o humano e a máquina. A razão é que o processo de interação entre humano e máquina, seguindo uma abordagem simplificada, não só gera insatisfação para a maioria de usuários, mas também resulta em dificuldades importantes no caso de usuários com necessidades especiais.

Ao longo dos anos, diferentes áreas de pesquisa têm investido no desenvolvimento de técnicas e mecanismos que permitam aos sistemas computacionais colher, armazenar, classificar e utilizar a riqueza de informações correspondentes ao contexto da interação. Particularmente na área de HCI (*Human-Computer Interaction*) tem sido dedicados esforços particulares para responder à demanda dos usuários de uma interação que se ajuste mais às características de cada um deles.

Inicialmente, com a aparição do computador pessoal, a demanda dos usuários foi a adaptação dessa nova tecnologia para "o usuário comum", isso quer dizer, para aquele usuário que não é um cientista, matemático ou da computação, especializado no uso de linguagens de programação complexas e códigos artificiais e pouco intuitivos. De alguma maneira podemos fazer uma analogia com o primeiro nível de adaptação apresentado na seção II, o usuário demandava uma adaptação para a espécie "usuário comum". Assim, a resposta do lado tecnológico foi a criação de mecanismos de interação mais intuitivos e simples para o usuário final, por exemplo foram criados novos dispositivos de entrada/saída para assistir o processo de interação (mouse, impressora, aparelhos de som, entre outros), foi desenvolvida uma nova interface gráfica para o usuário final, no lugar de só disponibilizar a interação através da console de linha de comando, foi implementado o método WYSIWYG ("*What you see is what you get*") para permitir ao usuário conhecer os efeitos da sua interação com o computador, foi desenvolvido o WIMP ("*Windows, Icons, Menu and Point*") um método de interação com o usuário que permite gerenciar uma interface gráfica baseado nesses quatro elementos.

Com o decorrer do tempo, essas adaptações de primeiro nível foram assimiladas, assim para finais da década de 1980 a maioria de computadores possuía um sistema operacional que era apresentado através duma interface gráfica de usuário e contava com diferentes dispositivos de entrada/saída. No entanto, a necessidade de um novo nível de adaptação surgiu. A premência de fornecer mecanismos alternativos de interação

para aqueles usuários com necessidades especiais. Assim, desde começo da década dos noventa, diferentes pesquisas procuraram oferecer tecnologias que permitam a incorporação de usuários com características diferenciadas do estereótipo de "usuário padrão", alguns deles são: crianças, idosos e portadores de deficiência física e cognitiva. A demanda por adaptação das tecnologias neste estágio significava oferecer acessibilidade e integração desses também membros da sociedade humana dentro da emergente sociedade da informação. Assim, estas adaptações pertencem a um nível dois, já não têm como foco o usuário de maneira geral, mas um tipo particular de usuário que possui necessidades diferentes que devem ser levadas em conta no processo de interação com a tecnologia.

Finalmente, com o início do século vinte um, uma nova geração de usuários, os chamados nativos digitais, colocou em evidência sua necessidade por maior nível de interação entre as tecnologias presentes e suas atividades. Já não é suficiente ter computadores do tipo PC, pois eles são fixos, agora é preciso contar com dispositivos que possam ser transportados para onde o usuário for. O "boom" da internet no começo dos anos noventa teve um novo impulso com a criação de dispositivos móveis, e como resultado nasceram as áreas de computação em nuvem, internet das coisas, computação vestível, entre outras. Esses avanços tecnológicos e sua rápida exposição para os usuários comuns trouxe como consequência que esses usuários começaram a demandar uma maior consideração não só para suas necessidades básicas, mas também para suas preferências, de maneira que a interação entre eles e os elementos tecnológicos se tornasse uma experiência mais próxima da chamada interação natural, a interação entre humanos, e, portanto, incorporasse informações particulares. Em resposta a este fenômeno diversas pesquisas têm sido desenvolvidas, novas modalidades de interação têm sido estudadas e analisadas, e como resultado novas tecnologias têm sido desenvolvidas anunciando a capacidade de satisfazer esta necessidade de adaptação.

Este trabalho quer destacar quatro termos presentes na nova terminologia forjada e fortemente entrelaçados com a capacidade de incorporar adaptações, esses são: personalização, customização, adaptabilidade e adaptatividade. A seguinte seção apresenta as definições gerais básicas desses termos.

IV. DEFINIÇÕES GERAIS DOS TERMOS

A. *Personalização*

O dicionário Aurélio da Língua Portuguesa define "personalização" como o ato ou efeito de personalizar, que por sua vez é definido como: tornar pessoal, dar caráter original a um objeto fabricado em série, e adaptar às preferências do usuário[25].

O dicionário Merriam-Webster[10], Longman[16] e Cambridge[22], de Língua Inglesa, definem "personalize" como: mudar ou planejar alguma coisa com a finalidade de ela ficar mais ajustada às necessidades ou requerimentos de uma pessoa, empresa, entre outros. O Collins[17], um outro conhecido dicionário de Língua Inglesa define "personalize" como: dotar alguma coisa com qualidades ou características pessoais ou individuais. Assim, é possível definir de maneira geral a personalização como o ato de mudar alguma coisa para fazê-la mais adequada com as qualidades ou características de uma entidade em particular.

Note-se que em todas as definições apresentadas a especificação de quais mudanças irão ser aplicadas não provêm explicitamente da entidade objeto da adaptação.

B. *"Customização"*

A palavra "customização" não é reconhecida como uma palavra formal dentro da Língua Portuguesa, no entanto, o seu uso tem sido aceito e generalizado não só no ambiente coloquial, mas também dentro das diferentes áreas acadêmicas relacionadas à tecnologia. Neste trabalho vamos utilizar o termo "customização" como um anglicismo do termo "customization".

O dicionário Merriam-Webster[11] define "customization" como a ação de mudar alguma coisa em ordem de cobrir as necessidades de uma pessoa, organização, negócio, entre outros. O dicionário Collins[19] descreve a ação de "customizar" como as mudanças que uma pessoa faz na aparência ou características de alguma coisa para fazê-la mais adequada aos seus gostos ou necessidades. O dicionário Cambridge[24] define o termo como as mudanças que são feitas em alguma coisa de acordo com as necessidades do cliente ou usuário. Assim, de maneira geral é possível dizer que "customização" é a ação de aplicar mudanças em alguma coisa com o propósito de satisfazer os requerimentos de uma pessoa em particular.

Note-se que em contraste com o termo "personalização" a customização contém em todas as definições apresentadas a menção explícita da origem das mudanças a serem aplicadas: elas são requerimentos do usuário. Portanto, é o usuário que tem o poder de decisão sobre quais são as características que irão sofrer mudanças.

C. *Adaptabilidade*

O dicionário Aurélio[27] define "adaptabilidade" como a qualidade ou capacidade do que é adaptável, e define adaptável como alguma entidade que pode ser mudada, em alguma das suas caraterísticas, com a finalidade de ajustá-la para uma nova situação. Uma entidade que possui adaptabilidade admite a possibilidade de aplicar mudanças nas suas características padrões. O dicionário Collins[17] define adaptabilidade como a capacidade de alguém ou de alguma coisa de poder ser mudada com a finalidade de fazê-la mais adequada para um novo propósito ou situação. O dicionário Merriam-Webster[15] define este termo como aquele que descreve a característica de alguma coisa de mudar ou ser mudada a fim de se adequar ou trabalhar melhor em alguma situação ou para algum propósito: o fato de ser apto ou capaz de se adaptar ou de ser adaptado.

É importante notar que em nenhum momento a definição de adaptabilidade faz menção sobre o agente executor da mudança, assim, uma entidade pode ser adaptada por um agente externo que tem o poder de decisão sobre as mudanças a serem feitas, as quais são aplicadas pela entidade a ser adaptada, ou, num caso particular, ela mesma pode governar o processo de decisão e aplicação das mudanças na sua própria estrutura, como veremos a continuação.

D. Adaptatividade

O dicionário Aurélio define adaptativo como a qualidade de alguma coisa que permite que a mesma esteja própria para se adaptar[28]. A sua vez, o dicionário define o termo "adaptar" como a ação de fazer com que uma coisa se combine convenientemente com outra[26]. O dicionário Collins, por sua parte, define "adaptivity"[20] como o estado de alguma entidade no qual ela tem a capacidade para adaptação, sendo esta entendida como o ato ou processo de ajuste de alguma coisa para se acomodar numa nova condição ou necessidade.

Os dicionários Oxford e Cambridge descrevem a adaptatividade como a capacidade de uma entidade de se ajustar às novas condições e mudanças no seu entorno.

Assim, de maneira geral é possível definir a adaptatividade como a capacidade que possui alguma coisa para conseguir efetuar uma mudança a fim de se adequar a uma nova situação. Esta palavra também expressa um estado, condição, um grau ou medida de dita capacidade.

É importante notar que as definições dos dois primeiros termos, personalização e "customização", oferecem informação sobre qual é a entidade que exerce a tomada de decisão sobre quais serão as mudanças a serem executadas. Assim tanto a personalização quanto a "customização" nascem da necessidade de uma tarefa, atividade, ferramenta, em geral um objeto, de considerar diferentes características dentro do seu ambiente de execução para se desempenhar de uma maneira adequada e atingir o objetivo para o qual foi criado.

No entanto, os seguintes dois termos falam tanto da capacidade para aplicar ditas mudanças quanto do organismo executor delas.

A Fig. 3 apresenta a relação entre esses quatro termos. Tanto personalização quanto "customização" se referem a mecanismos pelos quais é possível incorporar alterações dentro das características duma entidade com a finalidade de torná-la mais adequada às novas condições de entorno ou necessidades do usuário. No entanto, a diferença que existe entre elas, torna inviável o uso indiscriminado duma ou outra. Por um lado, um fato interessante evidenciado por esta representação é que a "customização" é possível através de adaptabilidade, mas não de adaptatividade. Isso deve-se ao fato de que na "customização" o controle sobre as mudanças a serem feitas está na pessoa, no cliente, no usuário, enquanto que a adaptabilidade implica que dito controle é uma propriedade da própria entidade sendo mudada.

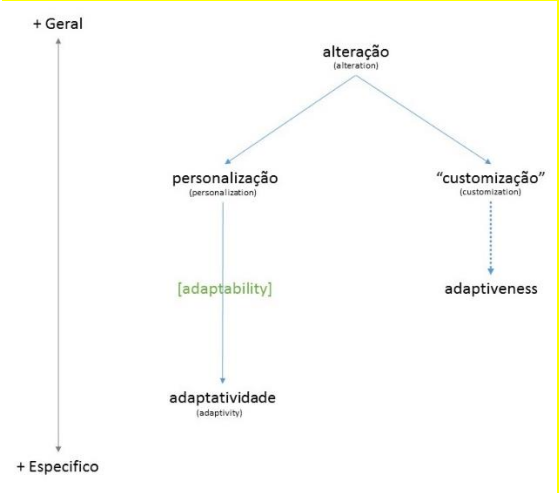


Figura 3. Relação entre os termos adaptação, personalização, customização, adaptabilidade e adaptatividade.

Da mesma maneira, notase que a customização não é implementada unicamente através da adaptabilidade, existe a possibilidade de criação de novas entidades (variações da entidade padrão) que incorporam as mudanças exigidas pelo usuário. Um exemplo deste caso na área tecnológica são os chamados dispositivos "ad-hoc".

V. CARACTERÍSTICAS NO ESCOPO TECNOLOGICO

Dentro da área tecnológica cada um dos termos que são foco deste trabalho apresenta particularidades. Algumas delas tão sutís que o perigo de utilizar o termo equivocado é mais frequente do que o recomendado.

O termo personalização é frequentemente utilizado no entorno de aplicações web e programação de aplicativos altamente flexíveis, ou também chamados *responsive software*. Neste entorno personalização significa que a ferramenta tecnológica conta com algum mecanismo que permite que ela mesma possa inferir algumas das preferências e necessidades do usuário a maneira de oferecer para ele as recomendações que melhor se ajustam com esses valores. Assim, por exemplo, o website da loja online Amazon é considerado como uma das ferramentas que mais tem trabalhado e incorporado o conceito de personalização, de maneira que é a própria interação que o usuário mantém com o site o que vai fornecendo informações para que as recomendações de produtos sejam de cada vez mais interesse para o usuário particular.

Já o termo "customização" diz que as mudanças a serem efetuadas provem de uma ordem expressa do usuário, o qual através de diretivas é capaz de escolher quais que são os novos valores que se adequam mais para suas necessidades em dita ferramenta. Existem duas maneiras de "customização" no entorno tecnológico, o primeiro deles se refere à aplicação de mudanças dentro da mesma instância de dispositivo, e o segundo obedece à criação (entendido como planejamento, design e desenvolvimento) de uma nova ferramenta tecnológica que é uma variação da primeira e incorpora as

mudanças especificadas para satisfazer as necessidades do usuário. Assim, um exemplo do primeiro caso é o ajuste dos níveis de contraste e brilho na tela do computador, o nível de volume para os autofalantes, as configurações sobre os aplicativos padrão para abertura de arquivos, entre outros. No segundo caso um exemplo são os dispositivos dedicados para interação com pessoas portadoras de deficiência. Dentro desse contexto é importante notar que a "customização" pode ou não implementar adaptabilidade ("*adaptiveness*") para a incorporação de mudanças, no entanto é teoricamente errado falar de adaptatividade como mecanismo para implementar "customização".

A relação entre os termos adaptabilidade e adaptatividade é um pouco mais complexa por tratar-se de conceitos em diferente nível de detalhe, mas que historicamente têm sido utilizados como equivalentes. Em [4] a relação entre ambos os termos é apresentada como um espectro, de maneira que em alguns sistemas tem-se tanto mecanismos que implementam adaptatividade quanto outros que atuam de maneira adaptável. A Fig. 4 apresenta alguns pontos neste espectro.

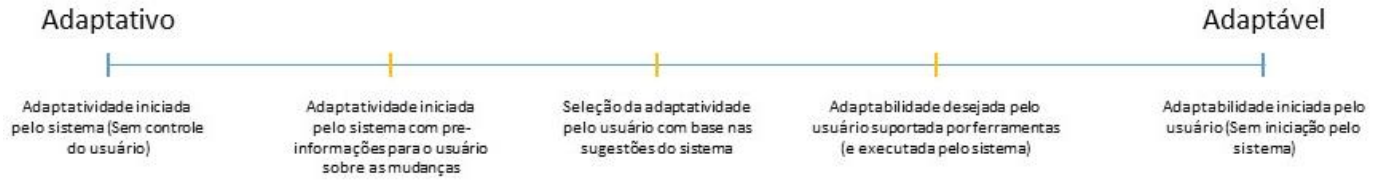


Figura 4. Espectro da relação entre os termos adaptativo e adaptável. Adaptado de [4].

Adaptabilidade dentro da área de sistemas computacionais deve ser entendida como a capacidade de um sistema de aplicar mudanças que são iniciadas como resposta a intervenções externas, por exemplo: comandos explícitos de usuário, eventos específicos, condições especiais detectadas, entre outros. A adaptabilidade é caracterizada pela participação não ativa do sistema, e a escolha por parte de um agente externo de alguma funcionalidade pré-existente que implemente o comportamento correspondente com a alteração desejada. Adaptatividade, pelo contrário e como especificado em [1], é a propriedade que apresenta um sistema, processo ou dispositivo computacional, que lhe permite, sem a interferência de agentes externos, tomar a decisão de executar alguma mudança, de maneira autônoma e dinâmica, no seu próprio comportamento como consequência de algum estímulo, entendido como alguma variação nas condições no seu ambiente de execução, e a sua configuração corrente[5]. No caso desses sistemas não só devem se levar em conta as consequências diretas das aplicações das mudanças, mas também os chamados efeitos colaterais que aparecerem como parte de dita adaptação.

VI. TERMOS RELACIONADOS

Nesta seção são apresentados os termos frequentemente usados na literatura para descrever entidades que possuem comportamentos que admitem mudanças dinâmicas.

A. Reconfigurável, autoconfigurável, y configuração dinâmica

O termo "sistema reconfigurável" [6] nasceu dentro da área de design de componentes de hardware, especificamente FPGAs (*Field Programable Gates Array*), pelo qual os resultados das pesquisas com esse termo estão relacionados com trabalhos nas áreas de arquitetura de computadores, design de elementos de hardware e aplicativos de software especializados no aproveitamento das capacidades de hardware para melhorar o desempenho no cômputo de alguns tipos de informações [7]. De maneira mais detalhada em [8] é indicado que "reconfiguração dinâmica" se refere à reconfiguração de algum dispositivo em tempo de execução, mas que não necessariamente tem suporte de software, o próprio usuário pode realizar dita mudança. Já o termo autoconfiguração implica que o sistema de maneira autônoma realizará as mudanças pertinentes.

B. Auto-ajustável, auto-modificável, de alteração dinâmica

A computação auto-ajustável se refere ao modelo computacional no qual os cômputos se ajustam a qualquer mudança externa em seus dados automaticamente [9] com a finalidade de fornecer uma melhora no desempenho das tarefas a serem executadas. Como indicado em [9] a maioria dos métodos utilizados para a construção de máquinas auto-ajustáveis utilizam algoritmos de propagação das mudanças para conseguir realizar o mínimo de computo necessário para responder às novas estruturas de dados de entrada fornecidos. Por outro lado, os termos "de alteração dinâmica" e "auto modificável" estão mais relacionados com a característica de um programa de efetuar alterações no seu próprio código,

algumas das linguagens de programação que permitem a escrita deste tipo de código são chamadas linguagens dinâmicas.

C. *Sistemas auto-adaptativos*

O termo "self-adaptive system" se refere aos sistemas computacionais que são autônomos na decisão e aplicação de mudanças nas suas próprias configurações com a finalidade de desempenhar de maneira bem-sucedida a tarefa para a qual foi desenvolvido, superando as possíveis dificuldades introduzidas no ambiente de execução.

D. *Observações sobre os resultados das buscas com as palavras relacionadas como palavras-chave*

Algumas das buscas realizadas durante a elaboração deste documento apresentam algumas observações nos resultados obtidos que devem ser levados em conta no momento de realizar a filtragem das informações. A seguir são detalhadas as observações mais marcantes:

- Dispositivos Adaptativos e Tecnologias Adaptativas: a busca feita sob essas palavras chaves normalmente retorna pesquisas relacionadas com dispositivos e tecnologias assistivas para fornecer suporte a pessoas portadoras de deficiências.
- Tecnologias auto-ajustáveis: quando a palavra-chave da busca é "self-adjusting technology" os resultados estão, na maioria das vezes, relacionados com tecnologia de colchões, o qual claramente é uma saída inesperada. Da mesma maneira, quando a busca é feita com a palavra-chave "self-adjusting technique" os resultados são, majoritariamente, da área de quiropraxia.

VII. CONCLUSÃO

Diante do exposto, conclui-se que, o uso adequado da terminologia relacionada à pesquisa tem impacto em diferentes momentos do desenvolvimento da mesma, e é, portanto, um assunto a ser levado em conta e que precisa tanto de clareza quanto de padronização. No caso das tecnologias adaptativas, a abrangência dos termos adaptatividade e adaptativo em diferentes áreas expõe a necessidade de alguns termos auxiliares para descrever e distinguir as pesquisas tecnológicas dos trabalhos em outras áreas. No entanto, o uso indiscriminado, descuidado ou incompleto de ditos termos pode gerar não só incongruências teóricas que podem ser espalhadas, mas também pode minguar a visibilidade e referências que o trabalho desenvolvido na área poderia ter.

É claro que não existe um único termo para descrever a funcionalidade ou comportamento de um dispositivo computacional, no entanto, é muito importante utilizar não só a terminologia adequada, mas também expressar a totalidade das características que dita entidade possui. Assim, por exemplo, um dispositivo adaptativo pode ser descrito como uma entidade dinâmica e autonomamente auto-reconfigurável ou auto-reajustável. Concerne ao critério de cada pesquisador utilizar o termo ou conjunto de termos que melhor cumpra tanto com pertencer ao vocabulário na área de conhecimento particular na qual é desenvolvida quanto com a abrangência da pesquisa e sua correta caracterização.

Da mesma maneira esta análise de terminologia permite-nos expor a importância do conhecimento da terminologia no campo de desenvolvimento da pesquisa, pois é dito conhecimento que permitirá o registro de uso de cada um desses termos para descrever algum tipo de tecnologia, e as mudanças que esses usos podem sofrer com o passar do tempo.

Uma outra conclusão é a importância do desenvolvimento de um glossário de termos relacionados à área de pesquisa. Assim mesmo, o registro da correlação entre alguns termos acunhados em algumas áreas de pesquisa permite desenvolver ferramentas que facilitam o levantamento de informações e seleção da terminologia. Esses instrumentos permitem dirigir os esforços dos pesquisadores na busca de informações, especialmente no levantamento de trabalhos relacionados, facilitando o filtro dos resultados em bases de dados de pesquisas, tarefa que normalmente se caracteriza por ser demorada.

Finalmente, a importância da padronização de definições para elaborar análise ao respeito das pesquisas desenvolvidas em tecnologias adaptativas: pesquisas interdisciplinares e multidisciplinares, tendências, novas áreas de aplicação, instituições fazendo pesquisa, novos jornais, novos eventos, interesse da indústria em aplicações de adaptatividade, entre outros. Todos elementos que auxiliam os pesquisadores em todos os níveis, tanto os iniciantes para conhecer a abrangência da adaptatividade quanto os experimentados para propor novos trabalhos que procurem satisfazer as necessidades tanto acadêmicas como de indústria e se mantenham alinhados com os avanços em outras áreas.

REFERÊNCIAS

[1] J. J. Neto, "Um Glosário sobre Adaptatividade", *Memórias do WTA 2009 - Terceiro Workshop de Tecnologias Adaptativas*, Laboratório de Linguagens e Técnicas Adaptativas, Departamento de Engenharia de Computação e Sistemas Digitais Escola Politécnica da Universidade de São Paulo, São Paulo, Brasil, 2009.

[2] D. O'neil, "Human Biological Adaptability – An Introduction to Human Responses to Common Environmental Stresses", *Behavioral Sciences Department, Palomar College, San Marcos, California*. Accessible from: http://anthro.palomar.edu/adapt/

[3] C. Brooks, "Adaptation is key in human evolution", *Online Stanford Report, January 16*, Stanford University, Stanford, California, 2009. Accessible from: http://news.stanford.edu/news/2009/january21/evoladap-012109.htm

[4] R. Oppermann, R. Rasher, "Adaptability and adaptivity in learning systems.", *Knowledge Transfer*, Vol. 2, p. 173-179, 1997.

[5] New England Complex Systems Institute. "Concepts: Adaptive". Acessible from: http://www.necsi.edu/guide/concepts/adaptive.html

[6] L. Braun et al.,"Adaptive Runtime System with Intelligent Allocation of Dynamically Reconfigurable Function Model and Optimized Interface Topologies", In M. Platzner et al. (eds.), *Dynamically Reconfigurable Systems*, Springer Science+ Business Media B.V. 2010.

[7] K. Compton and S. Hauck, "An introduction to reconfigurable computing". *IEEE Computer, 2000*.

[8] M. Persson. "Adaptive Middleware for Self-Configurable Embedded Real-Time Systems: Experiences from the DySCAS Project and Remaining Challenges." (2009).

[9] U. A. Acar, "Self-Adjusting Computing", PhD Thesis, *School of Computer Science, Carnegie Mellon University, Pittsburgh, Peennsylvania*, 2005.
[10] Personalize [Def. 2] (n.d.). In *Merriam-Webster: Dictionary and Thesaurus 2015*. Retrieved October, 2015, from http://www.merriam-webster.com/dictionary/personalize
[11] Customize [Def. 1] (n.d.). In *Merriam-Webster: Dictionary and Thesaurus 2015*. Retrieved October, 2015, from http://www.merriam-webster.com/dictionary/customize
[12] Adaptive[Def. 1] (n.d.). In *Merriam-Webster: Dictionary and Thesaurus 2015*. Retrieved October, 2015, from http://www.merriam-webster.com/dictionary/adaptive
[13] Adaptable[Def. 1] (n.d.). In *Merriam-Webster: Dictionary and Thesaurus 2015*. Retrieved October, 2015, from http://www.merriam-webster.com/dictionary/adaptable
[14] Adapt [Def. 1 and 2] (n.d.). In *Merriam-Webster: Dictionary and Thesaurus 2015*. Retrieved October, 2015, from http://www.merriam-webster.com/dictionary/adaptable
[15] Adaptability [Def. 1] (n.d.). In *Merriam-Webster: Dictionary and Thesaurus 2015*. Retrieved October, 2015, from http://www.merriam-webster.com/dictionary/adaptability
[16] Adapt [Def. 1] (n.d.). In *Collins English Dictionary 2015*. Retrieved October, 2015, from http://www.collinsdictionary.com/dictionary/english/adapt
[17] Adaptability [Def. 1] (n.d.). In *Collins English Dictionary 2015*. Retrieved October, 2015, from http://www.collinsdictionary.com/dictionary/english/adaptability
[18] Personalize [Def. 1] (n.d.). In *Collins English Dictionary 2015*. Retrieved October, 2015, from http://www.collinsdictionary.com/dictionary/english/personalize
[19] Customize [Def. 1] (n.d.). In *Collins English Dictionary 2015*. Retrieved October, 2015, from http://www.collinsdictionary.com/dictionary/english/customize
[20] Adaptivity [Def. 1] (n.d.). In *Collins English Dictionary 2015*. Retrieved October, 2015, from http://www.collinsdictionary.com/dictionary/english/adaptivity
[21] Personalize [Def. 2] (n.d.). *Longman Dictionary of Contemporary English, 2015*. Retrieved October, 2015, from http://www.ldoceonline.com/dictionary/personalize
[22] Adapt [Def. 2] (n.d.). *Longman Dictionary of Contemporary English, 2015*. Retrieved October, 2015, from http://www.ldoceonline.com/dictionary/adapt
[23] Personalize [Def. 1] (n.d.). Cambridge Free English Dictionary and Thesaurus, 2015. Retrieved October, 2015, from http://dictionary.cambridge.org/us/dictionary/english/personalize
[24] Customize [Def. 1] (n.d.). Cambridge Free English Dictionary and Thesaurus, 2015. Retrieved October, 2015, from http://dictionary.cambridge.org/us/dictionary/english/customize
[25] Personalizar [Def. 4] (n.d.). Dicionário Aurélio da Língua Portuguesa, versão online. Retrieved October, 2015, from http://dicionariodoaurelio.com/personalizar
[26] Adaptar [Def. 2] (n.d.). Dicionário Aurélio da Língua Portuguesa, versão online. Retrieved October, 2015, from http://dicionariodoaurelio.com/adaptar
[27] Adaptabilidade [Def. 2] (n.d.). Dicionário Aurélio da Língua Portuguesa, versão online. Retrieved October, 2015, from http://dicionariodoaurelio.com/adaptabilidade
[28] Adaptativo [Def. 2] (n.d.). Dicionário Aurélio da Língua Portuguesa, versão online. Retrieved October, 2015, from http://dicionariodoaurelio.com/adaptativo

Rosalia Edith Caya Carhuanina nasceu na cidade de Lima, Perú. Recebeu o grau de *Bachiller en Ciencias e Ingeniería con Mención en Ingeniería Informática* e o Título Profissional de *Ingeniera de Computación y Sistemas* da Pontificia Universidad Católica del Perú (PUCP) em Lima, Perú em 2009. 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 Linguagens e Técnicas Adaptativas. Suas principais áreas de pesquisas são: Inteligência Artificial, Processamento de Linguagem Natural, Interação Humano-Máquina 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.

Classificação de Espécies de Peixe usando Inferência Gramatical no Reconhecimento de Padrões em Problemas de Visão Computacional

M. R. Borth[1], L. C. Ribas[2], H. Pistori[3], W. N. Gonçalves[4], A. A. C. Junior[5]

***Abstract* — Este artigo apresenta uma nova abordagem para representar uma imagem como uma sentença gerada a partir de um agrupamento baseado em estrutura hierárquica, de modo que técnicas de inferência gramatical possam ser usadas em problemas de visão computacional. Para a classificação de imagens, este trabalho inicialmente detecta e descreve pontos de interesse usando o algoritmo SIFT; em seguida, constrói o alfabeto com base no conjunto de descritores dos pontos de interesse, rotulando cada um deles; na sequência, realiza a geração de uma sentença por agrupamento hierárquico; e, por fim, realiza a aprendizagem da gramática para a classificação de imagens utilizando o algoritmo de inferência gramatical *k-testable*. Dois experimentos foram realizados utilizando dois novos bancos de imagens de peixes, criados pelos autores deste trabalho, a fim de explorar parâmetros da abordagem e ter um primeiro resultado sobre o desempenho da técnica. Os resultados são encorajadores para continuar e explorar novos caminhos na área de reconhecimento sintático de padrões.**

***Keywords* — Geração de sentença; Reconhecimento sintático de padrões; Inferência gramatical; Agrupamento hierárquico; Classificação de imagens.**

I. INTRODUÇÃO

Informações que tratam sobre reconhecimento, classificação, distribuição, quantificação e migração de espécie de peixes muitas vezes são críticas para ecologistas, estudiosos do meio ambiente, órgãos governamentais e empresas de pescado [1]. Nesse contexto, a criação de softwares para monitorar o ambiente de onde os peixes estão inseridos possibilita adotar alternativas ou ações para atuarem, desde o controle preventivo até a criação de novas soluções para controlar o crescimento saudável de peixes. Além disso, sistemas de classificação de espécies de peixes podem ajudar biólogos a resolver questões de disponibilidade de alimentos, relação entre predador e sua presa [1, 2, 3], inclusão de estudos ecológicos e ambientais das comunidades de peixes [4], projeto e colocação de escadas de peixes em represas para geração de energia hidrelétrica [5], alimentação estratégica pelos piscicultores [6] e avaliação de ações para gestão das pescas [7]. Em geral, alguns diagnósticos são realizados por peritos, onde eles usam a própria visão para detectar determinadas características ou doenças. Entretanto essas pessoas são escassas, especialmente em áreas rurais. Cada espécie de peixe possui características específicas, tais como: tamanho de nadadeiras, padrões de cores, tamanho do espécime, etc. Na classificação de peixes, por exemplo, existem 47 características diferentes por espécie para que se consiga um bom nível de exatidão na identificação [8].

Diferente do que acontece no dia a dia do trabalho que envolve a classificação de peixes, o qual se baseia em procedimentos manuais de alto custo e baixa eficiência, este trabalho apresenta uma abordagem para a classificação automática de espécies de peixes que pode ser usada nas atividades relacionadas anteriormente. Este trabalho permite criar uma solução automatizada que ajude o trabalho manual executado por seres humanos, mediante o uso de um sistema computacional baseado em técnicas de visão computacional.

Alguns trabalhos propõem o desenvolvimento de um método para classificação de formas geométricas básicas por meio de autômatos adaptativos [9, 10]. Esses trabalhos utilizam a abordagem sintática de reconhecimento de padrões, apresentando algumas vantagens sobre práticas tradicionais permitindo realizar uma identificação de propriedades estruturais das formas contidas na imagem. A característica adaptativa aperfeiçoa o processo de reconhecimento ao considerar pequenas diferenças entre bordas, geradas por distorções residuais nas fases de segmentação e pré-processamento. Entretanto, a abordagem se restringe apenas a atributos relacionados a borda do objeto, diferentemente deste trabalho que pode utilizar informações de toda a parte da imagem. Ainda assim, durante o processo de aprendizagem da gramática, a geração dos autômatos utilizados para validar as sentenças pode ser realizada a partir do uso de técnicas e conceitos de adaptatividade.

Este trabalho apresenta uma nova forma de ordenação de pontos de interesse detectados em uma imagem a fim de produzir uma sentença. A sentença, também conhecida por *string*, palavra ou cadeia de caracteres, é a representação simplificada de uma sequência finita de palavras visuais ordenadas, por exemplo, "ABACDECBCADA". A sentença é construída com base no tamanho de um alfabeto que é definido usando o algoritmo k-médias (*k-means*), como no

[1] Doutorando em Ciências Ambientais e Sustentabilidade Agropecuária pela Universidade Católica Dom Bosco (UCDB). Professor do Instituto Federal do Paraná (IFPR) – Campus Umuarama. E-mail: marceloborth@gmail.com

[2] Mestrando em Ciência da Computação e Matemática Computacional pela Universidade de São Paulo. E-mail: lucascorreiaribas@gmail.com

[3] Doutor em Engenharia Elétrica pela Universidade de São Paulo. Professor da Universidade Católica Dom Bosco (UCDB). E-mail: pistori@ucdb.br

[4] Doutor em Física pela Universidade de São Paulo. Professor da Universidade Federal de Mato Grosso do Sul (UFMS) – Campus Ponta Porã. E-mail: wnunesgoncalves@gmail.com

[5] Doutor em Engenharia Elétrica pela Universidade de São Paulo. Professor da Universidade Federal de Mato Grosso do Sul (UFMS) – Campus Ponta Porã. E-mail: amaury.ufms@gmail.com

histograma de palavras visuais [11, 12]. O alfabeto, também conhecido como dicionário, é um conjunto finito de símbolos, números ou caracteres. Para gerar a sentença, a ordenação é baseada no agrupamento hierárquico do rótulo de cada ponto de interesse.

O trabalho propõe a representação de uma imagem como sentença, gerada a partir de agrupamento baseado em estrutura hierárquica. Para a classificação de imagens, inicialmente são detectadas e descritos pontos de interesse usando o algoritmo *Scale Invariant Feature Transform* – SIFT [13]. Esse algoritmo permite identificar e descrever pontos de interesse relevantes de imagens para realizar o reconhecimento de padrões. Em seguida, é construído o alfabeto com base no conjunto de descritores dos pontos de interesse, rotulando cada um deles. O alfabeto é definido usando o algoritmo k-médias (*k-means*), como no histograma de palavras visuais (*Bag of Visual Words*). Na sequência, uma sentença para cada imagem é gerada. A sentença possuirá tamanho n, tal que n é a quantidade de pontos de interesse detectados na imagem. A ordenação gerada é baseada na técnica de agrupamento hierárquico, unindo pontos de interesse próximos a partir da posição espacial (x, y) até formar uma árvore binária completa. Por fim, é realizada a aprendizagem da gramática para a classificação de imagens utilizando o algoritmo de inferência gramatical *k-testable* [14].

Dois experimentos foram realizados usando imagens de peixes, a fim de analisar a estratégia de ordenação dos pontos de interesse e o impacto do tamanho do alfabeto sobre o desempenho da classificação. Para esses experimentos, o algoritmo de detecção de pontos de interesse SIFT e o algoritmo de inferência gramatical *k-testable* foram utilizados. Entretanto, a técnica pode ser aplicada usando qualquer algoritmo de detecção de pontos de interesse e de inferência gramatical disponíveis.

A principal contribuição deste trabalho está na ordem empregada para a geração das sentenças, pois é baseada na técnica de agrupamento hierárquico, unindo pontos de interesse próximos entre si a partir da posição espacial (x, y) que possuem menor distância, até formar a estrutura de uma árvore binária completa. Também, contribui em testes com uma larga faixa de tamanho de alfabeto, em que o limite é a capacidade de memória do computador. Outra contribuição é a publicação de dois bancos de imagens de peixes. O aquario10e[7] constituído de 596 imagens de peixes e o aquarioSeg10e[8] com 100 imagens de peixes segmentadas manualmente. Ambos os bancos de imagens possuem imagens de 10 espécies de peixes e se destacam por conter apenas imagens capturadas por celulares, visto que os bancos de imagens atuais fornecem em sua maioria imagens de câmeras fotográficas e/ou em ambientes controlados.

A seguir é apresentada uma revisão teórica dos principais conceitos utilizados neste trabalho. Na sequência, são apresentados os trabalhos relacionados, discutindo os problemas e limitações das abordagens atuais. Posteriormente, é apresentada a proposta deste trabalho, os experimentos realizados e discussões sobre os resultados. Por fim, são apresentadas as considerações finais e futuros direcionamentos.

II. REVISÃO TEÓRICA

Nesta seção, são apresentados os principais conceitos utilizados para a realização deste trabalho.

A. SIFT – Scale Invariant Feature Transform

A extração e descrição de características em imagens têm sido extensivamente empregada na área de reconhecimento de imagens. Atualmente, o *Scale Invariant Feature Transform* - SIFT e o *Speeded Up Robust Features* – SURF [15] são os dois métodos mais populares. Esses métodos extraem da imagem uma coleção de vetores de características locais, chamados de pontos de interesse. Cada ponto de interesse é composto de uma parte pequena da imagem e a quantidade de pontos detectados pode variar de uma imagem para outra. Essas técnicas buscam ser invariantes a rotação, escala da imagem e mudança de iluminação. Esses extratores são extremamente relevantes para tarefas de reconhecimento de padrões. Sua implementação usa a função de diferença de Gaussianas. Ao serem identificados, cada ponto de interesse é representado por quatro elementos:

$$p_i = \{(x_i, y_i), \sigma_i, \theta_i, \varphi_i\}$$

onde,

- (x_i, y_i), localização espacial na imagem;
- σ_i, escala em que ele foi detectado;
- θ_i, orientação predominante do gradiente; e,
- $\varphi_i \in R^{128}$, descrição do ponto de interesse, representado por um vetor de características contendo 128 valores que descrevem a região ao redor do ponto.

B. Histograma de palavras visuais

O histograma de palavras visuais ou, simplesmente, B*ag of Visual Words* - BOVW, é um vetor de atributos extraídos de uma imagem ou conjunto de imagens. Para a extração desses atributos é necessário o uso de um algoritmo auxiliar, como o SIFT. Assim, o BOVW pode usar os descritores extraídos pelo SIFT para construir o alfabeto e associá-los aos pontos de interesse. Após a construção do alfabeto, os pontos de interesse podem ser referenciados como palavras visuais, pois cada um deles terá uma palavra visual associada a ele que é baseada na sua descrição.

O histograma de palavras visuais é uma técnica bastante utilizada em tarefas de classificação. Basicamente, pode acontecer dela contar a quantidade de ocorrências das palavras visuais de uma determinada sentença e fazer a distribuição de frequência dessas palavras, gerando um histograma. Assim, para determinar a classe de uma nova imagem cria-se um

[7]Disponível em: https://www.dropbox.com/sh/crq14nug14jcuc4/AAAy-1ul8YBQ5AtXA7M1fz1wa

[8]Disponível em: https://www.dropbox.com/sh/50u4z5o197bstqk/AACDf64uxVOFKTyti7smoIG-a

histograma de palavras visuais para uma nova imagem, o qual é comparado com histogramas das imagens de treinamento. A classe determinada será a do histograma mais semelhante com as imagens de treinamento, pois imagens de uma mesma classe possui maior ocorrência de certas palavras visuais.

Apesar dos avanços recentes e dos resultados promissores, o poder descritivo dessa técnica é limitado, pois descarta informações espaciais e estruturais das palavras visuais na imagem. Essas informações podem ser características importantes em tarefas de classificação de imagens, pois quando apenas são contadas as ocorrências de uma palavra visual em uma imagem, não é considerada sua localização espacial na imagem e nem seu posicionamento com as demais palavras visuais. Logo, para algumas técnicas de classificação de imagens é importante saber o posicionamento dos pontos de interesse na imagem e manter a relação espacial dos pontos na imagem como, por exemplo, no reconhecimento sintático de padrões. Essa técnica, por sua vez, pode gerar diferentes ordens na formação de sentença a partir dos pontos de interesse detectados, o que pode influenciar no resultado da classificação.

C. K-TESTABLE

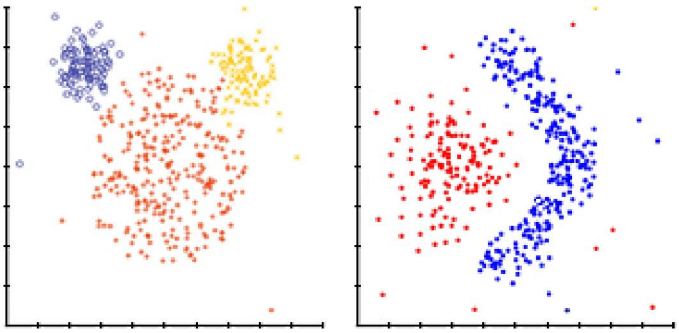
O *K-Testable* é uma técnica que tenta encontrar um autômato para reconhecer uma linguagem a partir de um tamanho de memória k_t. Uma linguagem *K-Testable* é uma subclasse das linguagens regulares que realiza a análise de qualquer sequência usando memória fixa de tamanho k. Esse algoritmo é capaz de inferir linguagens em tempo polinomial por meio de prefixos, sufixos e partes de sentenças que ocorrem nos dados de treinamento [16]. A principal característica é que cada caractere é dependente apenas dos k_t-1 caracteres anteriores.

D. AGRUPAMENTO HIERÁRQUICO

O agrupamento (*clustering*) é o estudo de métodos para agrupar objetos de acordo com características similares entre si, como a distância entre dois pontos. Essa é uma tarefa que separa grupos similares, com o objetivo de melhor discriminar objetos pertencentes a classes diferentes. Por exemplo, dada uma representação de N objetos, encontram-se k grupos baseando-se em uma medida de similaridade, como a distância euclidiana, tal que objetos presentes em um mesmo grupo são considerados semelhantes e objetos de grupos diferentes são considerados distintos. A similaridade indica o nível de proximidade entre dois objetos em um conjunto de dados. Existem várias aplicações que utilizam técnicas de agrupamento, por exemplo, segmentação de imagens [17, 18, 19], agrupamento de documentos [20, 21, 22], estudo de dados de genoma [23], dentre outras.

Uma técnica de agrupamento hierárquico organiza os dados em uma hierarquia de grupos. Essa solução pode ter maior vantagem em relação às abordagens planas, uma vez que divide os pontos de interesse em vários níveis de especificidade e diferentes granularidades [24]. Nesse caso, a técnica pode iniciar a tarefa considerando cada ponto de interesse como um grupo distinto e, prosseguir sucessivamente combinando pontos de interesse mais similares, até que todos os pontos sejam alocados a um único grupo. A árvore hierárquica gerada não é apenas um conjunto de grupos, mas uma hierarquia com vários níveis, formando uma árvore binária, onde os grupos de um nível são unidos com grupos do próximo nível. Isso possibilita decidir o nível de agrupamento mais adequado para cada tipo de aplicação ou como será feita a separação dos grupos baseado no comportamento da árvore hierárquica. Os vários grupos podem ser úteis em aplicações porque os pontos de interesse de cada grupo podem representar graus diferentes de similaridade.

O processo de agrupamento possui 5 fases [25]: pré-processamento, seleção da medida para o cálculo de similaridade, agrupamento dos dados, avaliação dos resultados e interpretação dos resultados. Após o cálculo de similaridade entre os objetos, na fase de agrupamento é aplicado um algoritmo de agrupamento dos objetos como, o k-médias, onde k é o número de grupos (*clusters*). Basicamente, o centro de cada grupo é definido como a média de todos os objetos que pertencem ao grupo. Assim, ao aplicar o método k-médias os objetos são separados em vários grupos baseado no número *k*, conforme pode ser visto na Figura 1 (a). Nessa figura, k é 3, logo são formados 3 grupos diferentes. Por outro lado, existe o agrupamento hierárquico que permite maior flexibilidade em um agrupamento, conforme ilustrado na Figura 1 (b). É possível ver a diferença entre ambas as técnicas, tal que, pelo agrupamento hierárquico é possível agrupar informações que pelo k-médias não seriam possíveis, pois o k-médias impõe a restrição de ter grupos esféricos. Detalhes e outros trabalhos sobre k-médias podem ser vistos em [26].



(a) Agrupamento por *k-médias* (b) Agrupamento hierárquico

Figura 1. (a) Exemplo de agrupamento pelo algoritmo k-médias e (b) agrupamento hierárquico.

III. TRABALHOS CORRELATOS

Na visão computacional é comum usar o BOVW para classificação de imagens. Com essa técnica, cada imagem é tratada similarmente como um documento na classificação de textos, em que se obtém a frequência das palavras em um documento, gerando um histograma de palavras baseado na sua repetição. Para a classificação de imagens é necessário seguir algumas etapas: extrair suas características; descrevê-las; gerar o histograma de palavras visuais; aplicar um método de agrupamento nos descritores dos pontos de interesse detectados, que geralmente é o k-médias; e, por fim, realizar a classificação usando um algoritmo como, J48 (C4.5), *k-Nearest Neighbors* (KNN) e *Support Vectors Machine* –

Sequential Minimal Optimization (SMO). Embora grandes avanços tenham sido publicados nos últimos anos, uma das limitações dos histogramas é desconsiderar totalmente a relação espacial/estrutural dos pontos de interesse na imagem, o que acarreta na perda de informação da localização dos pontos de interesse.

Para superar a limitação do BOVW na classificação de imagens e permitir o uso de informação espacial dos pontos de interesse, alguns trabalhos foram propostos [27, 28, 29, 30, 31, 32]. Em geral, após a identificação dos pontos de interesse são geradas as palavras visuais com base em um tamanho de alfabeto utilizando o BOVW. Assim, é comum na área de reconhecimento sintático de padrões, criar uma sentença com uma determinada ordem a partir das palavras visuais extraídas da imagem. Dentre os trabalhos citados [27] propõem o uso de inferência gramatical, trazendo resultados interessantes no problema de classificação de imagens. O principal problema no uso de gramáticas em imagens está na representação da imagem a partir de uma sentença que represente suas características relevantes sem ignorar a relação espacial das palavras visuais na imagem [27]. Nesse trabalho, os autores apresentam uma nova estratégia para representar uma imagem através de uma sentença, de modo que algoritmos sejam usados no reconhecimento de padrões em imagens. A proposta explora o padrão textual da sentença e estratégias que visam preservar a informação espacial dos pontos de interesse e testa o tamanho do alfabeto e o tipo de ordenação dos pontos de interesse para a geração da gramática.

As abordagens de ordenação de palavras visuais para a formação da sentença propostas por [27] possuem limitações quanto à sua ordenação. A ordem randômica (cria a sentença obtendo as palavras visuais de forma aleatória), radial (inicia pelo ponto de interesse central da imagem e, segue sucessivamente concatenando os pontos mais próximos usando a distância euclidiana), ordem de leitura (concatena os pontos de interesse iniciando de cima para baixo e da esquerda para a direita), etc., mantém a estrutura espacial da imagem, porém não mantém pontos de interesse mais próximos um do outro juntos na sentença, o que pode prejudicar o processo de classificação. Logo, se as características da imagem não estiverem na ordem correta na sentença, o desempenho na classificação será prejudicado. Nas 4 imagens de peixe da Figura 2, é possível perceber que a orientação espacial se mantém, porém a sentença gerada para o mesmo peixe não é invariante a rotação, pois a ordem das palavras visuais é diferente para cada uma das quatro sentenças geradas. Logo, a taxa de classificação correta pode ser prejudicada se aplicada uma abordagem que mantém a orientação espacial e não é invariante a rotação para uma mesma imagem.

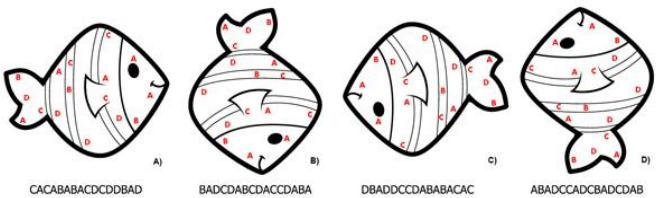


Figura 2. Quatro imagens de peixe com variação de 90° na rotação de uma para outra. Abaixo a sentença gerada de cada imagem a partir da ordenação dos pontos de interesse pela ordem de leitura ocidental (iniciando da parte superior e concatenando da esquerda para a direta).

Outro problema é destacado na Figura 3, pois quando realizada a ordenação dos pontos de interesse pelo método radial a geração da sentença é invariante a rotação mas não a translação. Mesmo sendo invariante a rotação, não há qualquer relação de um ponto de interesse anterior ou posterior na sentença gerada, o que pode prejudicar a classificação. Por exemplo, a palavra visual P_x não tem qualquer relação com a palavra visual P_x-*1* ou P_x+*1* na sentença. Dessa forma, uma palavra visual do olho do peixe pode ser a próxima de uma que representa a cauda e, assim, sucessivamente, misturando várias palavras visuais que podem ser totalmente distintas e sem proximidade uma da outra na imagem.

Figura 3. Distribuição das palavras visuais e marcação radial em uma imagem de peixe. Sentença resultante pela ordenação radial: HFFMGAABNMBKJ.

No processamento sintático de padrões, a ordem das palavras visuais pode caracterizar a importância ou o quanto cada palavra visual está conectada ou tem relação uma com a outra. Nesse contexto, a ordem das palavras visuais em uma sentença pode ser levada em consideração para identificar características semelhantes entre imagens, situação que não ocorre utilizando o *Bag of Words*, pois a informação estrutural é perdida ou, até mesmo, pelas abordagens mais recentes citadas nos trabalhos relacionados, pois não são invariantes a rotação ou não mantém um agrupamento dos pontos de interesse semelhantes.

No processo de formação da sentença, entendemos que é preciso manter a relação espacial entre as palavras visuais encontradas na imagem e ser invariante a rotação e translação, pois é importante manter uma estrutura organizacional das características de uma imagem. Logo, é preciso ter uma alternativa para organizar a informação estrutural da imagem como, por exemplo, a taxonomia. Essa é uma classificação de entidades de informação em forma de hierarquia de acordo com relacionamentos estabelecidos com entidades do mundo real que a representam [33]. As taxonomias possuem relacionamentos de generalização e especialização, e podem ser utilizadas para classificar informação de forma hierárquica. É em cima de toda a problemática apresentada nessa seção que a abordagem proposta tem sua maior contribuição, conforme detalhada a seguir.

IV. ABORDAGEM PROPOSTA

Neste trabalho é proposta uma nova forma de ordenação de palavras visuais em imagens baseada na técnica de agrupamento hierárquico, com objetivo de manter a estrutura das palavras visuais em uma sentença. A ideia principal da proposta é impor uma ordem para os n pontos de interesse e produzir uma sequência de tamanho n. O classificador pode ser dividido em 5 etapas: 1) extração de pontos de interesse; 2) construção do vocabulário; 3) rotulação de pontos de interesse; 4) geração da sentença; e, 5) aprendizagem da gramática.

A. Extração de pontos de interesse

Primeiramente, são detectados os pontos de interesse nas imagens e gerado um vetor de descrição para cada um deles usando o SIFT.

B. Construção do alfabeto

Este trabalho apresenta duas abordagens para a construção do alfabeto, não supervisionada e supervisionada, conforme usado por [34]. Na abordagem não-supervisionada, o conjunto de descritores é composto por todos os descritores de todas as imagens. Assim, o alfabeto é construído desconsiderando a classe de cada ponto de interesse, como se todas as classes fossem a mesma, deixando o descritor responsável para determinar a que grupo do alfabeto cada ponto de interesse pertencerá. A construção do alfabeto é realizada com base no conjunto de descritores D dos pontos de interesse, dado por:

$$D = [\, \varphi_i^j \,], 1 \leq i \leq M_j, 1 \leq j \leq N$$

onde, N é o número de imagens de treinamento e M_j é o número de pontos extraídos da imagem j.

O conjunto de descritores são agrupados usando o algoritmo k-médias e o conjunto de k centroides C é obtido a partir de:

$$C = k\text{-médias}\ (D)$$

O conjunto de centroides C é descrito como o alfabeto aprendido para as imagens de treinamento. Cada centroide possui a mesma dimensão dos descritores dos pontos de interesse, em que $C \in R^{128}$.

Na abordagem supervisionada, por sua vez, o alfabeto é construído separadamente para cada classe. Logo, a quantidade distinta de palavras visuais possíveis P_V de um alfabeto para cada classe é dada por:

$$P_V = T / n_c$$

onde, T é o tamanho do alfabeto e n_c é o número de classes. Os centroides C_p, são obtidos usando o algoritmo k-médias, aplicado apenas em um conjunto D_p contendo somente os descritores das imagens pertencentes a classe p:

$$D_p = [\, \varphi_i^j \,], \text{ se } j \text{ pertence à classe } p$$

$$C_p = \text{k-médias}\ (D_p)$$

Considerando n_c como o número total de classes, n_c conjuntos de centroides serão criados. Assim, para a construção do alfabeto supervisionado, os n_c conjuntos de centroides são concatenados. A vantagem da construção do alfabeto supervisionado em relação ao não-supervisionado é que o aprendizado das sentenças será obtido para cada classe separadamente, o que não acontece na construção não-supervisionada. Logo, com a abordagem supervisionada, cada classe pode gerar centroides mais característicos baseando-se nos seus próprios descritores.

C. Rotulação dos pontos de interesse

A rotulação de cada ponto de interesse das imagens de treinamento é feita na sequência. Dados cada ponto de interesse p_i^j para a imagem j, cada um deles é rotulado com o índice da palavra mais próxima do alfabeto, ou seja, o centroide mais próximo conforme apresentado abaixo:

$$r_i^j = \arg\min_{l=1}^{k} |\varphi_i^j, C_l|$$

onde r_i^j corresponde ao rótulo do ponto de interesse i da imagem j e $|\, \varphi_i^j, C_l \,|$ é a distância euclidiana.

Após essa etapa, cada ponto de interesse possuirá uma palavra visual, ou rótulo, de forma que pontos de interesse com descritores semelhantes possuirão as mesmas palavras visuais. Um exemplo pode ser visto na Figura 3, onde as palavras visuais são indicadas por letras do alfabeto

D. Geração da sentença

A ordenação baseada em agrupamento hierárquico foi usada para gerar a sentença a partir das palavras visuais da imagem. Essa é uma técnica invariante a rotação e translação. O algoritmo de agrupamento hierárquico segue 2 passos:

1. Encontrar a similaridade entre cada par de pontos de interesse no conjunto de dados: nessa etapa, é calculada a distância entre ponto de interesse usando a distância Euclidiana; e,
2. Agrupar os objetos em uma árvore hierárquica binária: nessa etapa, são relacionados pares de pontos de interesse que estão em proximidade usando o algoritmo *linkage*, implementado no Matlab. Uma vez que a proximidade entre os objetos no conjunto de dados já foi calculada, é possível determinar como os objetos no conjunto de dados devem ser agrupados. Essa função usa a informação de distância gerada na etapa anterior a fim de determinar a proximidade dos objetos entre si. Assim, o *linkage* pega as distâncias e relaciona pares de objetos que estejam juntos em clusters (dois a dois). Em seguida, relaciona esses grupos recém-formados para si e para outros objetos para criar grupos maiores até que todos os objetos do conjunto de dados estejam relacionados entre si, formando uma árvore hierárquica.

Nessa ordenação, as palavras são extraídas da imagem de acordo com a distância euclidiana da posição espacial de um ponto a outro, onde as palavras visuais mais próximas entre si são agrupadas para formar a sentença. Também é usada a ordenação radial como forma de comparação, onde a sentença é gerada de acordo com o valor da distância euclidiana de cada palavra visual ao centro da imagem. Assim, as palavras visuais mais próximas ao centro são adicionadas primeiro na sentença, repetindo, sucessivamente, até a palavra visual mais distante do centro.

A Figura 4 ilustra pontos de interesse extraídos de uma imagem e a árvore hierárquica gerada baseada na construção da sentença desta abordagem. As palavras visuais agrupadas mais ao topo da hierarquia representa maior distância em relação as demais palavras visuais que possuem um agrupamento mais abaixo da hierarquia. Quanto mais alto for o agrupamento da palavra visual maior dissimilaridade terá com os grupos mais baixo, conforme ilustrado à direita, a escala de similaridade. Logo, os nós próximos das folhas representam palavras visuais que possuem maior similaridade entre si e, consequentemente, ficarão juntas na geração da sentença.

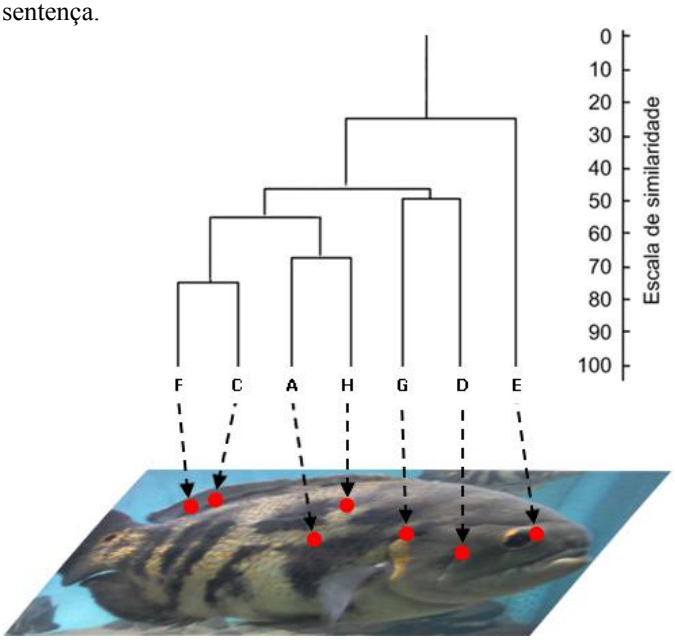


Figura 4. Ordenação dos pontos de interesse na imagem do peixe a partir do agrupamento hierárquico representado por um dendograma. A sentença gerada para essa imagem seria "F C A H G D E".

E. Aprendizagem da gramática

A última etapa da abordagem é inferir uma gramática, ou regras, para cada classe a partir das sentenças geradas, resultando um autômato para cada classe. Para a inferência, o algoritmo *K-Testable* é utilizado. O *K-Testable* é uma técnica que dado um tamanho de memória k_t, tenta-se encontrar um autômato para reconhecer uma linguagem representada por um número de cadeias passadas como parâmetro. Uma linguagem k-testável é uma subclasse de uma linguagem regular que encontra prefixos, sufixos e sub-cadeias nos dados de treinamento [27]. A principal característica é que cada caractere é dependente apenas dos k_t-1 caracteres anteriores e a análise de uma cadeia de caracteres pode ser feita usando uma memória de tamanho fixo k_t [27].

Nesse momento, as sentenças que representam as imagens de teste são validadas nos autômatos gerados. Para validar uma sentença é contado o número de erros para cada autômato. Os erros ocorrem quando há um caractere na sentença que não pertence a linguagem ou não existe a transição na linguagem de um autômato. O autômato que retornar o menor número de erros é a que pertence a sentença testada.

O tratamento de erros é um tópico muito importante na área de reconhecimento sintático, uma vez que pode conter erros ao validar uma sentença em uma gramática. Existem duas fases para o tratamento de erros: i) a detecção do erro e, ii) o tratamento ou recuperação do erro. Apenas detectar o erro não torna o analisador eficiente, uma vez que ele deve se recuperar do erro. Como os analisadores sintáticos analisam a gramática de linguagens, pode ser que símbolos esperados não sejam encontrados.

Um problema recorrente na área da teoria da computação é encontrar o equilíbrio entre expressividade e usabilidade na recuperação de erros, uma vez que máquinas de Turing, por exemplo, são formalismos muito expressivos, o que torna desconfortável o seu uso em problemas reais. Por outro lado, as máquinas de estados finitos são fáceis de usar, porém são mais restritas, o que também prejudica sua aplicação em problemas reais. Uma alternativa para isso é usar técnicas adaptativas [35, 36].

Os algoritmos adaptativos aumentam a expressividade de um formalismo sem afetar a usabilidade. Assim, é possível ser adaptativo sem modificar a sintaxe e a semântica original. Um algoritmo alternativo que poderia ser usado para esse tipo de problema na recuperação de erros é o AdapTree [37]. Ele é um algoritmo para indução de árvores de decisão que usa teoria de autômatos e enriquecido com tecnologia adaptativa. Umas das principais características deste algoritmo é a utilização conjunta de estratégias sintáticas e estatísticas.

V. EXPERIMENTOS

Dois experimentos foram realizados utilizando imagens de peixes de 10 espécies diferentes. O primeiro experimento traz imagens do banco de imagens aquario10e. Ao total são 596 imagens coloridas divididas em 10 classes. O segundo banco de imagens chamado aquarioSeg10e é composto de 10 imagens segmentadas (recortadas manualmente) escolhidas aleatoriamente de cada espécie do banco aquario10e, totalizando 100 imagens, 10 de cada espécie. A Figura 5 apresenta as espécies do banco aquario10e e a Figura 6 apresenta as espécies do banco aquarioSeg10e. As imagens variam umas das outras em tamanho e resolução, pois elas foram capturadas de diferentes celulares e posteriormente recortadas para deixar apenas o peixe na imagem. A distância da câmera ao peixe foi variável, de 20 centímetros a 1 metro, com iluminação natural do ambiente. Por isso, interferências como alteração na cor, iluminação desigual, reflexo, fundo diferente, sedimentos na água, plantas subaquáticas, etc., podem ser encontradas em algumas imagens. A principal diferença de um banco de imagens para outro é referente ao

primeiro banco ter o fundo em cada imagem, o que dificulta a aprendizagem para a classificação por conta de ruídos. Nesse caso, conseguimos analisar o quanto o fundo da imagem interfere na classificação da abordagem proposta. O nome e a quantidade de imagens de cada espécie podem ser obtidos na Tabela 1.

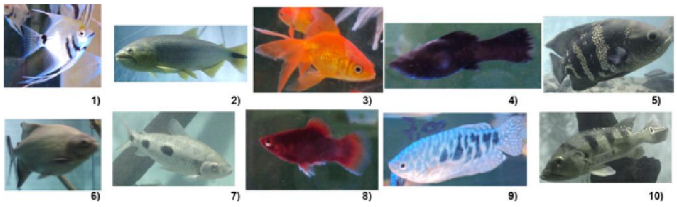


Figura 5. Imagens das espécies de peixes do banco de imagens aquario10e.

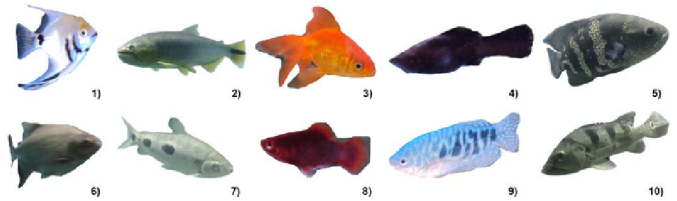


Figura 6. Imagens das espécies de peixes do banco de imagens aquarioSeg10e.

A construção do alfabeto foi realizada de duas formas, supervisionada e não-supervisionada com o tamanho do alfabeto variando entre 10 e 3000. O parâmetro do *K-Testable* utilizado nos experimentos foi $k_t = 2$, uma vez que usando valores maiores a geração da gramática não melhorou o resultado do classificador. Para a divisão do conjunto de dados de treinamento e teste foi usado o modelo de validação cruzada com 10 dobras. A Medida-F (*F-Measure*), também conhecida como *F-Score*, é adotada como parâmetro de comparação entre as abordagens testadas. Essa métrica é a média harmônica da precisão e revocação.

Espécie/Classe	**Total de imagens**
1. Acará bandeira (*Pterophyllum scalare*)	60
2. Dourado (*Salminus brasiliensis*)	80
3. Kinguio (*Carassius auratus*)	41
4. Molinésia Preta (*Poeclia Shenops*)	44
5. Oscar (*Astronotus ocellatus*)	101
6. Pacu (*Piaractus mesopotamicus*)	49
7. Piau 3 Pintas (*Leporinus friderici*)	75
8. Platy Sangue (*Xiphophorus maculatus*)	43
9. Tricogaster Azul (*Trichogaster trichopterus*)	41
10. Tucunaré (*Cichla piquiti*)	62

Tabela 1. Número de imagens para cada espécie de peixe do banco de imagens aquario10e, totalizando 596 imagens.

VI. RESULTADOS E DISCUSSÃO

A Tabela 2 apresenta os resultados dos experimentos para ambos os bancos de imagens, na forma supervisionada e não-supervisionada, com diferentes tamanhos de alfabeto e dois tipos de geração de sentença, radial e hierárquica. A construção do vocabulário supervisionado forneceu resultados bem superiores quando comparado com a construção do vocabulário não-supervisionado. Esse resultado demonstra que a obtenção de palavras individuais para cada categoria separadamente é importante no processo de descrição de imagens. A Medida-F mais alta obtida em todas as configurações foi para o banco de imagens aquarioSeg10e, na abordagem supervisionada, ordenação hierárquica e com tamanho do alfabeto de 2400.

A Figura 7 ilustra um gráfico baseado nos resultados desse banco de imagens. O gráfico compara os dois processos de construção do alfabeto, pela forma supervisionada e não-supervisionada, mostrando a Medida-F para cada tamanho do alfabeto. No gráfico podemos observar que a abordagem supervisionada é superior em praticamente todos os tamanhos de alfabetos testados.

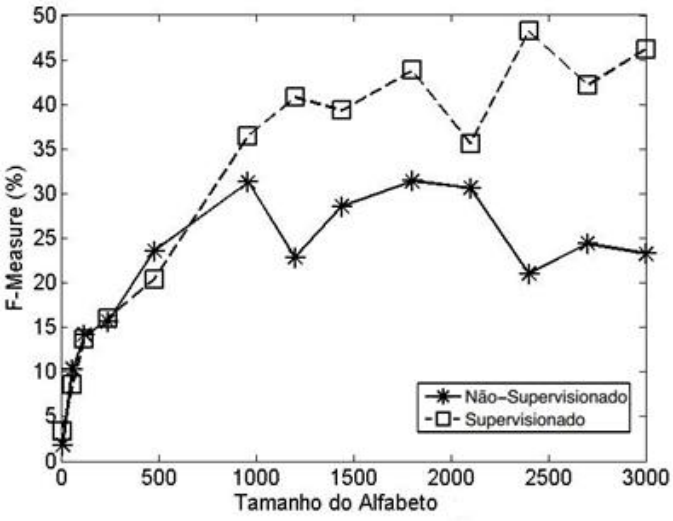


Figura 7. Gráfico comparativo das Medidas-F para a abordagem supervisionada e não supervisionada observando o tamanho de cada alfabeto com o banco de imagens aquarioSeg10e e ordenação hierárquica.

Em geral, percebe-se que o desempenho do classificador melhora conforme se aumenta o tamanho do alfabeto, pois o tamanho está diretamente relacionado ao poder descritivo das características dos pontos de interesse nas imagens. Logicamente, existe um limite para atingir sua descrição máxima que, nesse caso, foi encontrado no alfabeto de tamanho 2400, pois com um dicionário maior a classificação reduziu a Medida-F. As melhores Medidas-F, para ambos os bancos de imagem, foram obtidas usando a geração de sentença por agrupamento hierárquico e tamanho de dicionário de 2400.

O resultado mais satisfatório na classificação de espécie de peixes foi para o banco de imagens aquarioSeg10e, pois nesse caso, ao aplicar a extração de pontos de interesse na imagem, os pontos são encontrados somente na imagem do peixe, ao contrário do que ocorre com o banco de imagens aquario10e, pois além dos pontos de interesse detectados no peixe são

detectados inúmeros pontos no fundo da imagem, conforme ilustrado na Figura 8. Nesse caso, os pontos de interesse detectados no fundo da imagem são ruídos, reduzindo o desempenho do classificador. Em várias imagens, a maioria dos pontos de interesse são detectados no fundo da imagem e não no peixe.

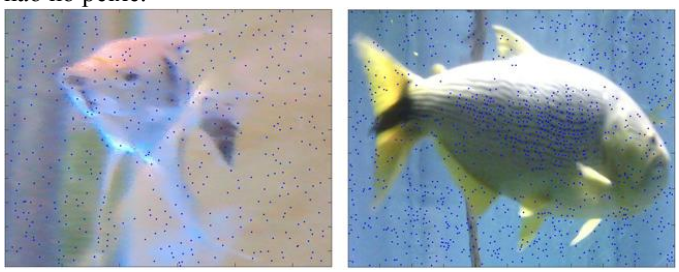


Figura 8. Exemplo de pontos de interesse detectados em imagens de duas espécies de peixe em que a maioria são ruídos

A Figura 9 apresenta a matriz de confusão para o melhor resultado obtido no experimento com a medida-F de 48,19. Percebe-se que a taxa de classificação correta se sobressai em dois pontos, na diagonal principal e na classificação da espécie Tricogaster, coluna vertical formada na matriz de confusão. Pela diagonal principal pode ser destacada a excelente taxa de recuperação da espécie Tricogaster de 100%, seguido pela Dourado com 90% e Molinesia Preta com 80%. Por outro lado, a espécie Tricogaster foi a que teve mais Falso-Negativo na classificação. É possível observar também que as espécies que tiveram as menores taxas de classificação correta foram a Kinguio e a Platy Sangue, ambas obtiveram classificação correta de apenas 10% de suas imagens, o que levou para baixo a Medida-F geral.

A hipótese de classificação errada das espécies conforme mostrada na matriz de confusão pode estar na complexidade da base de imagens, pois as classes possuem características muitas similares entre si. Além disso, pode ser justificada pela quantidade de pontos de interesse detectados pelo algoritmo de extração, pois quanto mais pontos de interesse analisados de uma classe, melhor será sua aprendizagem e, consequentemente, mais imagens poderão ser classificadas devido a ter mais descrição da classe como um todo, podendo abranger até características de outras classes.

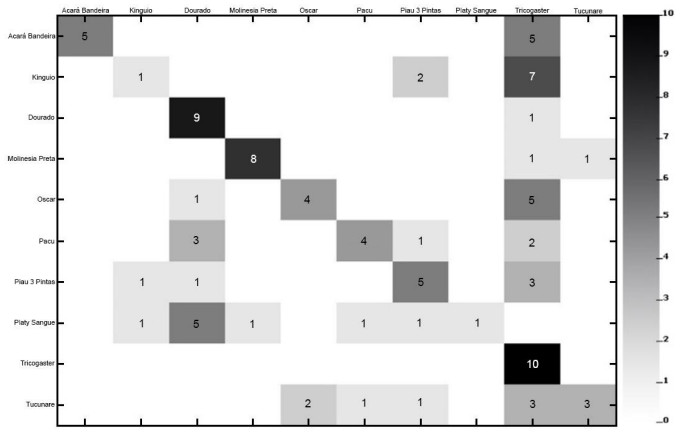


Figura 9. Matriz de confusão mostrando o número de imagens classificadas corretamente para cada espécie de peixe para o resultado de maior Medida-F (banco de imagens aquarioSeg10e, abordagem supervisionada, geração da sentença pela ordenação hierárquica e tamanho de alfabeto 2400).

Essa discrepância de quantidade de informação para cada classe resultou em uma diferença grande de classificação de uma espécie para outra. Ao visualizar a matriz de confusão da classificação é possível notar a predominância da classe Tricogaster na classificação. Nota-se por meio da exibição da quantidade de imagens classificadas apresentadas na vertical. Essa classe possui a maior quantidade de pontos de interesse detectados, uma vez que possui 21,5% do total de pontos identificados. Consequentemente, para essa espécie foram classificadas 37% de todas as imagens. Logo, a espécie Tricogaster foi a classe que mais obteve extração de pontos de interesse nas imagens que, consequentemente, teve maior descrição de características da classe. Como resultado, obteve 100% de precisão na classificação, porém resultou em uma grande quantidade de Falso-Negativo, prejudicando a precisão de outras espécies.

Em geral, as classes de maior poder descritivo classificaram mais imagens, pois abrangeu e descreveu mais características quando comparadas às classes com menos pontos de interesse detectados. Em [38] já afirmavam em seus trabalhos que a capacidade de reconhecer padrões de uma imagem sobre um conjunto de imagens depende da quantidade de informações que se conhece a priori do objeto em questão.

O processo de reconhecimento não é uma tarefa trivial, pois os peixes podem aparecer nas imagens em diferentes tamanhos, formas, escalas, orientações, distorções, ruídos, cores diferentes (mesmo para uma mesma espécie) e diferente contexto de fundo. Nesse tipo de experimento, todo ruído nas imagens pode prejudicar na classificação. Logo, esses são desafios que dificultam o processo de classificação de imagens.

VII. CONCLUSÕES

Na visão computacional, a tarefa de classificação de peixe apresenta aos pesquisadores uma série de desafios. O ambiente natural onde vivem os peixes dificulta as abordagens como a proposta neste trabalho, a exemplo disso podemos citar a iluminação e a subtração de fundo. Esse tipo de problema abrange a teoria computacional e os algoritmos, pois treinar o algoritmo sem um conhecimento completo da classe é um desafio. Neste trabalho, abordamos uma nova proposta para converter imagens em sentenças, ou cadeias de caracteres, baseando-se na metodologia do BOVW, como proposto por [27].

Este trabalho mostrou-se encorajador para seguir com novas propostas de estudos e aprimoramentos, uma vez que houve uma melhora no desempenho conforme é usado valores altos de alfabetos, combinados com a construção do alfabeto pela forma supervisionada. Além disso, a abordagem proposta neste trabalho abre novas frentes de investigação ao permitir novas formas de representar uma imagem por meio de cadeias de caracteres quando comparadas às técnicas usadas em [9, 10].

Como trabalho futuros pretendemos estudar novas formas de geração de sentença a partir dos pontos de interesse detectados; expandir para outros algoritmos de indução gramatical; uso de autômatos adaptativos para realizar a

classificação; verificar a viabilidade do uso de gramáticas regulares e outras que usam exemplos positivos e negativos podem ser alternativas para melhorar o desempenho da classificação; usar uma estratégia de recuperação de erros e planejar um experimento para classificação de objetos a partir de um conjunto de imagens utilizados na literatura.

AGRADECIMENTOS

Os autores agradecem o apoio financeiro da CAPES – Coordenação de Aperfeiçoamento de Pessoal de Nível Superior fornecido ao aluno de doutorado Marcelo Rafael Borth, o qual este trabalho é resultado.

REFERÊNCIAS

[1] ROVA, A.; MORI, G.; DILL, L. M. One Fish, Two Fish, Butterfish, Trumpeter: Recognizing Fish in Underwater Video. In IAPR Conference on Machine Vision Applications, pp. 404-407, 2007.

[2] HEITHAUS, M. R.; DILL, L. M. Food availability and tiger shark predation risk influence bottlenose dolphin habitat use. Ecology, 83(2), pp. 480-491, 2002.

[3] ZION, B.; SHKLYAR, A.; KARPLUS, I.: In-vivo fish sorting by computer vision. Aquacultural Engineering 22, pp. 165-179, 2000.

[4] BOWEN, M.; MARQUES, S.; SILVA, L.; VONO, V.; GODINHO, H. Comparing on Site Human and Video Counts at Igarapava Fish Ladder, Southeastern Brazil. Neotropical Ichthyology, vol. 4, pp. 291–294, 2006.

[5] FERNANDEZ, D. R.; AGOSTINHO, A. A.; BINI, L. M. Selection of an Experimental Fish Ladder Located at the Dam of the Itaipu Binacional, Paraná River, Brazil. Brazilian Archives of Biology and Technology, vol. 47, no. 4, pp. 579–586, 2004.

[6] CHAN, D.; HOCKADAY, S.; TILLETT, R.; ROSS, L. A Trainable N-Tuple Pattern Classifier and its Application for Monitoring Fish Underwater," in International Conference on Image Processing and its Applications, pp. 255–259, 1999.

[7] HOGGARTH, D.; ABEYASEKERA, S.; ARTHUR, R.; BEDDINGTON, J. Stock Assessment for Fishery Management: A Framework Guide to the Stock Assessment Tools of the Fisheries Management Science Programme. FAO Fisheries Technical Paper, Rome, Technical Report 487, 2006.

[8] NERY, M. S. Determining the Appropriate Feature Set for Fish Classification Tasks. Graphics, Patterns and Images, SIBGRAPI Conference on, pp. 173-180, 2005.

[9] NETO, L. C. B.; HIRAKAWA, A. R.; MASSOLA, A. M. A. Aplicação de técnicas adaptativas em reconhecimento de formas. In: Segundo Workshop de Tecnologia Adaptativa - WTA. EPUSP, 2008.

[10] COSTA, E. R.; HIRAKAWA, A. R.; NETO. J. J. An adaptive alternative for syntactic pattern recognition. Proceeding of 3rd International Symposium on Robotics and Automation, ISRA, v.1, 2002.

[11] SIVIC, J.; ZISSERMAN, A. Video Google: A text retrieval approach to object matching in videos. In Proceedings of ICCV, volume 2, pages 1470–1477, Nice, France, oct., 2003.

[12] CSURKA, G.; DANCE, C. R.; FAN, L.; WILLAMOWSKI, J.; BRAY, C. Visual categorization with bags of keypoints. In Workshop on Statistical Learning in Computer Vision, ECCV, pp. 1–22, 2004.

[13] LOWE, D. G. Distinctive image features from scale-invariant keypoints. International Journal of Computer Vision, vol. 60, no. 2, pp. 91–110, 2004.

[14] AKRAM, H. I.; DE LA HIGUERA, C. XIAO, H. ECKERT, C. Grammatical inference algorithms in matlab. In ICGI 2010: Proceedings of the 10th International Colloquium on Grammatical Inference. Valencia, Spain: Springer-Verlag, 2010.

[15] BAY, H.; ESS, A.; TUYTELAARS, T.; VAN GOOL, L. Speeded-Up Robust Features (SURF). Computer Vision and Image Understanding, vol.110, no. 3, pp. 346–359, 2008. Disponível em: <http://dx.doi.org/10.1016/j.cviu.2007.09.014>

[16] DE LA HIGUERA, C. Grammatical Inference: Learning Automata and Grammars. Cambridge University Press, 2010.

[17] FRIGUI, H.; KRISHNAPURAM, R. A robust competitive clustering algorithm with applications in computer vision. IEEE transactions on pattern analysis and machine intelligence, 21, 450–465, 1999.

[18] JAIN, A. K.; FLYNN, P. Advances in image understanding. IEEE Computer Society Press. Chap. Image segmentation using clustering, pp. 65–83, 1996.

[19] SHI, J.; MALIK, J. Normalized cuts and image segmentation. IEEE transactions on pattern analysis and machine intelligence, 22, 888–905, 2000.

[20] IWAYAMA, M.; TOKUNAGA, T. Cluster based text categorization: a comparison of category search strategies. Pages 273–281 of: Proceedings of the 18th ACM international conference on research and development in information retrieval, 1995.

[21] SAHAMI, M. Using machine learning to improve information access. Ph.D. thesis, Computer Science Department, Stanford University, 1998.

[22] BHATIA, S.; DEOGUN, J. Conceptual clustering in information retrieval. IEEE transactions on systems, man and cybernetics, 28(B), 427–436, 1998.

[23] BALDI, P.; HATFIELD, G. DNA microarrays and gene expression. Cambridge University Press. 2002.

[24] SAHOO, N.; CALLAN, J.; KRISHNAN, R.; DUNCAN, G.; PADMAN, R. Incremental hierarchical clustering of text documents. In: 15th ACM international conference on Information and knowledge management, pp. 357–366, New York, NY, USA, 2006.

[25] JAIN, A. K.; MURTY, M. N.; FLYNN, P. J. Data clustering: a review. ACM Computing Surveys, v.31, n.3, p.264-323. 1999.

[26] JAIN, A. K. Data clustering: 50 years beyond K-means. Pattern Recognition Letters, v.31, n.8, pp. 651-666, 2010. Disponível em: <https://s3-us-west-2.amazonaws.com/mlsurveys/45.pdf>.

[27] PISTORI, H.; CALWAY, A.; FLACH, P. A new strategy for applying grammatical inference to image classification problems. In: IEEE International Conference on Industrial Technology (ICIT), pp.1032–1037, 2013.

[28] ZHANG, E.; MAYO, M. Improving bag-of-words model with spatial information. In 25th International Image and Vision Computing New Zealand (IVCNZ), pp. 1-8, nov., 2010.

[29] ZHANG, C.; WANG, S.; HUANG, Q.; LIU, J.; LIANG, C.; TIAN, Q. Image classification using spatial pyramid robust sparse coding. Pattern Recognition Letters, vol. 34, no. 9, pp. 1046 – 1052, 2013. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0167865513000573>.

[30] PEDROSA G.; TRAINA, A. From bag-of-visual-words to bag-of-visual phrases using n-grams. In: 26th Conference on Graphics, Patterns and Images (SIBGRAPI), pp. 304–311, aug., 2013.

[31] LAZEBNIK, S.; SCHMID, C.; PONCE, J. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In: IEEE Conference on Computer Vision and Pattern Recognition, vol. 2, 2006.

[32] YUAN, J.; WU, Y.; YANG, M. Discovery of collocation patterns: from visual words to visual phrases. In: IEEE Conference on Computer Vision and Pattern Recognition, pp.1-8, jun., 2007.

[33] DACONTA, M.; OBRST, L.; SMITH, K. The Semantic Web. Wiley Publishing Inc., 2003.

[34] RIBAS, L. C.; BORTH, M. R.; CASTRO JR., A. A.; GONÇALVES, W. N.; PISTORI, H. Grammatical Inference and SIFT for Scene Recognition. X Workshop de Visão Computacional (WVC), Uberlândia-MG, 2014.

[35] Neto, J. J. Solving complex problems efficiently with adaptative automata. Conference on Implementation and Application of Automata – CIAA, Julho, 2000.

[36] 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, Vol. 2494, Pretoria, South Africa, Jul. 23-25, Springer-Verlag, pp. 234-250, 2001.

[37] Pistori, H.; Neto, J. J. AdapTree - Proposta de um Algoritmo para Indução de Árvores de Decisão Baseado em Técnicas Adaptativas. Anais Conferência Latino Americana de Informática - CLEI. Montevideo, Uruguai, Novembro, 2002.

[38] PEURA, M.; LIVARINEN, J. Efficiency of Simple Shape Descriptors. In: Aspects of Visual Form, pp. 443–451. World Scientific, Singapore, 1997.

Tamanho do Alfabeto (k)	**Medida-F (%)**							
	Aquario10e				AquarioSeg10e			
	Não Supervisionado		Supervisionado		Não Supervisionado		Supervisionado	
	Radial	Hier.	Radial	Hier.	Radial	Hier.	Radial	Hier.
10	**1.82**	**1.82**	**1.82**	**1.82**	1.81	1.81	3.29	**3.39**
60	7.15	6.56	**9.48**	9.10	8.42	10.32	**10.81**	8.60
120	**14.06**	10.17	9.46	7.46	15.31	14.21	**16.09**	13.68
240	**10.52**	10.19	5.88	6.73	15.74	15.63	**18.70**	15.99
480	15.93	**16.44**	15.05	13.49	19.40	23.59	**26.36**	20.43
960	18.27	17.03	**19.82**	17.96	31.34	31.27	**38.10**	36.43
1200	**19.54**	19.02	18.08	19.19	33.49	22.87	35.47	**40.78**
1440	19.79	19.09	**20.11**	18.19	21.66	28.53	**46.91**	39.31
1800	19.38	**20.19**	19.05	19.25	23.27	31.41	35.98	**43.80**
2100	**20.39**	18.49	20.30	20.28	**40.47**	30.61	39.01	35.60
2400	20.64	18.68	18.94	***22.31***	23.81	21.02	40.72	***48.19***
2700	19.07	18.43	**20.60**	20.29	20.96	24.39	38.29	**42.14**
3000	19.20	**21.51**	21.08	21.31	23.36	23.27	**46.99**	46.21

Tabela 2. Medidas-F para a abordagem proposta executada no banco de imagens aquario10e e aquarioSeg10e. Os valores mais altos para cada tamanho do alfabeto para ambos os bancos de imagens estão marcados em negrito. A Medida-F mais alta para cada banco de imagens está marcada com asteriscos.

Modelo Adaptativo para um Framework Educacional

A. M. Baldi, A. A. C. Junior, E. S. C. Rodrigues, F. A. Rodrigues

***Resumo* — Esse artigo descreve um modelo adaptativo para um *framework* educacional, que se utiliza de dados atencionais a partir de um aparelho eletroencefalograma e do comportamento com base em respostas de exercícios como estímulos de entrada para a modificação do funcionamento da interface de ensino com a tecnologia de adaptatividade. É esperado que o modelo proposto auxilie e aprimore o processo educacional através da individualização e personalização do ensino.**

***Palavras-chave* — Educação, Adaptatividade, Eletroencefalograma, Aprendizagem, Comportamento, Autômato Adaptativo, modelagem.**

I. INTRODUÇÃO

Com a evolução da tecnologia, as crianças e jovens possuem acesso a uma quantidade maior de informações, são mais críticos, questionadores e adoram desafios [6]. É de conhecimento que cada indivíduo possua um tempo diferente de aprendizagem e uma maior facilidade em fixar o conteúdo de uma determinada matéria. É preciso admitir uma metodologia individualizada e personalizada na educação para que exista uma melhor abordagem do conteúdo ministrado nas escolas, utilizando um material didático de suporte composto por diversos recursos visuais, auditivos e interativos.

A proposta de um modelo adaptativo para um Framework Educacional permeia o uso da tecnologia na educação e o desenvolvimento de ambientes e sistemas educacionais que possam oferecer um suporte adequado ao estudo e à aprendizagem dos alunos. Tal proposta é classificada como uma coleção de Objetos de Aprendizagem, recursos digitais que podem ser reutilizados em diferentes contextos [8], desta forma o sistema proposto pode ser utilizado em diferentes níveis de ensino, por diferentes faixas etárias, bastando a modificação do conteúdo no banco de dados. Deve-se destacar, no entanto, que o papel e a atuação do professor, mesmo em ambientes como esse, é imprescindível para o acompanhamento e a mediação adequada no desenvolvimento do aluno e favorece a interação homem-máquina no processo de ensino e aprendizagem.

O uso da Tecnologia Adaptativa na proposta auxilia no processo de ensino através da individualização e personalização dos Objetos de Aprendizagem. A Tecnologia Adaptativa é um modelo que possui a propriedade de modificar seu próprio comportamento, sem a interferência de qualquer agente externo [3]. Esse modelo é de extrema importância em relação a construção de programas de cunho educacional pois possui uma alta capacidade de aprendizagem agregada [4].

O modelo aqui proposto serve como base para a implementação de um protótipo que seja capaz de aferir as diversas ondas cerebrais referentes a atenção do indivíduo e características pessoais de aprendizagem do usuário, fazendo uso da adaptatividade para modificar o funcionamento de uma interface didática. O protótipo consiste em uma implementação desse modelo, um software com interface humano-máquina, que faz uso da eletroencefalografia para a verificação dos níveis atencionais e se utiliza desses dados e dados de respostas do usuário para a modificação da interface didática. A construção do protótipo tem a finalidade de validar o modelo proposto.

II. FUNDAMENTAÇÃO

A. SISTEMAS GUIADOS POR REGRAS

Autômatos e máquinas de estados constituem uma classe de ferramentas formais matemáticas que se mostram bastante adequadas para a descrição e modelagem de uma grande diversidade de sistemas do mundo real, e por essa razão sua utilização é muito frequente nas atividades associadas à resolução de uma vasta gama de problemas de áreas variadas, entre as quais se destaca a engenharia de computação e afins.

Como característica do seu funcionamento, em cada estágio de sua operação esses sistemas assumem diferentes configurações, as quais de uma certa maneira codificam uma espécie de memória histórica do funcionamento do sistema, bem como a situação a que tal sistema está sendo exposto em cada momento.

Assim, usualmente, a configuração de um sistema guiado por regras deve compreender o conteúdo de todo o conjunto dos elementos disponíveis de memória, que sejam responsáveis pelo armazenamento de informações relevantes ao correto funcionamento do sistema.

A operação do sistema é determinada por essa configuração, em conjunto com outro grupo de informações que, embora não façam parte integrante do sistema, são diretamente responsáveis por alimentá-lo com os estímulos externos, determinando assim os passos seguintes de operação a serem executados pelo dispositivo.

A. M. Baldi é estudante de graduação em Ciência da Computação na Universidade Federal de Mato Grosso do Sul – Câmpus Ponta Porã.

A. A. C. Junior é professor na Univers idade Federal de Mato Grosso do Sul – Câmpus Ponta Porã.

E. S. C. Rodrigues é professora na Universidade Federal de Mato Grosso do Sul – Câmpus Ponta Porã.

F. A. Rodrigues é professor na Universidade Federal de Mato Grosso do Sul – Câmpus Ponta Porã.

Observa-se, portanto, que o conjunto de regras é previamente conhecido e determina o comportamento do sistema durante toda a sua operação. Dessa forma, a despeito da maneira como a configuração corrente foi atingida, tal configuração, complementada pela sequência dos estímulos externos a que o sistema é submetido, determina integralmente seu comportamento futuro, sem nenhuma perspectiva de alteração das regras que definem esse comportamento.

A operação de um sistema dessa natureza se dá, portanto, pela movimentação sucessiva da tecnologia que o implementa, de uma configuração para outra, como resposta à sequência de estímulos externos a que é submetida, os quais, na maioria dos casos, são consumidos a partir de uma sequência de estímulos de entrada.

Sem perda de generalidade, pode-se afirmar que tais sistemas iniciam sua operação em alguma configuração inicial pré-estabelecida, e que obedece a um conjunto bem conhecido e invariável de condições, impostas quando da concepção do sistema.

O comportamento de um sistema guiado por regras depende, exclusivamente, de um conjunto finito de regras, responsáveis por mapear cada possível configuração do sistema em uma configuração seguinte correspondente, em resposta a algum estímulo de entrada recebido.

B. SISTEMAS ADAPTATIVOS

Os Sistemas Adaptativos também possuem um conjunto de regras. Sem perda de generalidade, pode-se afirmar que tais sistemas iniciam sua operação em alguma configuração inicial pré-estabelecida, e que obedece a um conjunto previamente conhecido, no entanto, variável de regras. Tais regras podem ser modificadas em situações e condições específicas, detectadas a partir dos estímulos externos percebidos.

Nesse contexto, os sistemas adaptativos podem ser descritos como uma dupla composta por uma componente que define um dispositivo guiado por regras não-adaptativo (ou sistema subjacente) e uma componente conhecida como camada adaptativa que define, através das ações adaptativas, as possíveis transformações sobre o dispositivo subjacente [1].

Em cada fase de operação desses sistemas, eles assumem alguma configuração, que é definida como sendo o conjunto de todos os elementos que definem o estado atual do sistema. Dessa forma, o sistema opera sucessivamente, movendo-se de uma configuração para outra, em resposta a estímulos recebidos como entrada. Sem perda de generalidade, pode-se afirmar que tais dispositivos partem de alguma configuração inicial, seguindo um conjunto de regras pré-definido.

Nesse contexto, para definir o modo de operação de um sistema adaptativo, será utilizada a seguinte terminologia:

(i) Passo subjacente: refere-se à mudança de configuração do dispositivo subjacente (não-adaptativo) correspondente.

(ii) Passo adaptativo: refere-se à mudança de configuração do sistema adaptativo. Neste caso, a aplicação de uma regra acompanha a execução de ações adaptativas que, por sua vez, podem provocar alterações no comportamento subsequente do dispositivo subjacente.

A Tecnologia Adaptativa é um campo de pesquisa emergente, ainda em fase de desenvolvimento, com foco sobre os problemas relacionados com a organização de sistemas complexos em ciência da computação, engenharia de computação e tecnologia da informação [1]. O uso da tecnologia adaptativa no framework educacional possui uma finalidade bastante simples: obter uma capacidade de, sem a interferência de qualquer agente externo, tomar a decisão de modificar seu próprio comportamento, em resposta ao comportamento do usuário. Desta forma, é adquirido um comportamento de acordo com as respostas às perguntas dos conteúdos na interface do usuário (dados de conhecimento) e de acordo com a aferição de dados atencionais relevantes através de um dispositivo eletroencefalograma (dados psicofisiológicos).

Os dados de conhecimento e psicofisiológicos são decisivos quanto ao tipo de alteração comportamental resultante do exercício da adaptatividade. Duas instâncias idênticas de um mesmo sistema adaptativo podem evoluir para comportamentos finais completamente diferentes, de acordo com a diversidade dos eventos a que forem submetidas em suas operações [5].

C. FORMALISMOS HABITUAIS E ALGUMAS DE SUAS APLICAÇÕES

A seguir, um panorama dos principais formalismos para modelagens de sistemas adaptativos e algumas aplicações:

- Autômatos Adaptativos

Baseado em autômatos de pilha estruturados, são formalismos com recursos de auto modificação, que possuem a característica de representar linguagens que são dependentes de contexto. Uma aplicação é a inferência de linguagens regulares e livres de contexto [11].

- Statecharts Adaptativos

Proposto pela tese de Almeida Junior em 1995, incorporando adaptatividade ao formalismo clássico de statecharts concebido por David Harel [12].

- Statecharts Adaptativos Sincronizados

Uma evolução dos Statecharts Adaptativos, acrescentando redes de Petri restritas [13].

- Redes de Markov Adaptativas

Uso das Redes de Markov para adaptatividade em fenômenos estocásticos. Uma aplicação é a geração automática de músicas por computador [14].

- Gramáticas Adaptativas

Apresentam capacidade de expressar os aspectos léxico, sintático e semântico de uma linguagem para a adaptatividade. Uma possível aplicação é no processamento de textos em linguagem natural [15].

- Tabelas de Decisão Adaptativas

Apresentam uma aplicação interessante do conceito de dispositivos adaptativos para o campo de sistemas de informação [16].

- Árvores de Decisão Adaptativas

Adequadas para as tarefas de aquisição e representação de conhecimento. Uma de suas aplicações é na construção do adaptree-E, um algoritmo de aprendizagem computacional proposto por Hemerson Pistori[17].

- Autômatos Finitos Adaptativos

Capazes de reconhecer quaisquer linguagens que possam ser reconhecidas por outros modelos computacionais conhecidos, se assemelhando aos Autômatos Adaptativos, porém com características restritivas tornando-os próximos aos Autômatos Finitos [18].

- Redes de Petri Adaptativas

Formalismo que envolve uma camada adaptativa nas Redes de Petri, tornando-as auto modificáveis. Uma aplicação implementada é o Visualizador Gráfico para Redes de Petri Adaptativa [19].

- Máquinas de Turing Adaptativas

Ambiente de execução no qual o programa a ser executado é armazenado na fita da máquina de Turing, permitindo a auto-modificação [20].

- Linguagens de Programação Adaptativas

Formas de modificar a linguagem de programação tradicional em códigos adaptativos. Um trabalho realizado é demonstrado em [21] com formas de programar códigos adaptativos.

D. AUTÔMATOS ADAPTATIVOS NA MODELAGEM DO FRAMEWORK EDUCACIONAL

Para modelar o Framework educacional, utilizaremos o formalismo de autômato adaptativo, proposto por Ricardo Luis de Azevedo da Rocha e João José Neto. Os autômatos adaptativos são ferramentas confiáveis para modelagem de ambientes físicos, possuindo poder computacional equivalente à Máquina de Turing [10]. O Autômato Adaptativo incorpora a característica dos Autômatos de Estados Finitos, desenvolvendo novas configurações em resposta a um estímulo externo. Os estados de um Autômato Adaptativo podem passar por consultas, exclusões e inclusões para a finalidade de atingir o objetivo final de aceitação.

Especificamente em nossa modelagem, vamos utilizar a coleção de objetos de aprendizagem formada por textos, imagens, vídeos e questionários por diversas lições.

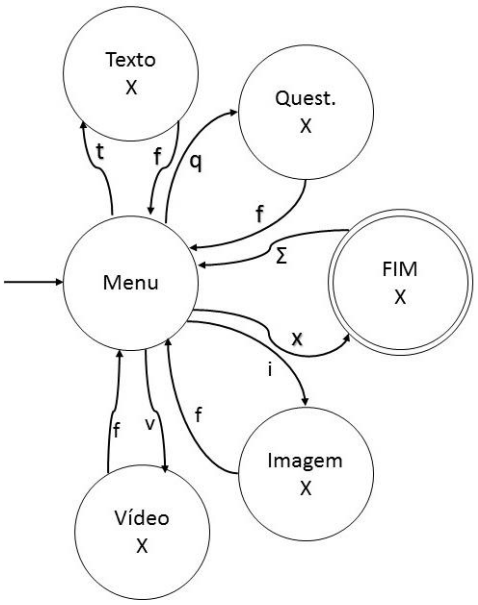


Figura 1. Autômato Representativo.

Na Figura 1 podemos observar a representação do funcionamento de um software educacional não-adaptativo que interage com o usuário para a mudança dos estados. Cada estado do autômato representa o funcionamento atual do framework. Por exemplo, o estado Texto X representa a exibição do texto encontrado na lição X (Sendo X=1,2,3,...). O estado Quest. X representa um questionário (formado por perguntas e opções de respostas) sobre a lição X.

Cada transição representa a ação tomada no sistema. As transições t,i,v,q,f,x representam respectivamente: requisição para objeto de aprendizagem texto da lição x, requisição para objeto de aprendizagem imagem da lição x, requisição para objeto de aprendizagem vídeo da lição x, requisição para objeto de aprendizagem questionário da lição x, finalização da execução do objeto de aprendizagem, finalização da lição dos objetos de aprendizagem. Podemos observar que a transição Σ retorna ao estado inicial de execução, no entanto, antes do retorno é feita a modificação da lição (X). É importante ressaltar que nesse modelo o usuário informa o tipo de objeto de aprendizagem e lição de sua preferência para a aprendizagem.

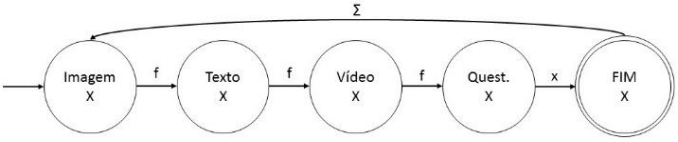


Figura 2. Autômato Finito Inicial.

Já na figura 2, podemos visualizar o Autômato Finito não-adaptativo que é o ponto "inicial" da modelagem do Autômato Adaptativo. Como podemos perceber, o usuário já deixa de ser o controlador do objeto de aprendizagem e da lição. Esse

autômato pode ser considerado a parte subjacente do dispositivo adaptativo formado pelo Framework Educacional. De acordo com as regras apresentadas, o Framework Educacional apresentará uma sequência dos objetos de aprendizagem Imagem, Texto, Vídeo e Questionário em uma determinada lição, trocando para a próxima lição ao final da execução. Nesse modelo, o usuário será forçado a passar por todos os objetos de aprendizagem, abandonando a ideia de uma individualização e personalização do conteúdo e aumentando a possibilidade de dispersão de atenção e abandono do software por parte do usuário.

Com a implantação da camada adaptativa, esse modelo torna-se inteligente pois há a possibilidade de substituir um objeto de aprendizagem por outro e uma lição por outra. Dessa forma, ao utilizar o framework, a sequência de conteúdo é atualizada, criando novos estados pelo qual o framework poderá executar. Note que o material ao qual o aluno será exposto não estará necessariamente em uma sequência, podendo ser considerado "novo".

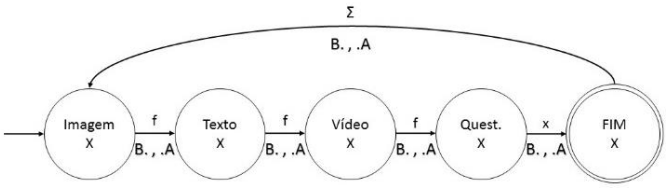


Figura 3. Autômato Adaptativo.

Na figura 3, temos o autômato adaptativo modelado. As ações adaptativas estão representadas por: B. que são as ações adaptativas aplicadas antes da transição e .A que são as ações adaptativas aplicadas após a transição.

Uma questão importante é em relação às regras adaptativas. Essas regras devem ser criadas pelo professor, pedagogo ou até mesmo médico especialista dependendo da regra a ser inserida. Por exemplo, regras que se utilizam de padrões nas ondas cerebrais só podem ser entendidas por um médico e, portanto, só podem ser criadas com o auxílio dele.

Como exemplo, podemos citar duas regras a serem aplicadas no Framework educacional:

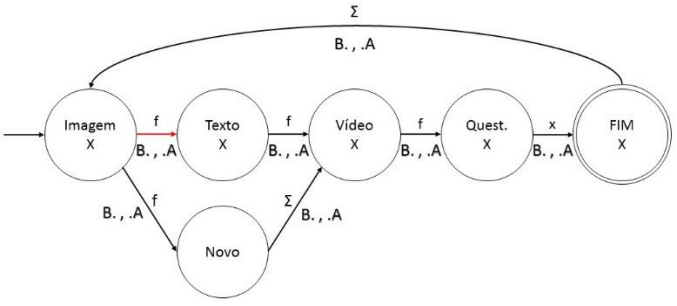


Figura 4. Retirada de objeto de aprendizagem.

1. Retirar o objeto de aprendizagem ao qual o aluno presta menos atenção, de acordo com os dados do eletroencefalograma (Fig. 4). Observe que a transição em vermelho não existe mais, dando origem a uma nova transição e estado (novo).

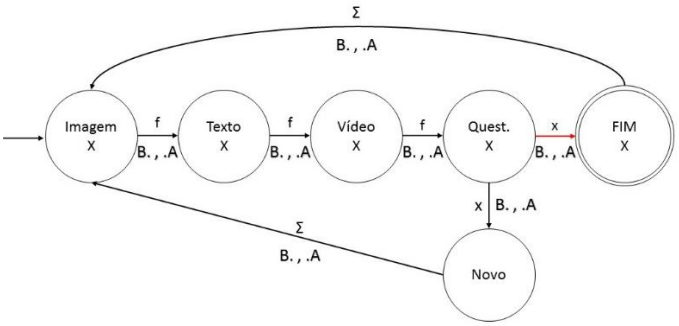


Figura 5. Lição não aprendida.

2. Voltar a lição não aprendida, de acordo com a resposta errada no questionário (Fig. 5). Observe que a transição em vermelho não existe mais, dando origem a uma nova transição e estado (novo).

É importante notar que dependendo da maneira na qual as regras de adaptatividade estão dispostas no dispositivo adaptativo, as instâncias podem evoluir de uma forma completamente diferente. Também é interessante pensar que é possível criar um dispositivo adaptativo sob a camada adaptativa, de forma que as regras adaptativas sejam também adaptadas.

III. A ADAPTATIVIDADE NO PROCESSO DE APRENDIZAGEM

"Atento", "descansado" e "calmo" são algumas das palavras que descrevem o estado ideal do indivíduo durante o processo de aprendizado. Essas palavras possuem um impacto significativo na aprendizagem. Por exemplo: um estudante cansado possui um rendimento menor que um estudante atento e um estudante desatento pode indicar que não gosta da maneira que o conteúdo está sendo transmitido. Esses estados podem mudar devido a diversas circunstâncias: circunstâncias físicas, circunstâncias emocionais e circunstâncias cognitivas [2].

O modelo de framework educacional aqui descrito utiliza a combinação de duas abordagens:

(i) Psicofisiológico: Faz o monitoramento de sensores biométricos (EEG) para inferir estados sobre o corpo e a mente.

(ii) Conhecimento: Faz o monitoramento das ações do usuário sobre as perguntas do programa educacional (erros e acertos) para inferir a aprendizagem do usuário sobre determinado assunto.

Figura 6. Abordagens do Modelo de Framework Educacional.

É importante notar que a combinação das duas abordagens (Fig. 6) fornece uma maior precisão na inferência do estado do usuário, evitando também interpretações erradas. Se existisse apenas o sensor EEG como estímulo externo, o mecanismo adaptativo faria a inferência apenas dos estados relacionados a atenção do indivíduo, sem levar em conta os erros e acertos do usuário no programa, dando a possibilidade de apresentar conteúdos em modo aleatório na interface sem individualizar e personalizar o método de ensino para atrair a atenção do usuário. Um bom exemplo descrito no livro Intelligent Adaptive Systems: An Interaction-centered Design Perspective é o detector de mentiras, que verifica batimentos cardíacos rápidos irregulares, excesso de sudorese e temperatura elevada, mas que cabe ao operador da máquina (que pode ser considerado o mecanismo adaptativo em questão) verificar se o indivíduo em interrogatório está nervoso, doente ou se o ambiente está com uma temperatura elevada [2]. Existem ainda diversas outras abordagens e sensores para estímulos externos que serão mencionados na seção "outras abordagens" desse artigo e que tornam o sistema ainda mais preciso.

Os sistemas não-adaptativos não possuem características de individualização e personalização do conteúdo e utilizam apenas a interação com o usuário para formar a base de conteúdo.

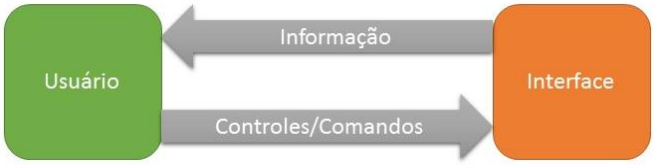


Figura 7. Modelo Não-adaptativo de Framework Educacional.

Desta forma, o usuário que utiliza um sistema educacional não-adaptativo (Fig. 7) possui mais chances de perder o interesse na aprendizagem da matéria.

O modelo proposto nesse artigo possui um mecanismo adaptativo que atua sobre o funcionamento da interface do usuário, recebendo os dados de monitoria em tempo real do estado do usuário através do monitoramento psicofisiológico e de conhecimento no processo de aprendizagem. O mecanismo funciona em paralelo a diversos módulos e não interrompe nenhuma tarefa em execução.

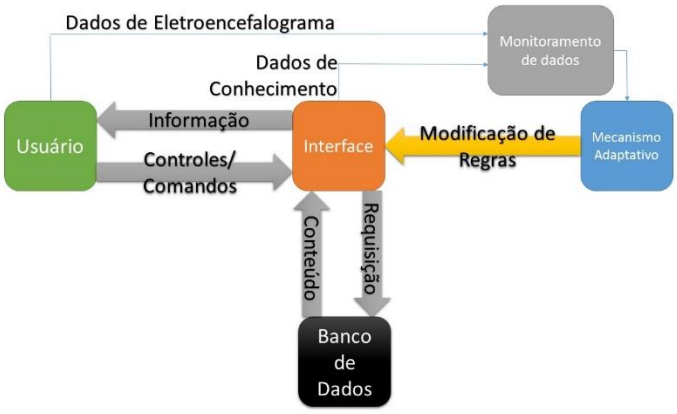


Figura 8. Modelo Adaptativo de Framework Educacional.

O modelo adaptativo (Fig. 8) é composto por quatro módulos:

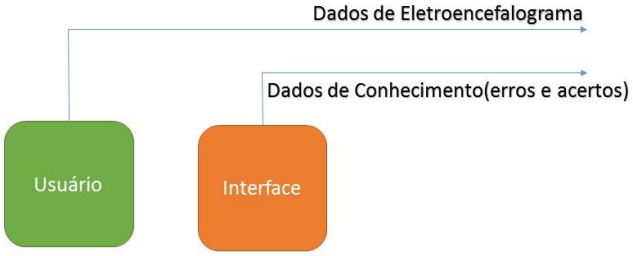


Figura 9. Módulo de Sensoriamento.

(i) Módulo de sensoriamento (Fig. 9), que consiste na aferição dos dados do aparelho de eletroencefalograma e verificação de erros/acertos no programa;

Figura 10. Módulo de Dados.

(ii) Módulo de dados (Fig. 10), que é o algoritmo que faz o monitoramento e avaliação da aprendizagem do indivíduo a partir do módulo de sensoriamento fazendo a extração de características relacionadas ao estado do indivíduo;

Figura 11. Módulo de Conteúdo.

(iii) Módulo de conteúdo (Fig. 11), que é o conjunto de conteúdos instrucionais, tais como vídeos, textos e exercícios para avaliação da aprendizagem;

Figura 12. Módulo Pedagógico.

(iv) Módulo pedagógico (Fig. 12), que é responsável pela verificação do módulo de dados e adequação da interface de ensino através da seleção dos conteúdos do módulo de conteúdo, modificando as regras da interface.

Os módulos anteriormente descritos integram um feedback que constitui a tecnologia adaptativa em si, tendo o módulo pedagógico como o mecanismo adaptativo, que ajusta as regras da interface conforme o feedback de dados do monitoramento. Desta forma, há o ajuste e adequação dos conteúdos na maneira como são mostrados na interface do sistema, aumentando a eficiência do processo de ensino.

O mecanismo de funcionamento desse modelo é correspondido a abstração formal, na qual o comportamento da interface é regido por um conjunto finito explícito de regras que especificam como deve ser a execução. Para cada nova situação em que se encontre as características do usuário, há uma alteração no conjunto de regras que define o funcionamento da interface e consequentemente nas requisições ao conjunto de conteúdo do programa.

V. IMPLEMENTAÇÃO DO PROTÓTIPO

Para validar o modelo proposto de um Framework Educacional que utiliza tecnologias adaptativas, será construído um protótipo utilizando as ferramentas:

A. ELETROENCEFALOGRAMA

O eletroencefalograma, ou EEG, é realizado através de um dispositivo leve, pequeno e prático de se usar, o MindWave Mobile, produzido pela empresa Neurosky.

Figura 13. Neurosky Mindwave

O dispositivo em questão (Fig. 13) é usado na parte externa da cabeça (couro cabeludo) e é responsável pela captura das ondas cerebrais através de um único eletrodo seco que é posicionado na região da testa.

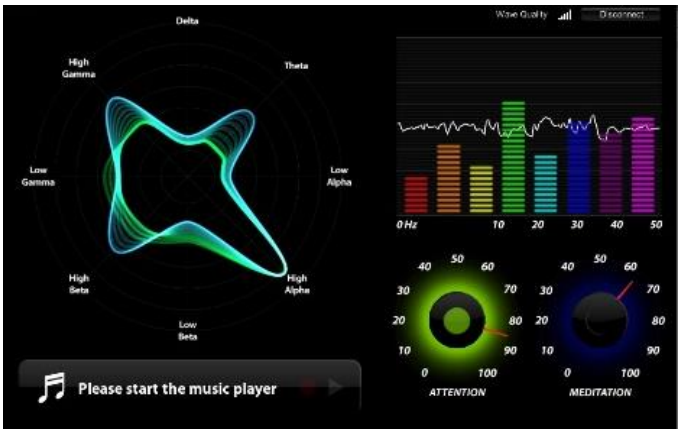


Figura 14. Representação Gráfica dos Dados de Captura do EEG

Os dados aferidos pelo EEG (Fig. 14) são os espectros de potência na forma das ondas: alfa alta, alfa baixa, beta alta, beta baixa, delta, gama alta, gama baixa e teta; valores por porcentagem de meditação e atenção; detecção do piscar de olhos; qualidade, ruído e problemas do sinal EEG. O dispositivo utiliza uma pilha AAA que garante a autonomia de até 8 horas para funcionamento. Os dados do EEG são transmitidos através da tecnologia bluetooth 2.1, que permite a mobilidade do usuário em até 10 metros [7].

B. PROCESSING

O Processing é uma IDE (ambiente de desenvolvimento) e linguagem de programação totalmente Open Source (de código

aberto) que foi criada em 2001 com o princípio de programar facilmente algoritmos que envolvem elementos artísticos em computação gráfica e que têm evoluído desde então para uma ferramenta profissional de desenvolvimento.

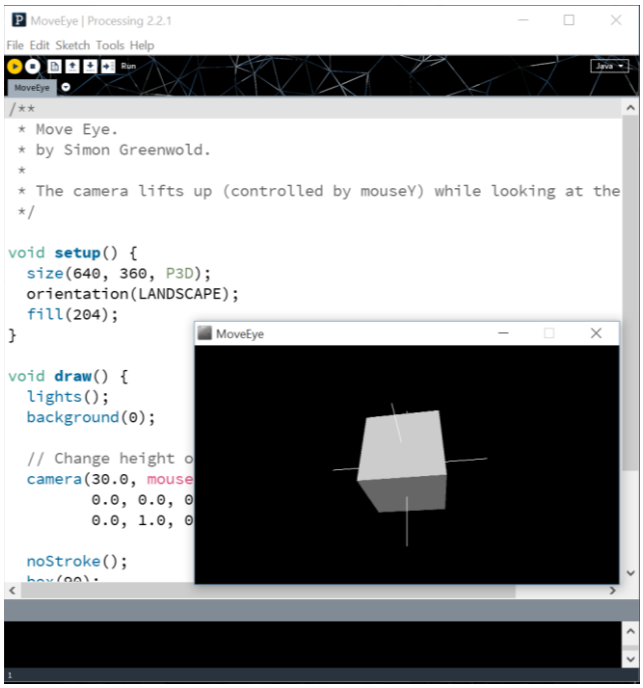


Figura 15. Ambiente de Desenvolvimento Processing

Atualmente, o Processing (Fig. 15) é mais utilizado como uma ferramenta para auxiliar no processo criativo, fornecendo um ambiente simples e intuitivo que trabalha em conjunto a diversas bibliotecas livres e comumente utilizadas.

Para o modelo proposto, as bibliotecas do Processing e sua forma de programação serão úteis na integração com o dispositivo eletroencefalograma e na apresentação de uma interface intuitiva para o usuário, mostrando conteúdo, avaliando e reagindo de acordo com os princípios pedagógicos e metas educacionais.

Como todo algoritmo adaptativo, o software será formado por um dispositivo subjacente e um dispositivo adaptativo. O dispositivo subjacente é a interface de interação com o usuário, que já possui regras definidas de funcionamento e que são modificadas pelo dispositivo adaptativo, o módulo pedagógico.

Dessa forma, o software é capaz de oferecer ao aluno a mesma qualidade de aprendizagem que um professor particular poderia oferecer, adequando a estratégia de aprendizagem de acordo com as necessidades.

VI. OUTRAS ABORDAGENS

Existem diversas outras abordagens que podem ser inclusas nesse modelo de sistema adaptativo, tornando-o mais acurado tanto na aferição da condição do usuário quanto na maneira em que as informações são exibidas.

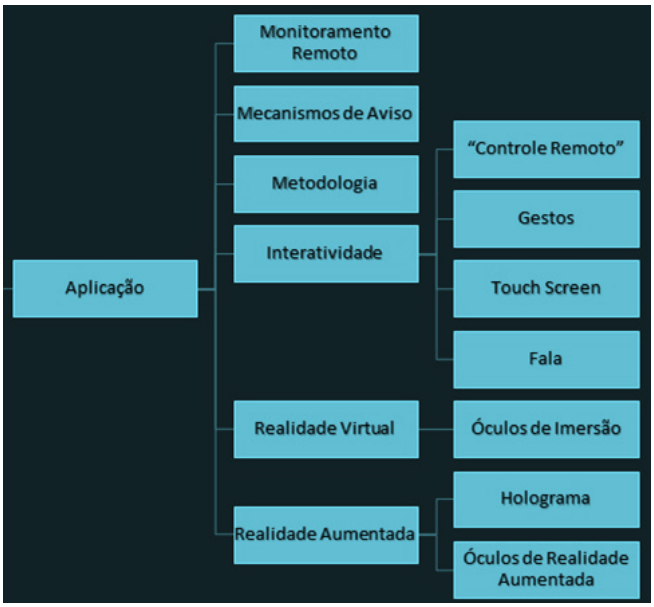


Figura 16. Novas Abordagens na Aplicação

Uma das novas abordagens é em relação à aplicação (Fig. 16), ou seja, em relação à interface e a maneira em que as informações são exibidas para o usuário e como ele interage com elas. Desta forma, algumas técnicas novas estão descritas abaixo:

(i) Monitoramento Remoto: profissionais podem ter acesso ao que é feito na interface em tempo real para ajustar as regras dos mecanismos de exibição. Além disso, podem ter acesso ao histórico de dados e sensores.

(ii) Mecanismos de Aviso: mecanismo que avisa a um profissional caso as modificações geradas pelo mecanismo adaptativo não sejam suficientes para manter a atenção do usuário.

(iii) Controle Remoto: utilização de um controle de movimentos como o Wii Remote para interação com a tela (mouse).

(iv) Gestos: utilização de uma câmera inteligente como o Kinect V2.0 para capturar gestos e interagir com a aplicação

(v) Touch Screen: utilização de tela com reconhecimento de toques para interação com a aplicação.

(vi) Fala: utilização de interação por conversas e comandos de voz.

(vii) Óculos de imersão: utilização de óculos de realidade virtual como o Oculus Rift para a criação de ambientes virtuais de ensino.

(viii) Hologramas: Construir um "avatar" e projetá-lo como um holograma para interagir com o usuário.

(ix) Óculos de Realidade Aumentada: usar um dispositivo como o Google Glass para criar objetos virtuais de interação com o mundo real.

Todas as abordagens anteriores melhoram a interação do usuário com o sistema, com consequência na melhora do processo de aprendizagem. É importante ressaltar que a adaptatividade poderia também estar presente em cada uma das abordagens de interação pois, desta forma, o sistema poderia aprender qual a melhor abordagem de interação para que o usuário aprenda melhor. Exemplo: Usuário aprende melhor utilizando comandos de voz ao invés de toques na tela.

Outras abordagens têm relação com o ambiente de estudos do usuário e são abordagens contextuais. Apesar do modelo proposto nesse artigo não citar nenhuma abordagem contextual, elas estão indiretamente ligadas a maneira em que o usuário está aprendendo e a modificação dessas abordagens resulta em um maior ou menor aprendizado do usuário em relação ao conforto do ambiente.

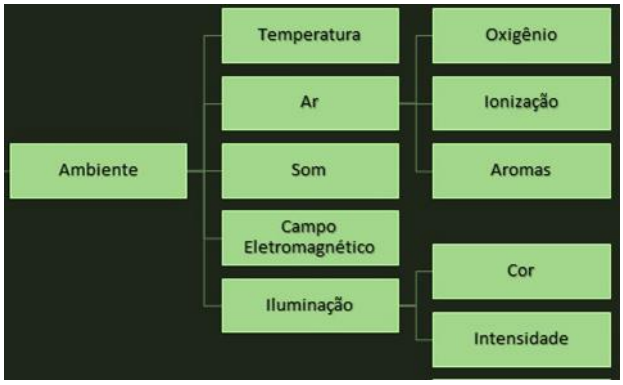


Figura 17. Novas Abordagens no Ambiente de Estudos (contextuais).

As abordagens contextuais ou do ambiente (Fig. 17) estão descritas abaixo e poderiam ser modificadas pelo sistema adaptativo:

(i) Temperatura: A temperatura ideal para que o usuário sinta confortável para estudar.

(ii) Oxigênio: nível de oxigênio ideal no ar permitindo ao usuário um maior foco nos estudos.

(iii) Ionização: benefícios produzidos pelos íons negativos que surtem efeito no foco do usuário.

(iv) Aroma: determinados aromas despertam sensações nos seres humanos e afetam os estudos.

(v) Som: efeitos no aprendizado com músicas e ruídos.

(vi) Campo Eletromagnético: os sistemas sem fio das atuais tecnologias produzem muitos campos eletromagnéticos que afetam o funcionamento do cérebro e consequentemente da qualidade dos estudos.

(vii) Cor da Iluminação: sensações que as cores possibilitam aos seres humanos (cromoterapia).

(viii) Intensidade da Iluminação: intensidades da luz que afetam os estudos.

As abordagens anteriores poderiam ser modificadas e adaptadas no ambiente de estudos através de equipamentos especiais de controle e automação e estariam à disposição do sistema adaptativo.

As próximas abordagens definem diversas variáveis que poderiam ser aferidas pelo sistema adaptativo para uma melhor acurácia na disposição das informações ao usuário.

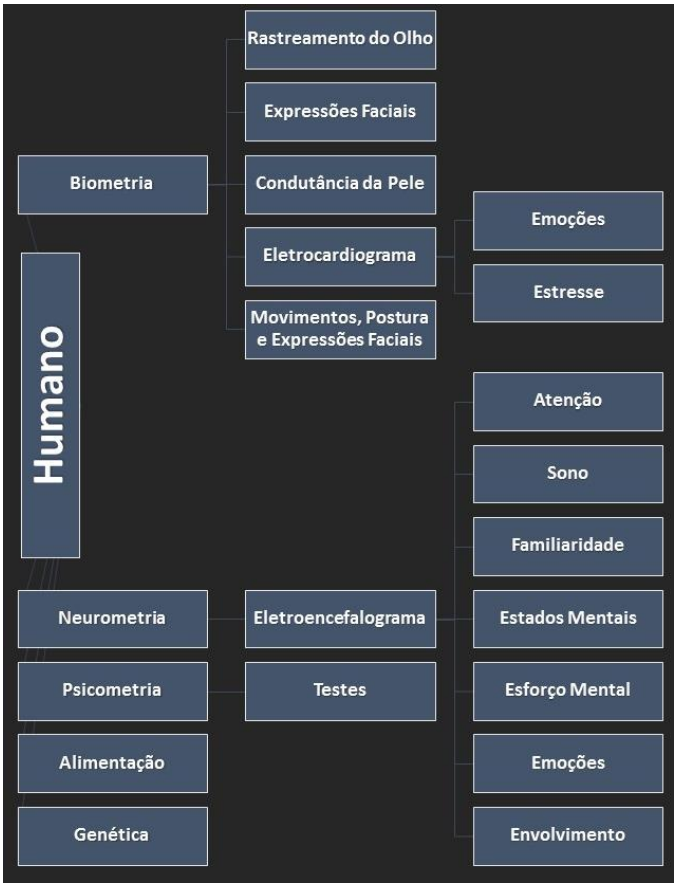


Figura 18. Novas Abordagens no Sensoriamento Humano.

As abordagens de sensoriamento humano (Fig. 18) permitem que o sistema adaptativo saiba com maior precisão o que adaptar na interface do usuário pois aferem outros dados que podem ser imprescindíveis na busca pela personalização e individualização do ensino.

(i) Rastreamento do Olho: observa a dilatação das pupilas, piscar dos olhos, movimento do globo ocular e direção do olhar.

(ii) Expressões Faciais: interpreta as respostas faciais de acordo com o processo de aprendizagem.

(iii) Condutância da Pele: interpreta estados de relevância emocional através da superfície da pele (produção de suor).

(iv) Emoções através do eletrocardiograma: interpreta batimentos cardíacos para detectar relevâncias emocionais.

(v) Estresse através do eletrocardiograma: interpreta batimentos cardíacos para detectar estresse.

(vi) Psicologia através dos Movimentos, Postura e Expressões Faciais: utiliza a visão computacional para detectar e aplicar técnicas da psicologia comportamental.

(vii) Testes psicométricos: faz testes para colher resultados a respeito da personalidade (auditivo, visual e cinestésico).

(vii) Alimentação: verifica a nutrição do usuário para detectar a presença de alimentos que melhoram a cognição.

(iv) Genética: verificar características próprias dos usuários para fazer adaptações individuais.

A Neurometria está sendo usada no modelo atual apresentado, no entanto há mais abordagens que podem ser obtidas a partir de outros dados das ondas cerebrais.

V. CONCLUSÃO

O artigo descreve um modelo adaptativo de um Framework para Ensino simples, que faz uso apenas da tecnologia de eletroencefalograma e das respostas do usuário à interface para modificar o comportamento dos conteúdos da interface de software.

O uso da tecnologia EEG no modelo adaptativo possui alguns benefícios como não ser invasivo, ser portátil, possibilitar o descobrimento de novos padrões comportamentais e aferir dados em tempo real. No entanto, essa tecnologia possui muitos dados de entradas, representação através de números complexos, é suscetível a ruídos de sinais eletrônicos e as medições podem ser modificadas por outras atividades elétricas geradas pelo movimento do corpo. A combinação da abordagem EEG (psicofisiológica) com o nível de acertos do usuário melhora e/ou resolve muitos dos problemas apresentados, aumentando a acurácia do sistema adaptativo.

O uso da tecnologia EEG combinada ao comportamento das respostas do usuário à interface num dispositivo adaptativo melhora com o uso de outras abordagens na aplicação, no ambiente e no sensoriamento do corpo humano. Os próximos passos dos autores serão a viabilização na implementação de outras abordagens ao modelo.

AGRADECIMENTOS

Agradecemos o apoio financeiro do Programa de Educação Tutorial / Ministério da Educação.

REFERÊNCIAS

[1] NETO, J. José. Um levantamento da evolução da adaptatividade e da tecnologia adaptativa. Revista IEEE América Latina, v. 5, n. 7, p. 1548-0992, 2007.

[2] HOU, Ming; BANBURY, Simon; BURNS, Catherine. Intelligent Adaptive Systems: An Interaction-centered Design Perspective. CRC Press, 2014.

[3] DIZERÓ, Wagner José. Formalismos adaptativos aplicados na modelagem de softwares educacionais. Tese de Doutorado. Universidade de São Paulo.

[4] NETO, J. José. Contribuições à metodologia de construção de compiladores. São Paulo: Tese de Livre Docência, USP, 1993.

[5] JUNIOR, A. A. de Castro. Adapbot desenvolvimento de sistemas adaptativos para o controle de processos robóticos. Chamada FUNDECT/CNPq No 08/2009 - CPPP, Outubro de 2009.

[6] Instituto Crescer. Crescer em rede. http://institutocrescer.org.br/cresceremrede/, 2015.

[7] Neurosky, Site de informações do Mindwave Mobile. http://store.neurosky.com/products/mindwave-mobile, 2015.

[8] EDUCATION, Emma Eccles Jones. Connecting learning objects to instructional design theory: A definition, a metaphor, and a taxonomy.

[9] RAMOS, Marcus Vinícius Midena; NETO, João José; VEJA, Ítalo Santiago. Linguagens Formais: teoria, modelagem e implementação. Bookman, 2009.

[10] ROCHA, Ricardo L. Azevedo; NETO, João José. Autômato adaptativo, limites e complexidade em comparação com máquina de Turing. In: Proceedings of the. 2000. p. 33. ROCHA, Ricardo L. Azevedo; NETO, João José. Autômato adaptativo, limites e complexidade em comparação com máquina de Turing. In: Proceedings of the second Congress of Logic Applied to Technology – LAPTEC. 2000. p. 33-48, 2001.

[11] NETO, João José. Adaptive automata for context-dependent languages. ACM Sigplan Notices, v. 29, n. 9, p. 115-124, 1994.

[12] NETO, J. José. Um levantamento da evolução da adaptatividade e da tecnologia adaptativa. Revista IEEE América Latina, v. 5, n. 7, p. 1548-0992, 2007.

[13] DOS SANTOS, José Maria Novaes. Um formalismo adaptativo com mecanismo de sincronização para aplicações concorrentes. 1997. Tese de Doutorado. Basseto, B. A. Um sistema de composição musical automatizada, baseado em gramáticas sensíveis ao contexto, implementado com formalismos adaptativos. Dissertação de Mestrado, Escola Politécnica da USP, São Paulo, 2000

[14] BASSETO, Bruno Arantes. Um sistema de composição musical automatizada, baseado em gramáticas sensíveis ao contexto, implementado com formalismos adaptativos. 2000. Tese de Doutorado. Dissertação (mestrado), EPUSP, São Paulo.

[15] IWAI, Margarete Keiko; NETO, João José. Introdução às Gramáticas Adaptativas. Boletim Técnico, PCS-POLI-USP, São Paulo, Brasil, 2000.

[16] NETO, João José. Adaptive rule-driven devices-general formulation and case study. In: Implementation and Application of Automata. Springer Berlin Heidelberg, 2002. p. 234-250.

[17] PISTORI, Hemerson. Tecnologia adaptativa em engenharia de computação: Estado da arte e aplicações. Universidade de São Paulo (USP), São Paulo, 2003.

[18] QUEIROZ, Diego. Uma definição simplificada para o estudo das propriedades dos autômatos finitos adaptativos. Quarto Workshop de Tecnologia Adaptativa - WTA 2010. EPUSP, 2010.

[19] CAMOLESI, A. R. Uma Metaferramenta de Apoio à Tecnologia Adaptativa. Revista IEEE.

[20] PELEGRINI, Eder José. Códigos adaptativos e linguagem para programação adaptativa: conceitos e tecnologia. 2009. Tese de Doutorado. Universidade de São Paulo.

[21] FREITAS, A. V. D., & Neto, J. J. (2006). Adaptive languages and a new programming style. WSEAS International Conference.

Alessandro Murta Baldi, cursando graduação em Ciência da Computação na Universidade Federal de Mato Grosso do Sul – Câmpus Ponta Porã (UFMS - CPPP). Bolsista integrante do grupo PET – Fronteira (Programa de Educação Tutorial - Ministério da Educação). Desenvolve pesquisa nas áreas de Sistemas Embarcados, Tecnologias Adaptativas, Tecnologias Educacionais e Informática Médica.
Curriculum Lattes: http://lattes.cnpq.br/7548137559386546.

Amaury Antônio de Castro Junior, obteve o título de bacharel e mestre em Ciência da Computação pela Universidade Federal de Mato Grosso do Sul (UFMS) e de doutor em Engenharia de Computação pela Escola Politécnica da Universidade de São Paulo (USP). É professor adjunto da Universidade Federal de Mato Grosso do Sul (UFMS), lotado no Campus de Ponta Porã (CPPP), atuando como docente na graduação e junto ao Programa de Mestrado Profissional em Computação Aplicada à Agricultura e Pecuária da Faculdade de Computação (FACOM) da UFMS. Bolsista do Programa de Educação Tutorial - PET/SESu/MEC, atuando como Tutor do Grupo PET/Fronteira; coordenador do Programa de Extensão NERDS da Fronteira (Núcleo Educacional de Robótica e Desenvolvimento de Software) e Secretário Regional da SBC (Sociedade Brasileira de Computação) no estado de Mato Grosso do Sul. Tem experiência em gestão acadêmica no ensino superior, adquirida através da atuação na coordenação de diversos cursos de graduação e na direção de câmpus avançado. Tem interesse pelos seguintes temas de pesquisa na área de Ciência da Computação: Teoria da Computação,

Tecnologia Adaptativa, Inteligência Computacional, Linguagens de Programação, Tecnologias Educacionais e Robótica.
Curriculum Lattes: http://lattes.cnpq.br/6311632162541654.

Elisângela Silva da Cunha Rodrigues, Doutora em Ciências, com ênfase em Engenharia de Computação e Sistemas Digitais, pela Escola Politécnica da Universidade de São Paulo (2012), Mestre em Informática pela Universidade Federal de Campina Grande (2002) e Bacharel em Informática pela Universidade Federal de Pelotas (2000). Professora Adjunto I da Universidade Federal de Mato Grosso do Sul, Campus de Ponta Porã. Atua na área de Ciência da Computação, com ênfase em Informática para Biodiversidade, Computabilidade e Modelos de Computação. Dentre os seus interesses de pesquisa, destacam-se: Entropia Máxima, Princípio MDL, Tecnologia Adaptativa, Algoritmos de Modelagem de Nicho Ecológico, Redes Neurais Artificiais e Tecnologia em Petróleo e Gás Natural.
Curriculum Lattes: http://lattes.cnpq.br/9860692793246856.

Fabrício Augusto Rodrigues, 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 da Inteligência Artificial e da Informática para a 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.
Curriculum Lattes: http://lattes.cnpq.br/4868911296156309.

Componente de ajuste de dificultad en un videojuego utilizando tablas de decisión adaptativas (9 de octubre de 2015)

K. Barrios y C. Zapata, *Member IEEE*

Abstract— While a lot of dynamic difficulty components in videogames act based upon the results of players actions and not the actions themselves, in this paper we try a different approach using information from every player action in every different situation. This component learns the player's reaction to every possible situation of: environment, enemy placement, enemy actions, and other variables, and use that to predict their next action so the enemies can act according to it. This is not a switch of difficulty settings, but a definition of the best enemy behavior that changes in real time. We propose using Adaptive Decision Tables to control the enemies' actions so their behavior can change according to the situation.

Keywords — adaptive table, videogames.

I. Introducción

Durante la creación de videojuegos (definidos por Ralph Baer[1] como "aparatos [...] con el propósito de jugar, entrenar con simulaciones y participar de otras actividades") se considera que los jugadores son diferentes y, por lo tanto, no pueden disfrutarlos de la misma forma unos y otros. Según Csikszentmihalyi [2], debe haber un balance entre dificultad y habilidad del jugador para que éste pueda divertirse con el juego; mientras que Hunicke [3] dice que los videojuegos son aburridos cuando son muy fáciles y frustrantes cuando son muy difíciles, casos en los que desaparece su fin: entretener.

Es por esto que existen diferentes formas para ajustar el reto a cada jugador, como mostrar el puntaje para que el jugador tenga un objetivo al tratar de mejorarlo, o dar la libertad de elegir la dificultad, que cambia algunas características predefinidas que aumentan o disminuyen el reto.

Una forma usada últimamente para que cada jugador tenga un reto adecuado es la modificación automática de la dificultad según la forma en que se juega. Muhammad Iftekher Chowdhury [4] menciona que "si el nivel de dificultad ajustado dinámicamente en un videojuego se iguala apropiadamente a la habilidad del jugador, entonces no solo atraerá a jugadores de diferentes demografías, sino que también hará que el jugador juegue el videojuego repetidas veces sin aburrirse".

Se identifica como un problema cambiar diferentes características de un videojuego de manera eficiente al detectar las habilidades del jugador. Y con la particularidad de que lo que se cambie sea el comportamiento de cada enemigo o personaje que no maneje el jugador (NPC o "non-player character").

Policarpo y Urbano[5] analizan diferentes reglas para el comportamiento de los NPC y concluyeron que sí es posible que una IA dinámica aprenda tácticas que se aprovechen de las debilidades en oponentes que usen IA estática. Huang [6] demuestra que, para un jugador con una determinada habilidad, el ajuste de dificultad dinámica en Inteligencia Computacional tanto en tiempo real como en ANN (Redes Neuronales) puede generar un oponente con un reto apropiado para satisfacer al jugador.

El cambiar el comportamiento de cada NPC es el problema que se usará para este proyecto. Esto implica tomar la información de las acciones o habilidades del jugador y de qué tan bueno es cumpliendo el objetivo del juego, para usar dicha información en la modificación de comportamiento. No importará si el jugador es bueno o es malo, el componente desarrollado identificará su forma de jugar, sus decisiones en cada situación y las acciones involuntarias que realice, aunque sea por razones imperceptibles debido al *factor humano*, y modificará el comportamiento de los oponentes para que se adecúen a dicho jugador, siendo completamente diferentes para cada persona.

A. Problema Seleccionado:

¿Cómo se puede hacer un algoritmo que modifique el comportamiento de los oponentes basándose en información de cada acción del jugador?

El problema se desarrollaría definiendo una forma para tomar la información del jugador y aplicarla al comportamiento de los enemigos.

Se buscaría analizar de forma profunda cada acción de los jugadores y modificar el comportamiento de los NPC según sus costumbres, habilidades y fallos. Cada movimiento que el jugador hace debería afectar directamente el comportamiento de los NPC.

II. Objetivos del Trabajo

El objetivo de este trabajo es implementar un componente de ajuste de dificultad que altere el comportamiento de los

oponentes en respuesta a los patrones de comportamiento del jugador utilizando tablas de decisión adaptativa.

A. *RESULTADOS ESPERADOS*

Se busca diseñar un componente de aprendizaje de las acciones del jugador lo que permitirá obtener:

1. Componente que identifica situación actual: estado actual del personaje (sobre el piso, saltando, cayendo), la ubicación respecto al entorno (cercanía de paredes, techo, piso, abismos respecto al jugador) y la situación de los enemigos que rodean al jugador (cantidad, posiciones y movimientos).
2. Componente que actualiza la lista de posibilidades de cada acción (saltar, correr, atacar, etc.) que el jugador haría en cada situación. Se debe tener también una lista de todas las situaciones posibles.

Una vez identificadas las acciones del jugador se diseñará un componente que controle el comportamiento de los NPC con tablas de decisión adaptativas para que realicen ataques contra el jugador según las probabilidades de su próximo movimiento. En consecuencia se obtendrá:

3. Reglas para la tabla de decisión adaptativa (reglas adaptativas y no adaptativas).
4. Componente de priorización de reglas (para evitar sobrecarga de proceso).

Para validar que lo obtenido logra el objetivo buscado se implementará el componente de aprendizaje y los patrones de comportamiento en un videojuego 2D de acción de vista lateral, para lo cual se realizarán las siguientes acciones:

5. Desarrollo de un videojuego de acción de vista lateral. Para implementar los componentes.
6. Implementación del componente de aprendizaje y de la Tabla de Decisión adaptativa. Se implementan los primeros cuatro objetivos específicos en el videojuego desarrollado en el quinto resultado específico.

III. MARCO CONCEPTUAL

A. *Videojuego*

Según la real academia española es “un dispositivo electrónico que permite, mediante mandos apropiados, simular juegos en las pantallas de un televisor o de un ordenador” [7]. Hunicke [3] dice que los videojuegos son aburridos cuando son muy fáciles y frustrantes cuando son muy difíciles. Cuando una o más personas controlan parámetros o personajes en un videojuego, se les considera jugadores. El jugador es quien “disfrutará” del videojuego al jugarlo.

El género de videojuego en el que se implementará la alternativa de solución es un juego de acción en 2D. Lo que vendría a ser un personaje que se mueve en un plano de dos dimensiones donde puede caminar, saltar y atacar, donde se encontrará con oponentes (NPC) con las mismas características y se dará en un escenario con superficies a diferentes niveles de altura

B. *NPC*

Van Horn [8] indica que NPC son las siglas de “non-player character” o “personaje que no es jugador”. Es un personaje de un videojuego con acciones que no dependen del control directo del jugador. Policarpo y Urbano [5] crean comportamientos basados en reglas para los NPC, estos muchas veces siguen acciones predefinidas o se usa algoritmos de inteligencia artificial para determinar su proceder en las diferentes situaciones del juego. Chin Hyong Tan [9] dice que los videojuegos involucran interacción del jugador tanto con el entorno como con los NPC.

El término NPC, usualmente, no se limita a oponentes, pero en este proyecto de fin de carrera así se les denominará.

C. *Dificultad*

La definición de la real academia española dice que dificultad es “embarazo, inconveniente, oposición o contrariedad que impide conseguir, ejecutar o entender bien algo y pronto” [7]. Csikszentmihalyi [2] dice que debe haber un balance entre dificultad y habilidad del jugador para que éste pueda disfrutar del juego.

Muhammad Iftekher Chowdhury [4] indica que la Dificultad dinámica automática es la técnica automática por la cual se cambia la elevación de dificultad para un videojuego en tiempo real (sin dejar de jugar) para que iguale las habilidades del jugador. Agrega también que implementarla es caro y consume mucho tiempo, por lo que su utilidad está limitada. Pederassi Lomba de Araujo y Deijó [10] indican que el uso de Dificultad dinámica mantiene al jugador constantemente motivado.

D. *Unity*

Unity es un Game Engine (de uso similar a un framework) que se caracteriza por tener muchas herramientas de fácil uso para la creación de videojuegos, además de que se puede usar en una gran cantidad de plataformas [11].

E. *Probabilidad de acción futura.*

Se determina la acción futura del jugador según probabilidades que se alteran con cada nueva acción que este efectúa en cada situación diferente.

F. *Tablas de Decisión*

Definen un comportamiento a tomar según una lista de reglas. Cada regla tiene ciertas condiciones que dan un resultado diferente

G. *Tablas de Decisión Adaptativas*

Las Tablas de Decisión Adaptativas tienen reglas adicionales en la capa adaptativa, que indican el cambio de los resultados de las reglas de la capa anterior. Neto dice que esta es la mejor alternativa entre las tecnologías adaptativas para usar en el problema propuesto[12].

En este proyecto desarrollará un método de ajuste del comportamiento de los NPC de forma dinámica en un videojuego de acción con vista lateral. Videojuego y dificultad son los conceptos básicos que se usan para el entendimiento del componente. NPC son los personajes de los cuales el componente modificará su comportamiento. Dificultad ajustada automáticamente es el concepto que refleja de qué forma el algoritmo modificará el comportamiento de los NPC.

IV. ESTADO DEL ARTE

Desde hace años se han definido diversas formas de aplicar el ajuste de dificultad en los videojuegos. Desde formas simples

como alterar el desempeño de los NPC según los logros del jugador, hasta formas más complejas como delimitar el rango de habilidad del jugador según varios factores y con eso cambiar parámetros del juego.
La revisión del estado del arte busca analizar cómo son en la actualidad las diferentes propuestas de ajustes de dificultad en los videojuegos, tanto las usadas en juegos comerciales, como las que no han sido aplicadas.

A. Método usado en la revisión del estado del arte

Se ha usado el método tradicional para la revisión del estado del arte. Se realizaron búsquedas en las bases de datos de artículos IEEE y Research Gate, para analizar diferentes alternativas de solución hay para el mismo problema. La búsqueda se realizó la primera semana de abril del 2015.

B. Formas de usar ajustes de dificultad:

1) Personajes que aprenden la mejor estrategia para cada situación gracias a comportamientos de reglas.
Policarpo y Urbano [5] definieron comportamientos definidos por reglas que logran que la IA de personajes aprenda la mejor estrategia para cada situación. Cada oponente tiene una lista de reglas, de las cuales se coloca las de mayor "peso" en el script de su comportamiento. Con cada interacción con el jugador se le aumenta el peso a las reglas que, al ejecutarse, tuvieron un resultado exitoso.

2) Algoritmos evolucionarios de multiple objetivos.
Van Hoorn [8] definió algoritmos evolucionarios con multiobjetivos relacionados a mejorar el desempeño de juego y lograr mayor similitud con el comportamiento humano. Indica también que hay varios usos para la imitación del jugador en diferentes tipos de videojuegos y que hay la necesidad de encontrar modelos que sean robustos y también creíbles.

3) Graduación de dificultad de IA mediante algoritmos de RL y EC.
Chin Hiong Tan [9] usa Reinforcement Learning y Computación Evolutiva para mejorar la satisfacción del jugador al aumentar la dificultad de la IA del juego mientras se juega. Los efectos de cambiar las escalas de aprendizaje (Learning) y mutación se examinan y se propone una regla para los parámetros. Se presenta dos algoritmos para mejorar la satisfacción del jugador, un Adaptive Uni-chromosome y un Adaptive Duo-chromosome. Se varió la tasa de aprendizaje y de mutación de ambos algoritmos para lograr una regla general. Los algoritmos propuestos demuestran la capacidad de igualar a sus oponentes en puntajes promedio y porcentaje de victorias. Ambos algoritmos pueden generalizar una variedad de oponentes.

4) Modificación masiva del comportamiento de cientos de personajes en simultáneo.
Maggiore [13] define planes (conjunto de acciones consecutivas futuras) y un algoritmo de búsqueda de caminos en grafos (cada nodo es una acción y se busca el mejor conjunto de acciones, o mejor plan), con lo que modifica el comportamiento en tiempo real de cientos de personajes, de forma diferente para cada uno, sin requerir información previa. Se usó una técnica de planificación por capas para crear NPCs que no solo reaccionen a estímulos externos, logrando un mundo más creible. Con un encadenamiento de acciones en un árbol de nodos se forman planes diferentes, junto a heurísticas para determinar los planes parciales menos efectivos e ignorarlos, además para memorizar y reutilizar planes pasados que hayan resultado efectivos. También toma en cuenta el tiempo que tomará, para que no empiece muy tarde. Es un planificador adaptativo, ya que las acciones que toma y los escenarios no están "hardcodeados".

5) Ajuste de dificultad dinámica al alterar comportamiento de oponentes con CI, MCTS y UCT.
Huang [9] define que se puede lograr un ajuste en la dificultad dinámica al cambiar el reto que presentan los oponentes mediante Inteligencia Computacional, al incluir el Árbol de Búsqueda de Monte Carlo (MCTS) y la Upper Confidence bound for Trees (UCT).

6) Ajuste de dificultad según desempeño y preguntas.
Holtkamp [14] hace que la dificultad se ajuste según características del jugador definidas por las respuestas a nueve preguntas y por su desempeño al jugar.

El estado del arte muestra el éxito y problemas que han tenido diferentes alternativas de solución similares a la presentada en este proyecto en el que se presenta una alternativa para solucionar los problemas de toma de información y modificación de comportamiento encontrados en la revisión del estado del arte, al enfocarse detalladamente en que cada movimiento del jugador defina un ajuste en el comportamiento de los NPC; ya que, al tomar en cuenta cada una de las acciones del jugador, se obtiene datos de una forma tan detallada que refleja sus costumbres al jugar.

V. Diseño e implementación de La Solución

A. Aprendizaje

En esta sección se explica la etapa de aprendizaje del componente, que comprende el levantamiento de la información y la forma en la que se utiliza. Además se define cómo esta será estructurada.
Existen varias formas de tomar los datos del jugador, los cuales son esenciales debido a que es la información que se usará. Lo que se busca es un formato o método de extraer los datos que no afecte el desempeño del juego (por ejemplo, que por sobrecarga de proceso al tomar los datos, el juego no sea fluido), pero que, aun así, obtenga la información necesaria para la metodología de ajuste de dificultad elegida.
Por ejemplo, Holtkamp [14] agregó a la toma de datos de desempeño durante el juego una serie de preguntas al jugador, que definían algunos parámetros de su personalidad; Policarpo y Urbano [5] analizan cuál de las múltiples reglas del comportamiento de cada NPC, en este caso un oponente, es más efectiva al enfrentarse al jugador y le dan prioridades diferentes en el script (archivo de procesamiento por lotes) dinámico que determina su comportamiento.
El problema radica en que estos métodos toman datos superficiales del jugador. No detallan lo que busca hacer el jugador, sino lo que obtienen, reduciendo así la variedad de la información tomada. Esto es porque varios jugadores diferentes podrían tener el mismo resultado jugando diferente y teniendo diferentes objetivos (un jugador diestro y uno que se aprovecha de un error de programación podrían conseguir ambos el puntaje máximo).

El levantamiento de información se hará analizando la forma en la que el jugador juega, identificando primero la situación en la que se encuentra y después la acción que ejecuta en esta. El aprendizaje se ejecuta de forma continua y, tras varias sesiones de juego, se define la acción más usada para cada situación, lo que a su vez son probabilidades para predicciones futuras, siendo este el punto de partida para la modificación del comportamiento de los enemigos.

1) Identificación de la Situación

Tiene que identificarse la situación actual según una serie de características del personaje, los NPC y el entorno. Esto servirá para relacionar cada acción del jugador a una situación particular.

2) Flags:

Para las características se usarán flags (definidos en la Tabla 1), que cambiarán según se está jugando. Cada vez que el jugador haga una acción se revisan los flags, se determina que esa acción la hace el jugador para ese conjunto de flags y se procede a modificar las probabilidades de cada acción para cada situación. La Situación está determinada por los valores del conjunto de flags.

Tabla 1. Flags que determinan situación

#	Flag
1	Personaje está en el piso (no en el aire).
2	Personaje está cerca de pared izquierda.
3	Personaje está cerca de pared derecha.
4	Personaje está caminando a la izquierda.
5	Personaje está caminando a la derecha.
6	El techo está cerca al piso.
7	Hay caída a la izquierda.
8	Hay caída a la derecha.
9	Hay un NPC cercano al personaje a la izquierda.
10	Hay dos o más NPC cercanos al personaje a la izquierda.
11	Hay un NPC cercano al personaje a la derecha.
12	Hay dos o más NPC cercanos al personaje a la derecha.

3) Acciones:

En cada Situación, el jugador puede hacer cualquiera de las acciones que desee del personaje. En la Tabla 6 vemos las acciones que puede realizar.

Tabla 2. Acciones que puede hacer el jugador

#	Acción
1	Caminar a la izquierda
2	Caminar a la derecha
3	Saltar
4	Saltar a la izquierda
5	Saltar a la derecha
6	Ataque 1 (hacia adelante)
7	Ataque 2 (hacia arriba)
8	Ataque 3 (proyectil a distancia)

4) Modificación de probabilidades

Para cada una de las Situaciones, hay una lista ordenada de todas las acciones del jugador. Cada vez que el jugador hace una acción en determinada Situación, dicha acción sube una posición en la lista correspondiente a la Situación en que se ejecutó. Se toma la acción en primera posición de cada lista como la acción que tiene más probabilidad que ejecute el jugador en cada situación.

Para cada situación, el jugador puede hacer cualquiera de las acciones, pero la que está en primer lugar en la lista de acciones probables para dicha situación es la que tiene mayor probabilidad de realización.

En la Tabla 7 podemos ver cómo cambia el orden de la lista de probabilidad de acciones para una misma situación.

Tabla 3. Cambio de prioridad de acciones para cada Situación

Situación (set de Flags)	Acciones (de mayor a menor prioridad)							
A (flags: 1-3-4)	1	2	3	4	5	6	7	8
Para la situación A hace la acción 3								
A (flags: 1-3-4)	1	3	2	4	5	6	7	8
Para la situación A hace la acción 4								
A (flags: 1-3-4)	1	3	4	2	5	6	7	8
Para la situación A hace la acción 3								
A (flags: 1-3-4)	3	1	4	2	5	6	7	8

B. Tabla de Decisión Adaptativa

Los métodos de ajuste de dificultad usan información para definir el comportamiento de los enemigos. La información puede obtenerse del jugador, de la efectividad de acciones pasadas de los enemigos, deduciendo la mejor elección para la situación actual, etc. Y puede ser procesada de diferentes formas, ya sea usando algún algoritmo de inteligencia artificial, búsqueda de caminos en árbol de nodos, etc.

Chin Hiong Tan [9] usa Computación Evolutiva para, mediante ensayo y error, definir una regla general con la que modifica los parámetros de los algoritmos. Maggiore [13] usa una búsqueda en un árbol de nodos para encontrar la mejor secuencia de acciones de los NPC.

El problema radica en que las diversas formas de procesar la información y aplicarla en el comportamiento de enemigos no tiene una "mejor forma" de hacerlo. Debe hacerse de una forma que sea óptima dependiendo de: el origen de la información usada, cómo se ha procesado y el género del videojuego; considerando en qué características se quiere ser más eficiente como el requerimiento de proceso, la velocidad de aplicación, máximo de enemigos simultáneos, frecuencia de la toma de datos, etc.

Se decidió usar Tablas de Decisión Adaptativas en vez que otras metodologías porque, según Neto[12], permiten definir reglas (ideal para el comportamiento de los NPC) y cambiarlas constantemente (ideal para la modificación de su comportamiento). Se evaluaron varias alternativas pero se optó por Tablas de Decisión Adaptativas.

Con las Tablas de Decisión Adaptativas se logra que el comportamiento de los NPC cambie mientras se juega continuamente.

1) Reglas para la tabla de decisión adaptativa

Hay dos listas de Reglas de la Tabla de Decisión Adaptativa: las reglas de la capa no adaptativa y las reglas de la capa

adaptativa. Las reglas de la capa no adaptativa son una lista de estímulos y sus respectivas respuestas. Las reglas de la capa adaptativa son otra lista de estímulos y sus respuestas, pero estas respuestas son todas las combinaciones de estímulo y respuesta de las reglas de la capa no adaptativa, y los estímulos señalan que estimulo con respuesta *actualizar* en la capa no adaptativa.

Definiremos a la cantidad de Situaciones como N y la cantidad de acciones de los NPC como M. Por esto la cantidad de reglas de la capa no adaptativa es N y la cantidad de reglas de la capa adaptativa es NxM. Además, un Contexto es la suma de una Situación y la Acción del jugador más alta en la lista de posibilidades para dicha Situación.

En la Tabla 8 puede verse un ejemplo de las reglas de la capa adaptativa y de las reglas de la capa no adaptativa.

Tabla 4. Ejemplo de reglas de Tabla de Decisión Adaptativa

Capa no adaptativa			Capa Adaptativa	
TAMAÑO = N			TAMAÑO = M X N	
Condicion A	Accion 1		Condicion A	Accion 1
Condicion B	Accion 3		Condicion A	Accion 2
Condicion C	Accion 2		Condicion A	Accion 3
Condicion D	Accion 3		Condicion B	Accion 1
			Condicion B	Accion 2
			Condicion B	Accion 3
			Condicion C	Accion 1
			Condicion C	Accion 2
			Condicion C	Accion 3
			Condicion D	Accion 1
			Condicion D	Accion 2
			Condicion D	Accion 3

Cada regla de la capa no adaptativa corresponde a un Contexto y la acción correspondiente que debe ejecutar el NPC. Hay un Contexto por cada Situación.

Las reglas de la capa adaptativa son la combinación de todos los contextos con todas las acciones que pueden hacer los NPC. Las reglas de la capa determinan el cambio de acción del NPC correspondiente a cada Contexto en las reglas de la capa no adaptativa.

2) Componente de priorización de reglas

Debido a la cantidad de Flags, acciones del jugador y de los NPC, se puede ver que tienen un gran tamaño la cantidad de Situaciones, Contextos, Reglas no adaptativas y, en una considerable mayor medida, Reglas adaptativas. Esto podría hacer que el rendimiento de la alternativa de solución para este problema disminuya por el alto y constante requerimiento de proceso.

Para disminuir esta carga de proceso, se ha tomado en cuenta, únicamente, a las reglas correspondientes a una cantidad reducida de Contextos. Los Contextos elegidos también pueden cambiar, según los que se hayan dado más veces en las últimas situaciones. Pero, este cambio de contextos elegidos no mantendrá la frecuencia de uso de la Tabla de Decisiones Adaptativa, porque sino la carga de proceso no disminuirá. Por ejemplo, si la Tabla de Decisiones Adaptativa es consultada con cada acción del personaje o de un NPC, el cambio de Contextos elegidos se podría hacer con un hilo separado cada medio minuto, o cada vez que se gana o pierde en un nivel.

VI. Conclusiones

Mantener el reto en el jugador es una de las formas de hacer interesante un juego. Este factor se puede obtener de diferentes maneras: una historia atractiva, objetivos complejos, distintos caminos para lograr los mismos objetivos, entre otros.

Este trabajo no sólo gira alrededor de a IA que se debe desarrollar para contar con enemigos que reacciones a las diversas acciones del jugador. Es muy importante contar con una adecuada toma de información sobre las acciones del jugador y estructurarlas de tal forma que afecten mínimamente el desempeño del juego.

El uso de tablas de decisión adaptativas se ajustan muy bien al proyecto pero se deben tomar ciertas decisiones sobre cómo aplicarlas para reducir el impacto en el jugador y el juego.

Dado que este proyecto aún no ha concluido no podemos dar cifras objetivas sobre el trabajo.

VII. TRABAJOS FUTUROS

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

1) Otros géneros

Adaptar la alternativa de solución para videojuegos de otros géneros, siendo los más fáciles de adaptar los videojuegos de lucha y los videojuegos de acción 3D.

2) Estandarizar dificultad

Analizar la probabilidad de victoria para cada acción del jugador en cada situación, y que se modifiquen diversas características del videojuego para mantener constante esta probabilidad en todas las situaciones, así se lograría mantener una dificultad estándar que depende del grado de habilidad del jugador.

3) Machine Learning en predicciones

Utilizar Machine Learning para las predicciones. En vez de que cada nueva acción del jugador cambie la lista de posibilidades, se acumularía la información y se utilizaría Machine Learning para la predicción del comportamiento futuro del jugador, debido a que son grandes cantidades de información las que se habrán tomado de todas las acciones previas, esa es la utilidad de Machine Learning según Kohavi [15]. La metodología a usar sería Feature Selection, donde se descartan las características que menor importancia tienen; se usaría para procesar solo la información relevante [16].

4) Reemplazos a autómatas adaptativos

Usar metodologías diferentes para la modificación de comportamiento de NPC: Autómatas no adaptativos, Tablas adaptativas de decisiones (modificando secciones de la tabla), Redes neuronales (con un set de datos iniciales para que haya una dificultad estándar y sobre eso modificarlos personalmente con cada jugador con métodos de aprendizaje)

o Sistemas Expertos (tomar el juicio de un experto como base para determinar la dificultad ideal para cada jugador).

VIII. Referencias

[1] R. H. Baer, W. T. Rusch y W. L. Harrison, «TELEVISION GAMING APPARATUS AND METHOD». USA/NH Patente United States Patent 3659285, 25 4 1972.

[2] M. Csikszentmihalyi, Flow : the psychology of optimal experience, New York: Harper & Row, 1990.

[3] R. Hunicke y V. Chapman, «AI for Dynamic Difficulty Adjustment in Games,» Northwestern University, Computer Science Department, Evanston, IL, 2004.

[4] M. I. Chowdhury y M. Katchabaw, «Bringing Auto Dynamic Difficulty to Commercial Games: A Reusable Design Pattern Based Approach,» de *CGAMES 2013, The 18th International Conference on Computer Games*, London, Ontario, Canada, 2013.

[5] D. Policarpo y P. Urbano, «Dynamic Scripting Applied to a First-Person Shooter,» Laboratório de Modelação de Agentes, FCUL, Lisboa, Portugal, 2010.

[6] W. Huang, S. He, D. Chang y Y. Hao, «Dynamic Difficulty Adjustment Realization Based on Adaptive Neuro-controlled Game Opponent,» de *Third International Workshop on Advanced Computational Intelligence*, Suzhou, Jiangsu, China, 2010.

[7] R. A. Española, Diccionario de la lengua española, Madrid, 2012.

[8] N. van Hoorn, J. Togelius, D. Wierstra y J. Schmidhuber, «Robust player imitation using multiobjective evolution,» Dalle Molle Institute for Artificial Intelligence (IDSIA), Manno-Lugano, Switzerland, 2009.

[9] C. H. Tan, K. C. Tan y A. Tay, «Dynamic Game Difficulty Scaling Using Adaptive Behavior-Based AI,» *Computational Intelligence and AI in Games, IEEE Transactions on*, vol. 3, nº 4, pp. 289 - 301, 2011.

[10] B. B. Pederassi Lomba de Araujo y B. Feijó, «Evaluating dynamic difficulty adaptivity in shoot'em up games,» de *SBC - Proceedings of SBGames 2013*, Río de Janeiro, 2013.

[11] «Unity - Game Engine,» [En línea]. Available: unity3d.com.

[12] J. J. Neto, Interviewee, *Uso de Tecnologías Adaptativas para el comportamiento dinámico de oponentes en videojuegos.* [Entrevista]. 7 9 2015.

[13] G. Maggiore, C. Santos, D. Dini, F. Peters, H. Bouwknegt y P. Spronck, «LGOAP: adaptive layered planning for real-time videogames,» de *Computational Intelligence in Games (CIG), 2013 IEEE Conference on*, Niagara Falls, ON, 2013.

[14] W. Holtkamp, P. Trevino, C. Yun y O. Johnson, «Dynamic Difficulty with Personality Influences,» de *GLS 8.0 (Games+Learning+Society)*, Madison, WI, 2012.

[15] R. Kohavi y R. Provost, «Glossary of Terms Special Issue on Applications of Machine Learning and the Knowledge Discovery Process,» Kluwer Academic Publishers, Boston, Manufactured in The Netherlands, 1998. [En línea]. Available: http://ai.stanford.edu/~ronnyk/glossary.html. [Último acceso: 08 06 2015].

[16] I. Guyon y A. Elisseeff, «An Introduction to Variable and Feature Selection,» Leslie Pack Kaelbling, 2003.

Um Mapeamento de Modelos Adaptativos para Dispositivos Adaptativos Guiados por Regras

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

***Abstract* — *Model Driven Engineering* (MDE) é uma abordagem para desenvolvimento de software em que modelos são artefatos fundamentais a serem construídos e manipulados, podendo gerar código-fonte automaticamente. Uma das possibilidades é a execução direta de modelos, e para isso caracterizam-se linguagens de modelagem interpretáveis. Recentemente, surgiu o conceito de modelos adaptativos, que são modelos executáveis com capacidade de passar por alterações estruturais durante sua execução através de autorresposta a estímulos externos. Este trabalho apresenta um mapeamento de tais modelos para uma formulação geral de dispositivos adaptativos guiados por regras encontrada na literatura. Como estratégia, um padrão de projeto de linguagens de modelagem executáveis é utilizado e estendido para incluir conceitos de adaptatividade. Algumas outras restrições são adicionadas aos modelos para que se possa identificar de forma geral todos os elementos necessários para mapeá-los. Com isso, modelos adaptativos são colocados no contexto geral da área de adaptatividade, aproveitando-se as teorias subjacentes já existentes.**

***Keywords*— Tecnologias adaptativas (*adaptive Technologies*), modelos executáveis (*executable models*) modelos adaptativos (*adaptive models*), *Model Driven Engineering*, *Set Based Meta Modeling*.**

I. INTRODUÇÃO

UM DISPOSITIVO adaptativo é composto por um dispositivo não-adaptativo subjacente e um mecanismo formado por funções adaptativas capaz de alterar o conjunto de regras que definem seu comportamento [3]. 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 das regras de comportamento do dispositivo não-adaptativo subjacente ou também do próprio mecanismo adaptativo.

Um autômato adaptativo, por exemplo, é um dispositivo adaptativo que estende o conceito de autômato finito incorporando a característica de desenvolver uma autorreconfiguração em resposta a um estímulo externo [4]. Criação de novos estados e transições em tempo de execução são exemplos de autorreconfiguração.

A tecnologia adaptativa traz novos mecanismos a serem utilizados em diferentes abordagens para resolver problemas, podendo ser aplicada em diversas áreas tais como automação e robótica [4], reconhecedores gramaticais [8] e até mesmo na área agrícola [9].

Um programa adaptativo pode ser entendido como uma especificação de uma sequência automodificável de instruções, que representa um código dinamicamente alterável [2]. Podem ser considerados dispositivos adaptativos em que o dispositivo não-adaptativo subjacente seria um programa estático, ou seja, um programa tradicional que não tem sua estrutura modificada ao longo de sua execução.

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), por exemplo, é uma linguagem de programação adaptativa de alto nível proposta por [2]. 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 [2] gera código para o ambiente de execução desenvolvido em [5], 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 [2], descrita em linguagem natural. Esta representação é formalizada em [7][10] com a utilização do *Set Based Meta Modeling* (SBMM) [20], um formalismo de metamodelagem. Técnicas para codificar programas adaptativos utilizando linguagens de programação orientadas a objetos usuais são apresentadas por [16].

Model Driven Engineering (MDE), um outro tópico de pesquisa em Engenharia de Software, é uma abordagem para desenvolvimento de software através da qual um sistema é construído pela transformação de modelos em código-fonte em alguma linguagem alvo. Essa transformação pode ocorrer em um número arbitrário de passos. Um modelo de um sistema é uma descrição ou especificação do mesmo considerando um determinado propósito, e, por isso, pode não representar todos os aspectos e propriedades do sistema por si só [6]. Porém, no cenário ideal, o código-fonte do sistema, em um estado pronto para ser compilado, seria completamente gerado a partir de transformações de seus modelos.

Com isso, os modelos de software servem não apenas como documentação, mas também como artefatos fundamentais para a construção do programa. Aumenta-se o nível de abstração no desenvolvimento e ganha-se na possibilidade de geração do mesmo sistema para diversas plataformas, uma vez que o

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

mesmo modelo poderia dar origem a código-fonte em mais de uma linguagem alvo, desde que se tenham as transformações necessárias disponíveis.

Usualmente um modelo pode ser representado por uma combinação de textos e desenhos, podendo estar descrito em linguagem de modelagem (ex: UML) ou em linguagem natural [1]. Entretanto, para que a MDE seja possível na prática, é necessário que os modelos de software estejam descritos em linguagens que permitam a leitura por programas sendo, portanto, a linguagem natural inadequada para este fim. Este problema é atacado pelos metamodelos.

Metamodelos são modelos de um tipo especial que descrevem a sintaxe abstrata de uma linguagem de modelagem. Existem, por exemplo, metamodelos que especificam a UML. Eles determinam as abstrações e conceitos que podem ser instanciados em seus modelos, bem como as restrições aplicáveis [11]. Em um diagrama de classes UML, como exemplo de restrição, não pode haver uma associação desconectada de classes. É o metamodelo que estabelece o conceito de associação e a restrição de que cada instância deve ter seus fins conectados em classes. O exemplo da Fig. 1 apresenta um metamodelo para máquinas de estado simples [11] e uma máquina de estado específica, que é um modelo conforme ao metamodelo. O metamodelo está denotado na linguagem *Meta Object Facility* (MOF). Restrições costumam ser definidas em alguma linguagem textual, tal como *Object Constraint Language* (OCL), e não são expressas no diagrama do metamodelo.

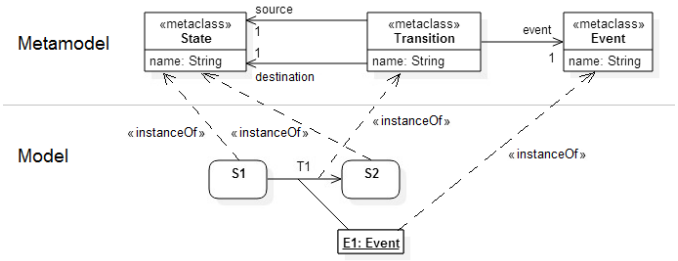


Fig. 1. Exemplo de um metamodelo para máquinas de estado e uma máquina de estado específica. Fonte: [11] (adaptado).

Alguns modelos de software possuem caráter descritivo, tais como diagramas de classes ou entidade-relacionamento, enquanto outros possuem caráter executável, tais como máquinas de estado ou redes de Petri. Linguagens de modelagem específicas de domínio, ou *domain specific modeling languages* (DSMLs) que permitem a criação de modelos executáveis são conhecidas como *executable DSMLs* (x-DSMLs). A execução de modelos conformes a essas linguagens pode ser alcançada não apenas por transformação de modelos em código-fonte, mas também por interpretação direta dos modelos. Linguagens com esta capacidade são denotadas pela sigla i-DSML e seus modelos são interpretados de acordo com uma semântica operacional processada por um motor (*engine*) de execução. Com esta abordagem, nenhuma transformação ocorre e o modelo é diretamente executável [12].

Como modelos executáveis são representações de programas de computador em mais alto nível de abstração, e alguns deles podem ser executados diretamente, então faz sentido considerar modelos adaptativos, uma vez que existem programas adaptativos. Modelos executáveis são estudados em diversos trabalhos tais como [13][14][15], entre outros. Porém, até onde nosso conhecimento permite afirmar, a adaptatividade de modelos executáveis parece ter sido estudada primeiramente no contexto da MDE por [12] e [18], sendo também abordada por [19], consistindo ainda um campo de pesquisa a ser explorado.

Este trabalho tem por objetivo utilizar as ideias apresentadas por [12] sobre caracterização de modelos adaptativos e o padrão de projeto de i-DSMLs apresentado por [15] para estabelecer um mapeamento de modelos adaptativos para a formulação geral de dispositivos adaptativos proposta por [3]. Assim, caracteriza-se esse tipo de modelo dentro dessa categoria de dispositivos, e permite-se aproveitar resultados existentes através dessa conexão. Ademais, teorias e ferramentas para metamodelagem têm evoluído de forma independente de modelos de computação, criando um *gap* cultural e técnico entre essas duas comunidades [17]. Busca-se com este trabalho contribuir com alguma redução neste *gap*.

A seção II apresenta a conceituação de modelos executáveis e modelos adaptativos encontrada na literatura. A seção III apresenta uma revisão da formulação geral para dispositivos adaptativos guiados por regras, enquanto a seção IV apresenta o mapeamento que é objetivo deste trabalho. A seção V discute os resultados e é seguida pela seção VI, que apresenta as conclusões. Por fim, a seção VII lista as referências bibliográficas.

II. Modelos Executáveis e Modelos Adaptativos

Esta seção apresenta a conceituação de modelos executáveis e modelos adaptativos encontrada na literatura. Ela é usada como base para o mapeamento apresentado mais adiante na seção IV.

A. Modelos Executáveis

Um modelo executável é um tipo especial de modelo que pode ser executado. Sua definição é discutida em diversos *papers* tais como [13][14][15], tendo sido estabelecido o seguinte consenso [12]:

- Alguns tipos de modelos são executáveis, enquanto outros não;
- Um motor (*engine*) é responsável por executar o modelo, tomando-o como entrada e efetuando o processamento necessário;
- Se o modelo armazena as informações necessárias para controle de sua própria execução através de um motor, então ele é dito autocontido (*self-contained*).

Um exemplo de modelo autocontido para máquinas de estado seria se o metamodelo previsse uma propriedade booleana *isCurrent* para a metaclasse que representa o estado e uma restrição determinando que só pode haver um estado corrente por vez. Em um modelo em execução, essa propriedade carregaria o valor *True* no estado corrente, dentro do modelo, e *False* para os outros. Se o gerenciamento do

estado corrente ocorre exclusivamente no motor, estando essa informação fora do modelo, então o modelo não é autocontido.

A estratégia dos modelos autocontidos parece poluir o metamodelo com detalhes não relevantes em tempo de projeto, mas eles podem ser justificados pelo fato de que a executabilidade é parte da natureza do modelo e, portanto, este pode ter seu nível de detalhes aumentado para atender este requisito [12]. Outra razão é que essa estratégia oferece a vantagem de que o estado corrente de execução pode ser serializado em um arquivo de saída, provendo um mecanismo de rastreabilidade da execução como sequências de modelos. Isso possibilitaria a oferta de operações úteis tais como *rollbacks*, depuração e teste.

Existe uma classificação geral que pode ajudar a identificar modelos executáveis [12]: a classificação produto ou processo. Modelos podem expressar produtos ou processos, independentemente do sistema sob consideração. O ponto central é que modelos de processos habilitam a executabilidade de seu conteúdo pois contêm conceitos relacionados a execução: ponto de início, ponto de fim, passo de evolução, etc. Por outro lado, modelos de produto são mais estruturais, tais como diagramas de classes UML, e eles não expressam diretamente comportamento ou interações.

Não consideramos essa classificação muito precisa para determinar modelos executáveis. Observemos um exemplo de modelo que descreve uma interface de usuário do tipo *Create/Retrieve/Update/Delete* (CRUD) para objetos persistentes em aplicativos de negócio [11]. O metamodelo correspondente define uma DSML que prevê apenas elementos estruturais da interface de usuário, podendo seus modelos serem classificados como modelos de produto. Entretanto, não podemos afirmar que esses modelos não são executáveis. Um motor, que implementa a semântica de execução, pode ser capaz de renderizar a interface e processar todas as interações com o usuário, tais como o clique em um botão para inserir um objeto. Em todo o caso, esta classificação ainda provê alguma evidência sobre a executabilidade de um dado tipo de modelo.

Para dar continuidade no mapeamento de modelos adaptativos, precisamos considerar apenas os modelos executáveis de processo. Sendo assim, deve-se pressupor que um modelo de produto executável pode ser mapeado em um modelo de processo equivalente. Sempre pode ser criado um modelo de processo de mais baixo nível de abstração (em outra linguagem de modelagem) que incorpore explicitamente conceitos implícitos no modelo de mais alto nível. Mesmo que esse mapeamento não seja de interesse da MDE, pois esta abordagem clama pelo aumento do nível de abstração, é de interesse para a caracterização e mapeamento desse tipo de modelo. Este princípio pode ser argumentado pela analogia com programas de computador: um programa escrito em linguagem de alto nível, que torna implícitos detalhes computacionais de execução, sempre pode ser reescrito ou compilado em um código equivalente em linguagem de mais baixo nível, no qual detalhes computacionais de execução são mais evidentes.

Um padrão recorrente pode ser observado nos constituintes da i-DSML [12][15]:

- Parte estática ou estrutural: metaelementos que expressam a organização estrutural do processo modelado;
- Parte dinâmica: metaelementos que expressam o estado do modelo sendo executado;
- Semântica de execução: expressa como o modelo se comporta enquanto estiver sendo executado.

Os metaelementos estruturais da parte estática são implementados por metamodelagem tradicional. No metamodelo da máquina de estados, eles seriam as metaclasses que representam estados, transições e eventos, por exemplo. Os metaelementos de estado da parte dinâmica têm por objetivo armazenar o estado do modelo em um dado ponto de execução no tempo. São eles que tornam os modelos autocontidos. Essas duas partes compõem, parcialmente, o metamodelo da i-DSML.

Além de considerar modelos de processo, também precisaremos considerar apenas modelos autocontidos. Se o metamodelo da i-DSML sob consideração não previr esses metaelementos, deve-se completá-lo para tal.

A semântica de execução pode ser definida de várias maneiras. Uma semântica axiomática permite completar a especificação do metamodelo com regras sobre como o modelo evolui em tempo de execução. Uma semântica translacional consiste em defini-la através de mapeamento com a semântica de execução de outro dispositivo executável já conhecida. Uma semântica operacional, usada mais adiante neste trabalho, consiste em definir o comportamento de execução em termos de ações através da implementação de um motor capaz de interpretar os modelos [12].

B. Modelos Adaptativos

Um modelo adaptativo é um modelo executável com capacidade de automodificação ao longo de sua execução. Uma i-DSML adaptativa permite a elaboração de modelos adaptativos e esta estende o conceito de i-DSML através da adição de duas partes [12]:

- Parte adaptativa: metaelementos que expressam em que situações deve ocorrer adaptatividade durante a execução do modelo através de ações adaptativas, definidas mais adiante;
- Semântica adaptativa: expressa quais são os efeitos decorrentes das ações adaptativas.

Os metaelementos da parte adaptativa expressam as possibilidades de ocorrência da adaptatividade durante a execução dos modelos.

Um autômato adaptativo, por exemplo, poderia criar nele mesmo novos estados e transições em tempo de execução. A definição de propriedades que permitem especificar ações adaptativas de inserção de estados e transições são exemplos de metaelementos da parte adaptativa, conforme ilustrado pela Fig. 2 [16]. Nesse autômato *M*, a transição de q_1 para ele mesmo dispara uma ação adaptativa definida pela chamada da função adaptativa $\mathscr{F}(p)$ com o argumento $p = q_2$, que irá criar um novo estado entre q_1 e q_2 e modificar as transições para que o caminho entre q_1 e q_2 seja linear e através de símbolos *b*, passando pelos estados intermediários criados. Tal autômato é

capaz de reconhecer a linguagem $L(M) = \{a^n b^n, n \in \mathbf{N}, n \geq 1\}$, e a definição completa de $\mathcal{F}$ pode ser encontrada em [16], onde são apresentados outros detalhes necessários.

A semântica adaptativa também pode ser especificada de várias maneiras, tais como axiomática e operacional. O motor de execução adaptativo de determinada i-DSML consiste no motor de execução original não-adaptativo estendido com capacidades adaptativas [12]. Ele é composto por um conjunto de funções adaptativas básicas que são chamadas durante a execução do modelo quando as capacidades adaptativas são acionadas. As chamadas específicas das funções adaptativas, com valores atribuídos aos argumentos, são as ações adaptativas.

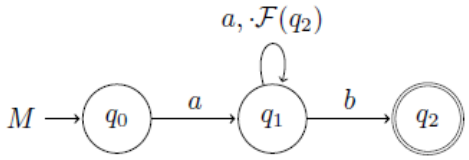


Fig. 2. Autômato adaptativo M que reconhece cadeias na forma $a^n b^n$ com $n \in \mathbf{N}$, $n \geq 1$. Fonte: [16].

Todas as partes da i-DSML adaptativa (estática, dinâmica, adaptativa e semânticas de execução e adaptativas) poderiam, em tese, ser alvo das ações adaptativas. Isso significa que a adaptatividade da adaptatividade (meta-adaptatividade) é possível quando a parte adaptativa está incluída como alvo possível das funções adaptativas.

III. Dispositivos Adaptativos Guiados por Regras

Esta seção apresenta uma revisão da formulação geral para dispositivos adaptativos guiados por regras [3], que são máquinas formais que permitem a alteração de seu comportamento dinamicamente como resposta a estímulos de entrada e sem interferência de agentes externos. A formulação se inicia através da especificação de dispositivos não-adaptativos. Em seguida, é apresentada a formulação de dispositivos adaptativos pela introdução de um mecanismo adaptativo sobre o dispositivo não-adaptativo subjacente. Nosso interesse é, na sequência, mapear modelos executáveis em dispositivos não-adaptativos guiados por regras e estabelecer modelos adaptativos como dispositivos adaptativos em que o dispositivo não-adaptativo subjacente seja o modelo executável não-adaptativo.

Um dispositivo não-adaptativo guiado por regras é todo e qualquer dispositivo formal cujo comportamento depende, de forma exclusiva, de um conjunto finito de regras que definem o mapeamento entre configurações [3]. O dispositivo é dito ser determinístico se, e somente se, para uma dada configuração inicial ou intermediária e um estímulo de entrada, seu conjunto de regras determina uma e apenas uma próxima configuração. Um dispositivo não-adaptativo ND é uma sêxtupla de acordo com (1).

$$ND = (C, NR, S, c_0, A, NA) \quad (1)$$

Em que:

- C é conjunto de todas as possíveis configurações de ND;
- NR é o conjunto de regras que descrevem ND;
- S é o conjunto de todos os possíveis eventos que são estímulos válidos de entrada para ND, com $\varepsilon \in S$, onde ε denota "vazio" e representa o elemento nulo do conjunto;
- $A \subseteq C$ é o subconjunto das configurações de aceitação;
- NA é o conjunto finito, com $\varepsilon \in NA$, de todos os possíveis símbolos que podem ser saídas de ND como efeitos da aplicação de regras em NR. Esses símbolos de saída podem ser mapeados em chamadas de procedimentos. Cadeias de símbolos de saída, portanto, podem ser interpretadas como sequências de chamadas.

Uma regra $r \in NR$ é uma quádrupla $r = (c_i, s, c_j, z)$ em que:

- $c_i \in C$ é a configuração de origem da regra r;
- $s \in S$ é o evento que dispara a regra r se a configuração corrente do dispositivo for c_i;
- $c_j \in C$ é a configuração de destino da regra r;
- $z \in NA$ é o símbolo a ser adicionado ao fim da cadeia de saída quando a regra r é disparada.

A regra r é dita ser compatível com a configuração corrente $c_{corrente}$ se, e somente se, $c_i = c_{corrente}$ [16]. A aplicação de r move o dispositivo da configuração c_i e este fato é denotado por $c_i \Rightarrow^s c_j$. Adicionalmente, z é adicionado ao fim da cadeia de saída do dispositivo. Notar que s, z ou ambos podem ser vazios.

Faça-se $c_i \Rightarrow^{\sim} c_m$ $(m \geq 0)$ denotar $c_i \Rightarrow^{\varepsilon} c_1 \Rightarrow^{\varepsilon} c_2 \Rightarrow^{\varepsilon} \ldots \Rightarrow^{\varepsilon} c_m$, que é uma sequência opcional de movimentos vazios. Faça-se também $c_i \Rightarrow^{\sim wk} c_j$ denotar $c_i \Rightarrow^{\sim} c_m \Rightarrow^{wk} c_j$, uma sequência opcional de movimentos vazios seguidas por uma não-vazia consumindo o símbolo w_k. Uma cadeia de entrada $w = w_1 w_2 \ldots w_n$ é dita ser aceita por ND quando $c_0 \Rightarrow^{\sim w1} c_1 \Rightarrow^{\sim w2} \ldots \Rightarrow^{\sim wn} c_n \Rightarrow^{\sim} c_{final}$, ou $c_0 \Rightarrow^{\sim w} c_{final}$, com $c_{final} \in A$.

Um dispositivo adaptativo AD é construído sobre um dispositivo não-adaptativo subjacente inicial ND_0 pela introdução de um mecanismo adaptativo. Seja k o passo de operação adaptativa de AD. Ele inicia com o valor zero e é automaticamente incrementado de um quando uma ação adaptativa é executada. Ele não é incrementado quando não ocorre uma ação adaptativa. Nesse caso, diz-se também que ocorreu uma ação adaptativa nula. Para cada passo de operação $k \geq 0$, AD segue o comportamento de ND_k até a próxima execução de uma ação adaptativa não-nula. ND_k denota o dispositivo não-adaptativo subjacente de AD no passo de operação adaptativa k.

Em qualquer $k \geq 0$, ND_k é definido pelo seu conjunto de regras NR_k. A execução de uma ação adaptativa não-nula incrementa k de 1 e faz ND_k evoluir para ND_{k+1}. Assim, AD inicia a operação no estágio $k + 1$ pela criação do conjunto NR_{k+1} como uma versão editada de NR_k. AD passará então a seguir o comportamento de ND_{k+1} até que outra ação adaptativa não-nula cause a evolução para ND_{k+2}. Este procedimento itera até que a cadeia de entrada seja totalmente processada.

O dispositivo adaptativo no estágio k é representado por (2). Essa representação inclui informação sobre o dispositivo não-adaptativo subjacente ND_k, que são C_k, S, c_k, A_k e NA.

$$AD_k = (C_k, AR_k, S, c_k, A_k, NA, BA, AA) \quad (2)$$

Assim, AD inicia sua operação na configuração c_0. A execução de uma ação adaptativa não-nula leva AD_k para $AD_{k+1} = (C_{k+1}, AR_{k+1}, S, c_{k+1}, A_{k+1}, NA, BA, AA)$, em que:

- AR_k é o conjunto de regras adaptativas conforme detalhado abaixo;
- BA e AA são os conjuntos de ações adaptativas "antes" (*before*) e "depois" (*after*), respectivamente, com $\varepsilon \in BA$ e $\varepsilon \in AA$.

O conjunto de regras adaptativas AR_k é dado por uma relação de acordo com (3).

$$AR_k \subseteq BA \times C_k \times S \times C_k \times NA \times AA \quad (3)$$

Isso quer dizer que cada regra adaptativa $ar \in AR_k$ é uma sêxtupla (ba, c_i, s, c_j, z, aa) significando que, em resposta a um estímulo de entrada $s \in S$, ar primeiro executa a ação adaptativa "antes" $ba \in BA$ (a execução de ba é abortada se ela elimina ar de AR_k), aplica a regra não-adaptativa subjacente $r = (c_i, s, c_j, z) \in NR_k$ e finalmente executa a ação adaptativa "depois" $aa \in AA$.

O leitor deve notar que NR_k não está explicitado em (2) pois está implícito em AR_k, descartando o primeiro e último elemento da sêxtupla.

Para que o dispositivo seja determinístico, para cada regra não-adaptativa proveniente de NR_k, existe um único par de ações adaptativas "antes" e "depois". Assim, o conjunto AR_k, relacionado ao estágio adaptativo k, pode ser obtido adicionando os devidos pares de ações adaptativas a cada regra não adaptativa pertencente a NR_k.

As ações adaptativas pertencentes a BA e AA podem ser definidas em termos de abstrações chamadas funções adaptativas, de modo similar às chamadas de funções em linguagens de programação usuais [16]. A especificação de uma função adaptativa inclui os seguintes elementos: (a) um nome simbólico, (b) parâmetros formais que referenciam valores passados como argumentos, (c) variáveis que armazenam valores de uma ação elementar de inspeção, (d) geradores que referenciam valores novos a cada utilização, e (e) o corpo da função.

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

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

Essas ações adaptativas elementares podem ser utilizadas no corpo de uma função adaptativa, incluindo padrões de regras que utilizem parâmetros formais, variáveis e geradores disponíveis [3][16].

IV. De Modelos Adaptativos Para Dispositivos Adaptativos Guiados por Regras

Esta seção apresenta um mapeamento de modelos adaptativos para a formulação de dispositivos adaptativos guiados por regras de [3] apresentado na seção III. Primeiramente, um modelo executável é mapeado em um dispositivo não-adaptativo seguindo a forma de (1). Para que isso seja possível de algum modo, recorre-se ao padrão de projeto de i-DSMLs proposto por [15]. Em seguida, o padrão é estendido para incluir uma parte adaptativa, a qual é mapeada no mecanismo adaptativo.

A. Padrão de Projeto para i-DSMLs

Seja L uma i-DSML estabelecida pelo metamodelo MM_L. Assume-se que L seja uma linguagem de modelos de processo autocontidos, conforme discutido na seção II. Ou seja, MM_L contém explicitamente metaelementos que expressam a parte dinâmica dos modelos, a qual engloba representação dos estados de execução possíveis, conforme visto na seção II.

Em uma i-DSML de máquinas de estado, por exemplo, essa parte do metamodelo pode ser expressa por uma propriedade *booleana* *isCurrent* na metaclasse *State* e uma restrição indicando que só pode haver um único estado do modelo (instância de *State*) com o valor *True* para esta propriedade em cada passo de execução. A Fig. 3 apresenta uma versão atualizada da metaclasse *State* da Fig. 1 contendo esta propriedade.

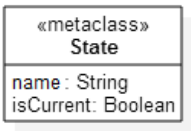


Fig. 3. Metaclasse *State* estendida com propriedade da parte dinâmica.

Em uma i-DSML de redes de Petri, como exemplo adicional, o estado de execução do modelo pode ser expresso por uma propriedade *numberOfTokens* na metaclasse que representa o conceito de posição, e esta armazenará um número inteiro (eventualmente com alguma restrição limitadora) para cada instância de posição nos modelos executados.

Nos exemplos dos dois últimos parágrafos, a parte dinâmica do metamodelo foi definida junto com a parte estrutural, não havendo uma distinção clara, à primeira vista, de quais são os elementos estruturais e quais são os dinâmicos. Isso porque a propriedade *isCurrent* (parte dinâmica) está definida juntamente com a propriedade *name* (parte estrutural) na

metaclasse *State*. Somente o leitor com conhecimento da semântica de execução desta linguagem conseguiria fazer essa distinção entre as partes.

Para que fique explícita esta distinção sem necessidade de interpretação sob a luz da semântica, [15] propõe um padrão de projeto (*design pattern*) para metamodelos de linguagens de modelagem executáveis. Nesse padrão, três pacotes (*packages*) MOF compõem o metamodelo e são unidos por associações de mesclagem (*merge*): *Domain Definition MetaModel* (DDMM), *State Definition MetaModel* (SDMM) e *Event Definition MetaModel* (EDMM). O primeiro pacote refere-se à parte estrutural, enquanto os dois últimos à parte dinâmica.

O SDMM engloba as propriedades necessárias para armazenar o estado de execução do modelo. Enquanto a propriedade *name* de *State* está no pacote DDMM, *isCurrent* fica em SDMM. A operação de mesclagem entre ambos gera uma metaclasse *State* completa com ambas as propriedades. Enquanto isso, o pacote EDMM especifica os eventos de execução que dirigem a execução dos modelos conformes a MM_L. Podem ser não apenas eventos de hardware, mas também eventos de software mais abstratos tais como eventos de leitura e escrita, eventos de comunicação para envio e recebimento de dados, notificações de relógios, etc. [15]. Os eventos concretos que ocorrem na execução de um modelo definem valores de propriedades dos elementos de EDMM. No exemplo da máquina de estado da Fig. 1, a metaclasse *Event* faria parte do pacote EDMM e é referenciada pela propriedade *event* de *Transition*.

Ademais, para modelos baseados em eventos discretos, o padrão estabelece que as metaclasses de evento de EDMM devem ser todas subclasses de uma metaclasse mãe abstrata, que abstrai o conceito de estímulo e que fica em um quarto pacote: *Trace Management MetaModel* (TM3).

O padrão também estabelece o pacote *Semantics* que carrega a semântica de execução nos modelos. Algumas considerações sobre esta parte serão vistas adiante. A Fig. 4 provê uma ilustração com visão geral sobre o padrão.

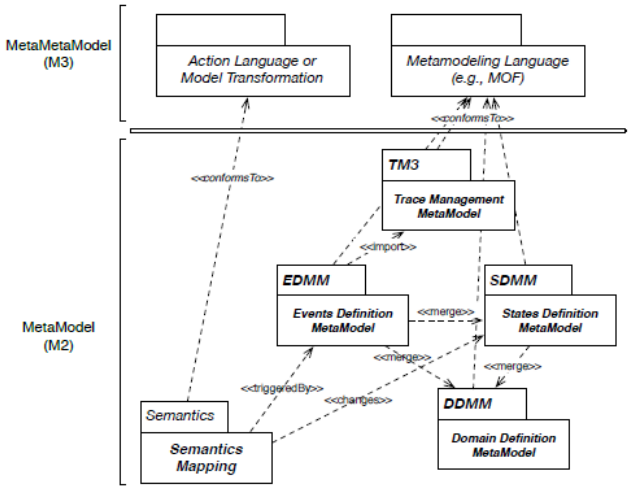


Fig. 4. Padrão de projeto para i-DSMLs. Fonte: [15].

B. Mapeamento de Modelos Executáveis em Dispositivos Não Adaptativos

O mapeamento de modelos executáveis em dispositivos não-adaptativos guiados por regras conforme (1) é apresentado através da descrição de um algoritmo com exemplos em cada passo. Como entradas, este algoritmo recebe o modelo executável que se deseja mapear e o metamodelo da i-DSML. Supõe-se que o metamodelo siga o padrão de projeto descrito na subseção anterior, ou seja, é possível distinguir os constituintes pertencentes a DDMM, SDMM e EDMM. Como saída, é produzido um dispositivo não-adaptativo $ND = (C, NR, S, c_0, A, NA)$.

ND inicia com *C*, *NR* e *A* vazios e c_0 nulo. *NA* e *S* iniciam como $\{\varepsilon\}$. Em seguida, uma configuração é gerada e incluída no conjunto *C* para cada combinação possível de valores das propriedades de estado (aquelas definidas em SDMM) para os elementos do modelo que a contém. As restrições definidas no metamodelo devem ser respeitadas.

No exemplo da máquina de estados, a propriedade *isCurrent* é a única definida em SDMM e pode assumir os valores *False* e *True*. Digamos que um modelo específico contenha três estados: *S1*, *S2* e *S3*, que são instâncias da metaclasse *State*. Essas instâncias assumem valores para *isCurrent*, e as combinações podem ser representadas por tuplas do tipo (*True, False, False*), (*False, True, False*), (*True, True, False*), etc. Uma restrição no metamodelo impõe que só pode haver o valor *True* para um único estado simultaneamente. Isso descarta a combinação (*True, True, False*), por exemplo, restringindo o conjunto de combinações na forma de tuplas que representam as configurações válidas do modelo em execução. Transições (instâncias de *Transition*) não são consideradas aqui pois não possuem propriedades definidas em SDMM, ou seja, não armazenam informações que definem o estado de execução do modelo.

No exemplo de redes de Petri, a propriedade *numberOfTokens* definida em SDMM representa o número de *tokens* contidos em cada posição em um dado momento de execução. Nesse caso, para uma rede com 3 posições, tuplas do tipo (*0,0,0*), (*1,2,0*), (*0,3,1*), etc. representam as configurações possíveis. É claro que se torna inviável criar configurações para cada combinação de inteiros possível, até porque o conjunto *C* seria infinito e isso não é permitido na definição de dispositivo não-adaptativo [3]. Nesse caso, deve-se impor restrições que limitem a faixa de operação do modelo conforme necessidade de aplicação, gerando um número de configurações finito condizente com a necessidade.

De maneira similar, geram-se as combinações de eventos concretos possíveis para popular o conjunto *S* de *ND*. Os eventos concretos do modelo executável são as instâncias das metaclasses definidas em EDMM preenchidas com cada combinação de valores de suas propriedades.

Seja um modelo de sistema de controle de velocidade de um trem, adaptado de [12] e apresentado na Fig 5. Na sintaxe concreta adotada para este modelo, as transições estão conectadas por linhas aos objetos de eventos que a disparam.

Assim, destaca-se que os eventos definidos no modelo são instâncias de metaclasses de EDMM.

O modelo apresentado na Fig. 5 é uma instância do metamodelo da Fig. 6, que consiste em uma especialização de máquinas de estado.

A três metaclasses de evento definidas em EDMM são *RedEvent*, *AmberEvent* e *GreenEvent* representando sinalizações luminosas. O primeiro tipo de evento indica que o trem deve parar. O segundo indica que deve trafegar a 40 km/h, enquanto o terceiro estabelece o nível normal de velocidade que pode ser 100 km/h ou 130 km/h. A sinalização *Green* é acompanhada por uma informação de qual velocidade o trem deve atingir. O sistema de controle do trem recebe essas três possíveis sinalizações do ambiente e ajusta sua velocidade de acordo. *RedEvent* e *AmberEvent* não possuem propriedades, enquanto *GreenEvent* possui a propriedade *speed* que pode assumir os valores 100 ou 130 nas instâncias. Cada sinalização é representada por uma instância de uma das três metaclasses, a saber: *Red*, *Amber*, *Green100* e *Green130*.

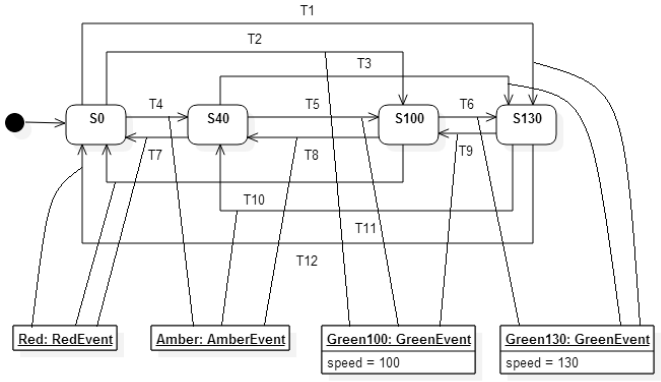


Fig. 5. Máquina de estados de sistema de controle de trem.

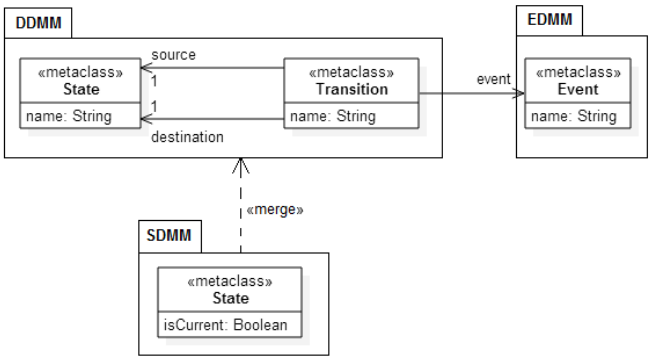


Fig. 6. Metamodelo (parcial) ao qual a máquina de estados de sistema de controle de trem é conforme.

O conjunto *NR* de *ND* deve ser montado de acordo com a semântica de execução do modelo. A semântica de um modelo executável é atrelada a um modelo de computação, ou *Model of Computation* (MoC). Considera-se para os fins deste trabalho um modelo de computação baseado em eventos discretos.

Uma das formas de se especificar essa semântica [15] é através de uma função de transição *f* que estabelece como o modelo evolui em sua execução. Esse modo reflete uma semântica operacional através da definição explícita das transições. Uma semântica translacional, por exemplo, traduziria os elementos da i-DSML em outra i-DSML com semântica conhecida. Deve-se supor que o algoritmo de mapeamento aqui proposto possua acesso a *f*, mesmo que tenha sido estabelecida de forma translacional, a tradução intermediária fica transparente para o algoritmo.

A função *f* toma como argumentos um modelo, uma representação do estado de origem, que pode ser a tupla que carrega os valores de propriedades de estado para cada elemento, e uma representação de evento concreto. É retornado o estado de destino do modelo na mesma representação do estado de origem.

No algoritmo de mapeamento, para cada combinação estado de origem × evento concreto do modelo, deve ser calculado o estado destino através de *f* e deve ser gerada uma regra a ser incluída em *NR* com os correspondentes objetos de configuração de origem, evento e configuração de destino do dispositivo não-adaptativo.

Deve-se frisar que *f* é definida com relação ao metamodelo, e não para modelos específicos. Por exemplo, para o metamodelo da Fig. 1, *f(M,descritor_estado, descritor_evento*) pode ser definida, resumidamente, através de um algoritmo que executa os seguintes passos: (i) localizar e armazenar na variável local *t* a instância da metaclasse *Transition* em *M* cuja propriedade *source* corresponda ao estado designado por *descritor_estado* e a propriedade *event* seja o estado designado por *descritor_evento*; (ii) obter o valor da propriedade *destination* de *t* e armazenar na variável *d*; (iii) retornar tupla de valores de propriedades de estado que corresponda a *isActive = False* para todos os estados exceto *d*, que deve corresponder a *True* (descritor do próximo estado). Dessa forma, *f* se aplica a qualquer máquina de estado conforme ao metamodelo.

Observa-se que a função *f*, que define a semântica de execução, "conhece" o metamodelo estruturalmente para determinar a próxima representação de estado com base no significado pretendido das abstrações. Por outro lado, sendo especificada desta maneira, em que os argumentos de estado e evento são descritores gerais aplicáveis a quaisquer modelos conformes a metamodelos que seguem o padrão de projeto, consegue-se padronizar também a assinatura *f(M, descritor_estado, descritor_evento*) → *descritor_estado* como forma de especificar uma semântica operacional.

O fato de *f* ser uma função implica que os modelos sejam determinísticos. Trabalha-se com esta hipótese por simplificação, mas se *f* for uma relação, então pode-se trabalhar também com modelos executáveis não determinísticos mediante algumas adaptações no algoritmo.

O padrão de projeto de [15] não especifica uma maneira padronizada de definir o estado inicial dos modelos executáveis. Essa característica seria importante para que o mapeamento proposto possa identificar o estado inicial de

qualquer modelo. Sendo assim, como os modelos são autocontidos por hipótese, vamos estabelecer que cada um deva, obrigatoriamente, definir um valor inicial (*default*) para cada propriedade proveniente de SDMM para cada uma de suas instâncias. No exemplo da Fig. 5, o estado inicial *S0* está indicado pela notação padrão de máquinas de estado. Porém, no modelo em si, essa situação é representada por *S0* possuir o valor inicial *True* para a propriedade *isCurrent*, enquanto os outros estados possuem *False*.

O algoritmo aqui apresentado é resumido em alto nível na Fig. 7, sem detalhar os laços de geração de combinações para fins de brevidade. Também não são detalhadas estruturas de dados auxiliares que mantêm correspondências entre elementos mapeados para uso ao longo do algoritmo.

O algoritmo da Fig. 7 não explora o conjunto *A* de estados de aceitação e nem o conjunto *NA* de símbolos de saída. Eles são sempre retornados como vazio e {ε}, respectivamente. Esta parte será deixada como possibilidade de extensão futura, a ser discutida na seção V.

C. Extensão do Padrão de Projeto para Adaptatividade

Embora o padrão de projeto proposto por [15] não preveja adaptatividade, é possível propor sua extensão através da inclusão de metaelementos que abstraem os conceitos necessários.

Propomos a inclusão de um pacote que represente a parte adaptativa do metamodelo da i-DSML descrita na seção II. Seguindo o padrão de nomenclatura, denominemos este pacote de *Adaptive Layer Definition MetaModel* (ALDMM).

Na extensão aqui proposta, o ALDMM é um pacote padronizado que pode ser aproveitado por todas as i-DSMLs. Ele define abstrações cujas instâncias especificam funções adaptativas, tomando como referência as ideias de [10] para modelagem de programas adaptativos, que é um tipo de dispositivo adaptativo. A Fig. 8 apresenta este pacote.

As decisões para a formulação dos metaelementos de ALDMM também foram baseadas nos conceitos apresentados pela seção III. Ou seja, usou-se o conhecimento mais geral da área de adaptatividade para estender o padrão de metamodelagem referido, incorporando explicitamente conceitos de adaptatividade. Os modelos executáveis das linguagens de modelagem que seguem este padrão ganham então a possibilidade de apresentar capacidades adaptativas, trazendo esta abordagem para o contexto da MDE.

A metaclasse *AdaptiveFunction* abstrai o conceito de função adaptativa contendo as propriedades *name* (nome simbólico), *parametersNames* (*array* de nomes de parâmetros), *generatorsNames* (*array* de nomes de geradores) e *body* (corpo da função). Cada instância desta metaclasse no modelo adaptativo representa uma função adaptativa que pode ser usada para compor ações adaptativas.

A metaclasse *AdaptiveAction* corresponde à abstração que representa as ações adaptativas, ou seja, chamadas de funções adaptativas. Cada instância desta metaclasse no modelo representa uma chamada de função adaptativa. Esta metaclasse está associada a *AdaptiveFunctionCallArgument* com multiplicidade 0..*.

```
Procedimento
Mapeia_Modelo_Executavel_em_Dispositivo_Nao_Adapt:

Entradas: M: Modelo executável
          MM: Metamodelo composto por DDMM,SDMM,
              EDMM, semântica operacional f
              e conjunto de restrições R.
Saídas:   ND: Dispositivo não-adaptativo guiado
              por regras = (C,NR,S,c0,A,NA)
Variáveis: config_candidata,c: tupla;
           lista_config_candidatas: conjunto;
           evento_candidato,s:  Inteiro;
           lista_eventos_candidatos: conjunto;
Início:

// Inicializações da saída:
C := conj. vazio; NR := conj. vazio;
S := {ε}; c0 := nulo;
A := conj. vazio; NA := {ε};

lista_config_candidatas :=
  Gerar lista de tuplas representando todas as
combinações de valores que representam estados de
execução do modelo. // Cada tupla codifica uma
configuração.

// Monta o conjunto C, adicionando apenas tuplas
// relativas a configurações que respeitam o
// conjunto R de restrições do metamodelo MM:
Para cada config_candidata em
          lista_config_candidatas {
   Se estado corresp. a config_candidata respeita R
   { C := C ∪ config_candidata; }
}

lista_eventos_candidatos :=
  Obter todos os elementos do modelo que são
instâncias de metaclasses provenientes de EDMM.
Cada uma será mapeada em um número inteiro (ex:
sequencial) que representará um símbolo permitido
na cadeia de entrada, e este inteiro será parte da
lista.

// Monta o conjunto S, adicionando apenas
// eventos que respeitam o conjunto R de restrições
// do metamodelo MM:
Para cada evento_candidato em
          lista_eventos_candidatos {
   S := S ∪ evento_candidato;
}

// Monta o conjunto NR
Para cada c em C {
   Para cada s em S {
      Se definido f(descritor de c,
                    descritor de s) {
         NR := NR ∪ (c,
                     s,
                     config. corresp. a
                     f(descritor de c,
                       descritor de s),
                     ε)
      }
   }
}

c0 := tupla representando a combinação de valores
iniciais das propriedades que representam estados
de execução do modelo;

Fim.
```

Fig. 7. Algoritmo em alto nível que mapeia modelos executáveis em dispositivos não-adaptativos guiados por regras.

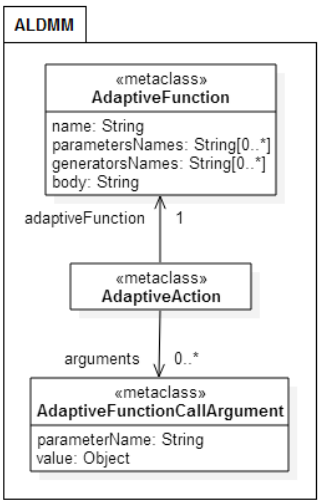


Fig. 8. Pacote ALDMM que estende o padrão de projeto de i-DSMLs executáveis para i-DSMLs adaptativas.

Cada instância dessa última metaclasse é um argumento que determina um valor para um dos parâmetros da função adaptativa chamada. A Fig. 9 mostra o padrão de projeto atualizado com a parte e semântica adaptativa.

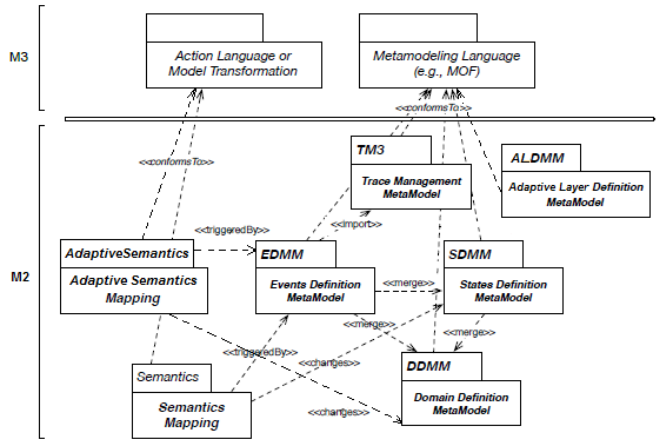


Fig. 9. Extensão do padrão de projeto para i-DSMLs adaptativa.

Conforme visto na seção II, também se faz necessária a introdução de uma semântica adaptativa para definir uma i-DSML adaptativa. Na subseção anterior apresentamos uma definição de semântica operacional através de uma função *f* que explicitamente define como o modelo executável evolui.

A semântica adaptativa pode ser estabelecida do mesmo modo por uma função *g* que toma os mesmos argumentos como entrada: um modelo, uma representação do estado de origem do sistema na forma de tupla que carrega os valores de propriedades de estado para cada elemento, e uma representação de evento concreto. Como saída, *g* fornece um par de ações adaptativas a serem aplicadas antes e depois da efetivação do passo de evolução do modelo executável. Como o pacote ALDMM tem a intenção de ser o mesmo para todos os metamodelos de i-DSMLs adaptativas, então dessa forma a assinatura da semântica operacional *g*(*M*, *descritor_estado*, *descritor_evento*) → (*ação_adaptativa_antes*, *ação_adaptativa_depois*) também fica padronizada, não sendo específica por metamodelo.

Uma constante *EmptyAdaptiveAction* representa uma ação adaptativa nula. Lembrando que ações adaptativas são instâncias de *AdaptiveAction* no modelo executável.

O corpo da função adaptativa é modelado como uma *string* através da propriedade *body* de *AdaptiveFunction*. Não há imposição aqui sobre a linguagem na qual ela deve ser definida. Para oferecer uma alternativa, citamos o formalismo SBMM [20], que define funções de edição de modelos genéricas, podendo ser usadas para modelos conformes a qualquer metamodelo.

- *insert_empty_instance(M, $name_i$, $name_c$)*: retorna um modelo atualizado construído pela inserção de uma nova instância nomeada por $name_i$ no modelo *M*, cuja metaclasse é aquela de nome $name_c$. Os valores iniciais de cada propriedade são os valores padrões definidos no metamodelo;
- *insert_or_edit_slot*(*M*, $name_i$, $name_p$, *val)*: retorna um modelo atualizado construído pela inserção de um novo *slot* [20] em *M*, i.e, atribuição de valor de uma propriedade a uma instância do modelo, ou atualização do mesmo caso o *slot* da mesma propriedade já exista. O *slot* será atribuído a instância de nome $name_i$, referente à propriedade de nome $name_p$ e terá valor *val*;
- *delete_slot(M, $name_i$, $name_p$)*: retorna um modelo atualizado com a remoção do *slot*, previamente adicionado, da instância $name_i$ de *M* referente a propriedade $name_p$, sendo de interesse para limpar valores atribuídos a propriedades em uma instância;
- *delete_instance(M, $name_i$)*: retorna um modelo atualizado com a remoção da instância de nome $name_i$ de *M*;
- *query_instance_by_prop(M, $name_p$, val)*: retorna um subconjunto de instâncias de *M* cujos *slots* da propriedade $name_p$ possuem *val* como valor atribuído.

Tais funções podem ser usadas como base para a construção do corpo de uma função adaptativa no contexto de modelos adaptativos.

Tendo em vista que as ações e funções adaptativas do modelo são definidas, pela extensão do padrão de projeto, como instâncias do mesmo modelo, mais especificamente das metaclasses *AdaptiveFunction*, *AdaptiveAction* e *AdaptiveFunctionCallArgument*, então as funções do SBMM aqui apresentadas também são capazes de alterar a parte adaptativa do modelo, provendo capacidade de meta-adaptatividade. No entanto, para fins do mapeamento pretendido, é imposta a restrição de que só podem ser utilizadas chamadas de funções adaptativas que alterem a parte estrutural do modelo, isto é, que criem ou removam instâncias de metaclasses de DDMM. Eventos concretos ou até mesmo ações adaptativas não podem ser criadas ou removidas pois o dispositivo adaptativo conforme formulado em [3] não permite a alteração do conjunto *S* nem *BA* ou *AA* em cada passo.

D. Mapeamento do Mecanismo Adaptativo

O algoritmo apresentado na subseção III.B é aqui invocado em um outro algoritmo que o completa, gerando também os

conjuntos *AR*, *BA* e *AA* do dispositivo adaptativo na forma de (2).

```
Procedimento
Mapeia_Modelo_Adaptativo_em_Dispositivo_Adapt:

Entradas: M: Modelo adaptativo
          MM: Metamodelo composto por DDMM,SDMM,
              EDMM, semântica operacional f,
              conjunto de restrições R, ALDMM e
              semântica adaptativa operacional g.
Saídas:   AD: Dispositivo adaptativo guiado
              por regras =
              (C,AR,S,c0,A,NA,BA,AA)
Variáveis: ND: Dispositvo não-adaptativo;
           par_aa,ar: tupla;
           lista_config_candidatas: conjunto;
           evento_candidato,s:  Inteiro;
           lista_eventos_candidatos: conjunto;
Início:

  // Inicializações da saída:
  AR := conj. vazio;
  BA := conj. vazio;
  AA := conj. vazio;

  ND :=
  Mapeia_Modelo_Executavel_em_Dispositivo_Nao_Adapt
   (M,MM)

  // Consideremos que ND = (C,NR,S,c0,A,NA) e
  // seus membros podem ser acessados por este
  // algoritmo.
  Para cada r = (c_src,s,c_dst,x) em NR {
     Se definido g(descritor de c_src,
                   descritor de s) {
        // par_aa = par de ações adaptativas
        // "antes" e "depois".
        par_aa := g(descritor de c_src,
                    descritor de s);
        // par_aa[1] é ação "antes".
        // par_aa[2] é ação "depois".
        BA := BA ∪ converte_aa(par_aa[1]);
        AA := AA ∪ converte_aa(par_aa[2]);
        AR := AR ∪ (converte_aa(par_aa[1]),
                    c_src,
                    s,
                    c_dst,
                    ε,
                    converte_aa(par_aa[2]));
     } Senão {
        AR := AR ∪ (ε,
                    c_src,
                    s,
                    c_dst,
                    ε,
                    ε);
     }
  }
Fim.
```

Fig. 10. Algoritmo em alto nível que mapeia modelos adaptativos em dispositivos adaptativos guiados por regras.

A função *converte_aa* usada no algoritmo abstrai a conversão de ações adaptativas como instâncias de *AdaptiveAction*, que seguem o padrão de projeto, para objetos que representam ações adaptativas dos conjuntos *BA* e *AA* de acordo com a formulação de dispositivos guiados por regras de [3].

V. Discussão

Este trabalho apresentou um mapeamento de modelos executáveis adaptativos para dispositivos adaptativos guiados por regras conforme a formulação de [3]. Para que isso fosse possível, foi necessário primeiramente reunir conceitos da literatura para estabelecer o entendimento sobre modelos executáveis e modelos adaptativos.

Como os graus de liberdade para se definir modelos executáveis são vários, buscou-se restringir o universo para modelos de processo autocontidos, conforme discussão da subseção II.A.

A estratégia geral adotada foi mapear modelos executáveis em dispositivos não-adaptativos e em seguida mapear a camada adaptativa dos modelos adaptativos para o mecanismo adaptativo dos dispositivos adaptativos guiados por regras. Para estabelecer o mapeamento, era necessário extrair de forma genérica, de qualquer modelo executável do universo considerado, as partes que descrevem seu estado e as partes que descrevem os eventos que causam a evolução do modelo no passo de execução. Para atingir esse objetivo, recorreu-se ao padrão de projeto de [15], que estabelece bem essa distinção através da separação em pacotes dos metaelementos estruturais, de estado e de eventos. Esse padrão de projeto refere-se ao nível dos metamodelos (e não ao nível dos modelos), e por isso também é chamado de padrão de metamodelagem [15].

Como o padrão mencionado não apresentava metaelementos para definir o estado inicial dos modelos conformes, foi assumido que as propriedades de estado definidas no metamodelo da i-DSML devam obrigatoriamente conter um valor inicial nos modelos, e esta combinação de valores define o estado inicial. Isso foi necessário para o mapeamento, já que a configuração inicial do dispositivo não-adaptativo é obrigatória. Os conceitos de símbolos de saída e de configurações de aceitação não foi explorado no mapeamento pois não existem no padrão apresentado. Uma sugestão de trabalho futuro é criar outra extensão do padrão de projeto que permita definir esses conceitos na i-DSML projetada e explorá-los no mapeamento. Mesmo sem eles, a executabilidade e adaptatividade dos modelos, foco deste trabalho, não é prejudicada.

Também para que o trabalho fosse possível, foi necessário estender o padrão de projeto para incluir a parte adaptativa e sua semântica, conforme previsto, mas não detalhado por [12]. O pacote ALDMM foi definido através de metaclasses que representam ações adaptativas e funções adaptativas. Recorreu-se ao formalismo SBMM [20], já que o MOF não estabelece operadores ou funções de edição de modelos, para se apresentar funções de edição de modelos que possam servir para especificar o corpo das funções adaptativas.

O mapeamento aqui apresentado não se propõe a ser o único possível, até mesmo por impor algumas restrições aos modelos de entrada e assumir que a i-DSML segue determinado padrão de projeto. Ele pode ser adaptado ou especializado conforme necessidade ou características da i-DSML em consideração.

Um dos problemas identificados é o alto número de combinações possíveis de preenchimento de propriedades de estado, principalmente quando números inteiros são utilizados para determinar o estado de execução do modelo. Este problema foi identificado no exemplo das redes de Petri e pode gerar muitas configurações no conjunto *C* do dispositivo não-adaptativo. A delimitação de valores permitidos conforme necessidade de operação prática pode limitar esse conjunto. No entanto, para fins conceituais, não há essa preocupação desde que *C* tenha um número finito de elementos.

VI. CONCLUSÃO

Modelos executáveis são de grande importância para a MDE na medida em que são uma das possibilidades de aplicação prática desta abordagem, consistindo em uma alternativa à transformação de modelos. A MDE é considerada uma das mais recentes mudanças de paradigma da Engenharia de Software [21]. Propõe trazer dois benefícios principais: o aumento do nível de abstração em relação à programação tradicional e a possibilidade de aproveitar os mesmos modelos de alto nível para gerar ou executar o sistema em várias plataformas.

Dentro do contexto de modelos executáveis, a adaptatividade é um recurso ainda pouco estudado [12] e, portanto, um campo a ser explorado. Por outro lado, o estudo da adaptatividade de forma geral já vem ocorrendo por diversos pesquisadores e apresenta resultados em várias áreas [4][8][9]. Sendo assim, estender o padrão de metamodelagem de x-DSMLs encontrado na literatura [15] para que as linguagens de modelagem incorporem capacidades adaptativas para seus modelos é de grande valia para ampliar os horizontes da modelagem executável. Além disso, colocar modelos executáveis no contexto mais amplo de dispositivos adaptativos guiados por regras permite aproveitar os resultados e o *toolbox* da teoria subjacente. Como exemplo, o algoritmo de operação de dispositivos adaptativos genéricos guiados por regras de [3] poderia ser aplicado para o desenvolvimento de motores de execução de modelos adaptativos com a utilização do mapeamento proposto.

Dessa forma, este trabalho contextualiza os modelos adaptativos dentro da área de pesquisa da adaptatividade, ampliando e conectando os conhecimentos. Com isso, busca-se reduzir o *gap* cultural e técnico entre as comunidades de metamodelagem/MDE e de teorias e modelos de computação.

Um possível trabalho futuro é a mudança da definição de semântica operacional de função para relação e a respectiva adaptação dos algoritmos para considerar modelos executáveis não determinísticos.

VII. REFERÊNCIAS

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

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

[3] J. J. Neto, *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.

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

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

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

[7] S. R. M. Canovas and C. E. Cugnasca, C., "Um Metamodelo para Programas Adaptativos Utilizando SBMM". *WTA 2014, Oitavo Workshop de Tecnologias Adaptativas*, São Paulo, 2014.

[8] A. Contier, D. Padovani and J. J. Neto, "Linguístico: Usando Tecnologia Adaptativa para a Construção de Um Reconhecedor Gramatical". *WTA 2014, Oitavo Workshop de Tecnologias Adaptativas*, São Paulo, 2014.

[9] A. P. Silva, R. I. Silva Filho and F. A. Rodrigues, "Swarm Robotics: comportamento Adaptativo Aplicado ao problema de dispersão de pássaros em áreas agrícolas". *WTA 2014, Oitavo Workshop de Tecnologias Adaptativas*, São Paulo, 2014.

[10] S. R. M. Canovas and C. E. Cugnasca, "Em Direção ao Desenvolvimento de Programas Adaptativos Utilizando MDE", *WTA 2015, Nono Workshop de Tecnologias Adaptativas*, São Paulo, 2015.

[11] S. R. M. Canovas and C. E. Cugnasca, "Applying Model Driven Engineering to Develop a Bee Information System", in *Proc. EFITA*, Poznan, Poland, 2015, pp. 103-114.

[12] E. Cariou, O. Le Goaer, F. Barbier and S. Pierre, "Characterization of Adaptable Interpreted-DSML", in *European Conference on Modeling Foundations and Applications (ECMFA 2013)*, volume 7949 of LNCS, pp. 37-53.

[13] E. Breton and J. Bézivin, "Towards and understanding of model executability", in *Proceedings of the international conference on Formal Ontology in Information Systems (FOIS '01)*. ACM (2001).

[14] E. Cariou, C. Ballagny, A. Feugas and F. Barbier, "Contracts for Model Execution Verification", in *Seventh European Conference on Modeling Foundations and Applications (ECMFA 2011)*, volume 6698 of LNCS. Springer (2011).

[15] B. Combemale, X. Crégut and M. Pantel, "A Design Pattern to Build Executable DSMLs and associated V&V tools", in *The 19th Asia-Pacific Software Engineering Conference (APSEC 2012)*. IEEE (2012).

[16] P. R. M. Cereda and J. J. Neto, "Utilizando linguagens de programação orientadas a objetos para codificar programas adaptativos", *WTA 2015, Nono Workshop de Tecnologias Adaptativas*, São Paulo, 2015.

[17] B. Combemale, C. Hardebolle, C. Jaquet, F. Boulanger and B. Baudry, *Bridging the Chasm between Executable Metamodeling and Models of Computation*. Software Language Engineering, 2013, pp. 184-203.

[18] E. Cariou, O. Le Goaer and F. Barbier, "Model Execution Adaptation?", in *7th International Workshop on Models@run.time (MRT 2012) at MoDELS 2012*.

[19] S. Pierre, E. Cariou, O. Le Goaer and F. Barbier, "A Family-based Framework for i-DSML Adaptation", in *European Conference on Modeling Foundations and Applications (ECMFA 2014)*, pp. 164-179.

[20] S. R. M. Canovas, *Uma proposta de formalismo para metamodelagem e suas aplicações na MDE*. PhD thesis, Escola Politécnica, Universidade de São Paulo, 2015. /no prelo/

[21] D. S. Kolovos, L. M. Rose, N. Matragkas, R. F. Paige, E. Guerra, J. S. Cuadrado, J. De Lara, I. Rath, D. Varró, M. Tisi and J. Cabot, "A research roadmap towards achieving scalability in Model Driven Engineering", in *Proceedings of the Workshop on Scalability in Model Driven Engineering*, pp. 2, ACM, 2013.

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

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

Proposta de um Mecanismo Adaptativo para Seleção de Planos de Compilação em Compiladores JIT

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

***Resumo*—Linguagens de programação que têm como uma de suas principais características a portabilidade de seus programas têm utilizado sistemas baseados em compilação *Just-in-Time* (JIT) para melhorar o desempenho de seus programas. Esses sistemas utilizam políticas de compilação para identificar *se* e *quando* diferentes regiões do programa devem ser compiladas e também *como*, através de um *plano de compilação*, compilar essas regiões. Em geral, os planos de compilação são definidos manualmente e um único plano de compilação é aplicado a todas as regiões do programa. Técnicas para a geração automática de planos de compilação também têm sido utilizadas. Entretanto, elas apenas selecionam, dentro de um conjunto fixo de planos de compilação, o melhor plano para uma determinada unidade de código. Este trabalho apresenta uma proposta de um mecanismo adaptativo para a seleção de planos de compilação em compiladores JIT. Esse mecanismo tem por objetivo permitir que o processo de seleção de planos de compilação adapte-se ao histórico de utilização de uma unidade de código e também a mudanças na plataforma de execução do programa.**

Keywords*—*otimização em compiladores, compiladores JIT, técnicas adaptativas

I. Introdução

Linguagens de programação como Java e C# têm como uma de suas principais características a portabilidade de seus programas, isto é, eles podem ser executados em qualquer dispositivo equipado com a máquina virtual correspondente. Entretanto, a portabilidade limita o formato de distribuição do programa para uma forma que é independente de qualquer processador ou sistema operacional. Para lidar com arquivos nesse formato, máquinas virtuais são usadas para a interpretação do programa ou compilação dinâmica antes da execução do programa. Como a execução interpretada de um programa é inerentemente lenta, é essencial que se utilize um modelo de compilação dinâmica ou compilação *Just-in-Time* (JIT) para atingir um desempenho em época de execução eficiente para tais programas [1].

A compilação JIT opera em época de execução o que contribui para o tempo total de execução do programa e, se realizada imprudentemente, pode piorar ainda mais a execução ou tempo de resposta do programa. Portanto, políticas de compilação JIT precisam ser cuidadosamente ajustadas para identificar *se* e *quando* diferentes regiões do programa são compiladas para atingir o melhor desempenho. Além disso, *como* compilar regiões do programa também é um importante componente de qualquer política de compilação.

Um dos componentes de uma política de compilação é o *plano de compilação*. Em um compilador otimizador, o plano de compilação é responsável por guiar as transformações aplicadas sobre um programa (ou parte dele) como o objetivo de melhorar determinada característica como, por exemplo, tempo de execução ou consumo de energia. As primeiras máquinas virtuais com compilação JIT utilizavam políticas estáticas de compilação. As máquinas virtuais modernas utilizam diversas técnicas para selecionar dinamicamente um subconjunto de código para compilação e também para selecionar o melhor plano de compilação. Tais técnicas são conhecidas na literatura como técnicas de otimização adaptativa [2]. Entretanto, apesar destas técnicas utilizarem o termo adaptativo, elas não modificam a estrutura ou o conjunto de regras a serem aplicadas pelo compilador JIT. Elas somente selecionam um determinado plano de compilação dentro de um conjunto de planos a partir das características da unidade de código que será compilada.

Este trabalho apresenta uma proposta de um mecanismo adaptativo, baseado em aprendizado por reforço, para a seleção de planos de compilação em compiladores JIT. O objetivo principal é que o processo de aprendizado seja contínuo e a escolha de um plano de compilação para uma unidade de código possa variar de acordo com seu histórico de utilização e arquitetura na qual ela está executando.

Este trabalho está dividido da seguinte forma: na seção II, baseada em [3], apresenta-se os principais conceitos de compiladores JIT e sua arquitetura genérica. Na seção III, baseada em [2], apresenta-se algumas técnicas disponíveis na literatura denominadas de otimização adaptativa. Na seção IV apresenta-se brevemente os conceitos da tecnologia adaptativa utilizados neste trabalho. Na seção V apresenta-se a proposta do mecanismo adaptativo. Por fim, na seção VI apresenta-se as conclusões e trabalhos futuros.

II. Compiladores JIT

Linguagens de programação que têm como um de seus objetivos permitir a portabilidade entre diferentes plataformas utilizam o modelo de máquinas virtuais para a execução de seus programas. Em geral, utilizam um modelo de interpretação de código para permitir a independência do *hardware* e do sistema operacional em que executam. Entretanto, código interpretados são mais lentos que códigos compilados. Um meio para se evitar a sobrecarga de interpretação é permitir ao interpretador gerar código de máquina para um segmento de código interpretado, antes que esse segmento de código seja necessário. Isto é chamado de compilação *Just-in-Time* (JIT) [4]. Uma vantagem dessa técnica é que o processo de compilação é ocultado do usuário, e que o código gerado pode ser adaptado para um processador particular. Ela possibilita ainda que o código compilado passe por um processo de otimização.

Esta técnica é adequada somente se o processo de compilação for rápido o suficiente para ser aceitável, já que o tempo de compilação está incluído no tempo total de execução do programa, pois a compilação ocorre durante sua execução.

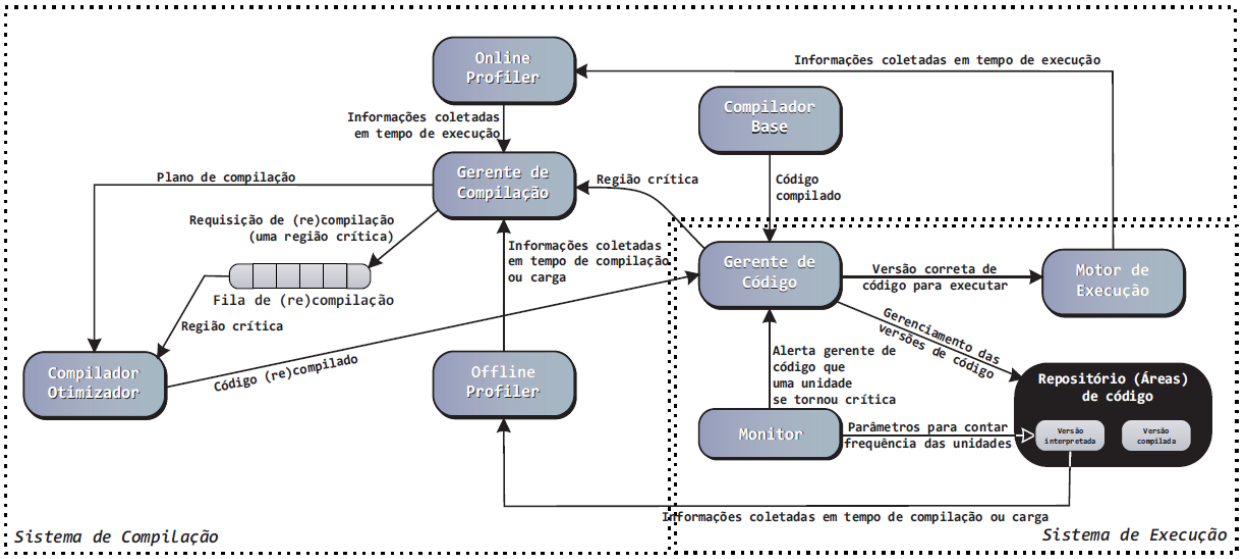


Figura 1. Arquitetura genérica de uma máquina virtual com compilação JIT. Extraída de [3].

Uma questão crítica é decidir *o que* e *quando* um segmento de código deve ser compilado e também *como* esse segmento de código deve ser compilado.

As primeiras máquinas virtuais com compilação JIT utilizavam políticas estáticas simples para decidir quais porções de código deviam ser compiladas e compilavam cada segmento de código com um conjunto fixo de otimizações [3]. Máquinas virtuais modernas utilizam diversas técnicas para selecionar dinamicamente um subconjunto de código para compilação e otimização. Elas somente compilam partes do código que contribuem significantemente para o tempo de execução do programa; esses códigos são chamados de regiões críticas (*hot spots*) do programa. Em alguns casos utilizam múltiplos níveis de qualidade na geração do código: uma compilação rápida produz código de qualidade moderada para a maior parte do código e, um código mais sofisticado, porém de compilação mais lenta, para regiões críticas mais importantes [4].

Os compiladores JIT podem ser classificados de duas formas em relação ao seu funcionamento: interpretação mais compilação ou compilação mais recompilação. Compiladores que utilizam a primeira forma interpretam todas as unidades de código do programa (funções ou métodos) e geram código nativo para regiões críticas com um compilador otimizador. Compiladores que utilizam a segunda forma compilam todas as unidades de código para código nativo utilizando um compilador base rápido e as regiões críticas são [3]: recompiladas apenas uma vez utilizando um compilador otimizador ou; recompiladas várias vezes de acordo com as políticas estabelecidas por um gerente de compilação.

A Figura 1, extraída de [3], apresenta os módulos de uma arquitetura genérica de uma máquina virtual com compilação JIT. A seguir apresenta-se uma breve descrição de cada um dos módulos.

O **compilador base** é responsável por gerar código nativo em época de execução para todas as unidades de código invocadas pelo sistema, antes da primeira invocação. Ele tem por objetivo gerar o código nativo de forma rápida, sendo assim, não aplica otimizações às unidades compiladas.

O **compilador otimizador** é um dos principais componentes da arquitetura apresentada. Ele é responsável por gerar código nativo em época de execução para as regiões críticas do programa. Em geral, todo código compilado é otimizado utilizando-se um conjunto de otimizações pré-definido. Esse conjunto de otimizações é descrito em um *plano de compilação*. As otimizações contidas no plano de compilação e o modo como elas são aplicadas satisfazem a decisão de projeto de cada desenvolvedor. Contudo, pode-se utilizar técnicas para ajustar automaticamente as configurações do compilador, alterando o conjunto de otimizações e também a ordem em que elas são aplicadas em uma determinada região crítica. Algumas destas técnicas são apresentadas na seção III.

As técnicas para ajuste automático das configurações do compilador utilizam informações de perfil das unidades de código, coletadas em época de compilação pelo *offline profiler* ou em época de execução pelo *online profiler*, para especializar o código com o objetivo de torná-lo mais eficiente.

O **offline profiler** tem por objetivo coletar informações em época de compilação ou de inicialização do programa. Essas informações são usadas pelo compilador otimizador para guiar o processo de geração de código.

O **online profiler** tem por objetivo coletar informações em época de execução do programa. Ele monitora a execução das unidades de código e coleta informações que possibilitam a geração de código mais especializado para essas unidades. Dentre as informações coletadas têm-se: tipos de variáveis, tamanho da unidade de código, quantidade de parâmetros e seus tipos, existência de chamadas aninhadas, existência de recursividade, tempo de interpretação, tempo de compilação e tempo de execução.

O **monitor** tem por objetivo atualizar os parâmetros de frequência que são usados para medir a frequência de execução das unidades de código do programa. Esses parâmetros são instrumentalizados nas unidades de código e podem ser de dois

tipos: *contadores* ou *fração de tempo*. Quando o parâmetro atinge um limite pré-definido, o monitor alerta ao gerente de código que a unidade de código se tornou frequente.

O **gerente de código** tem por objetivo gerenciar as áreas de código do programa. Ele executa basicamente duas tarefas: 1) envio da versão correta da unidade de código acionada para execução, priorizando o envio do código nativo mais atual; 2) envio de uma unidade de código para (re)compilação. O gerente de código interage com o monitor para identificar quais unidades de código são frequentes. Assim que elas são detectadas o gerente de código as envia para o gerente de compilação para que seja providenciada uma nova versão desta unidade.

O **gerente de compilação** tem por objetivo criar um *plano de compilação* para o compilador otimizador utilizar durante a geração de código. O gerente de compilação pode utilizar as informações obtidas pelo *offline* e/ou *online profilers* para a criação de um plano de compilação adequado para uma determinada unidade de código. Em geral, um plano de compilação define um conjunto de otimizações pré-determinado que é o mais adequado para uma determinada região de código. Além disto, esse plano pode definir um conjunto de otimizações mediante a análise do próprio programa em execução. Portanto, neste último caso as otimizações serão habilitadas durante a execução do programa, bem como a ordem em que elas serão aplicadas.

O **repositório de código** tem por objetivo armazenar versões de código interpretado e/ou compilado. Em sistemas do tipo interpreta e compila, o código interpretado é o padrão de execução. Nos sistemas que compilam e recompilam é somente mantida a área de código nativo. A medida que as regiões críticas são (re)compiladas a área de código nativo se expande. Em geral, a prioridade de execução é sempre do código especializado para o comportamento ativo.

O **motor de execução** tem por objetivo executar as unidades de código. Em sistemas do tipo interpretação mais compilação, ele é responsável por interpretar o código e também invocar o código nativo. Nos sistemas do tipo compilação mais recompilação ele é responsável apenas pela invocação do de código nativo.

III. Otimização Adaptativa em Compiladores JIT

O advento das arquiteturas de máquinas virtuais e o processo de compilação dinâmica introduziu novos desafios para se atingir um alto desempenho de execução dos programas. Para lidar com esses desafios, diversas pesquisas têm sido realizadas em tecnologia de otimização adaptativa [2]. Este tipo de tecnologia tem como objetivo melhorar o desempenho de execução dos programas através do monitoramento do seu comportamento e utilizando as informações coletadas em época de execução para direcionar decisões de otimização.

O termo *adaptativo* é utilizado nesta seção conforme definição dos autores dos trabalhos citados. Esse termo não está de acordo com a definição utilizada na Tecnologia Adaptativa. Na seção V apresenta-se um termo mais adequado, de acordo com [5], para este tipo de técnica de otimização adaptativa.

Em [2] os autores dividem as técnicas para otimização adaptativa em quatro categorias: 1) *otimização seletiva* são técnicas utilizadas para determinar quando e em quais partes do programas devem ser aplicadas otimizações em tempo de execução; 2) *profiling techniques for feedback-directed optimization (FDO)* são técnicas para coletar informações do perfil de execução dos programas com alta granularidade; 3) *feedback-directed code generation* são técnicas usadas utilizando as informações de perfil do programa para melhorar a qualidade do código gerado pelo compilador otimizador; 4) *outras técnicas FDO* que usam informações de perfil do programa para melhorar o seu desempenho.

O processo de *Feedback Directed Optimization* coleta as informações sobre o tempo de execução utilizando técnicas de perfil (*profile*) de execução do programa. A seguir, são citadas algumas destas técnicas:

- *Contadores de Desempenho*: Contadores são inseridos no código para detectar os caminhos utilizados com mais frequência. Esses contadores são incrementados no momento em que determinada parte do programa é executada.
- *Amostragem*: O sistema coleta um subconjunto representativo de uma classe de eventos, permitindo um limitado gasto de tempo para se obter informações do perfil do programa.
- *Monitoramento de Serviços*: Monitora os serviços requisitados pelo programa durante sua execução. São exemplos de serviços: a entrada e saída e o gerenciamento de memória.
- *Instrumentação*: Adição de código para coleta de informações.

Um sistema que utiliza otimização seletiva é composto basicamente de três componentes [2]: 1) um mecanismo de perfil para identificar regiões de código para otimização futura; 2) um componente de decisão para selecionar quais otimizações aplicar a cada região de código; 3) um compilador otimizador dinâmico para realizar as otimizações selecionadas. Em um sistema com otimização seletiva é possível aplicar para cada unidade de código um conjunto diferente de otimizações, dependendo das informações de perfil coletadas.

O componente de decisão em um sistema de otimização seletiva utiliza heurísticas para guiar o processo de transformações do código para um determinado objetivo. Em geral, essas heurísticas são definidas manualmente pelos desenvolvedores do compilador. Este processo é caro, consome muito tempo e é propenso a erros, e pode levar a um desempenho sub-ótimo [6]. Para lidar com tais questões, têm-se utilizado aprendizado de máquina em compiladores com o objetivo de automatizar o processo de geração de heurísticas [7], [8].

Um dos primeiros trabalhos utilizando aprendizado de máquina em compiladores foi para lidar com transformações *looping unrolling* [9]. Este trabalho utilizou o compilador GNU FORTRAN alterando-se a heurística de ativação de *loop unrolling* para utilizar um modelo aprendido através de exemplo. O algoritmo de aprendizado utilizado era baseado

em árvores de decisão, o que permitia que o modelo gerado pudesse ser interpretado por um especialista.

Cavazos e O'boyle [7] aplicaram a técnica de *logistic regression* para habilitar ou desabilitar transformações em planos de compilação na Jikes RMV para a compilação específica de método. Estudos apresentados mostraram que as transformações poderiam obter melhores resultados se fossem selecionados planos de compilação específicos para cada método do programa. A Jikes RVM foi modificada para compilar métodos individuais variando as transformações aleatoriamente, uma medição de tempo foi adicionada a cada método e os planos de compilação com melhor tempo de execução foram selecionados e usados para o treinamento da *logistic regression*. A versão utilizando aprendizado de máquina superou a configuração padrão presente na Jikes RVM.

Hoste, Georges e Eeckhout [6] apresentam uma proposta de sintonia automática de planos de compilação em um compilador JIT baseada em uma busca evolucionária com múltiplos objetivos. A ideia é aplicar uma sintonia fina no compilador para cenários específicos: uma dada plataforma de *hardware*, um conjunto de aplicações, ou um conjunto de entradas para aplicações de interesse. A sintonia inicia com uma exploração dos planos de compilação para descobrir quais deles são ótimo de Pareto (*Pareto-optimal*), e então um subconjunto deles é atribuído ao compilador JIT.

Sanchez et al. [8] aplicaram a técnica de *Support Vector Machine* (SVM) para habilitar ou desabilitar transformações em planos de compilação em uma máquina virtual de grande porte, a Testarossa da IBM, para a compilação específica de método. O processo de modificação dos planos de compilação partia do plano original da Testarossa realizando pequenas alterações ao longo do tempo em busca de melhorias locais. O desempenho de vazão não sofreu alterações, mas foi possível melhorar o desempenho de inicialização.

Kulkarni e Cavazos [10] apresentaram uma técnica que automaticamente seleciona a melhor ordem prevista de transformações para métodos diferentes de um programa. A técnica utiliza uma rede neural artificial para predizer a ordem de transformações que é mais benéfica para um determinado método. As redes neurais artificiais são automaticamente inferidas usando *NeuroEvolution for Augmenting Topologies* (NEAT). A técnica foi implementada na máquina virtual Jikes RVM e apresentou melhorias de desempenho em um conjunto de *benchmarks* se comparado com a aplicação de transformações em uma ordem fixa.

Leather, Bonilla e O'boyle [11] apresentaram um mecanismo para automaticamente buscar características de um programa que tem um maior impacto na qualidade de heurísticas de aprendizado de máquina. O espaço de características é descrito por uma gramática e é então percorrido com programação genética. Esse mecanismo foi aplicado para a transformação *loop unrolling* no compilador GCC.

IV. 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 [12]. 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.

Um dispositivo adaptativo é constituído de: 1) uma componente não-adaptativa, na forma de um dispositivo subjacente guiado por regras, o qual serve como base para a construção da versão adaptativa; 2) um mecanismo adaptativo acoplado ao dispositivo subjacente, a qual é capaz de alterar o conjunto de regras que define o comportamento desse dispositivo subjacente [13].

O mecanismo adaptativo é composto por um conjunto de funções especiais, denominadas *funções adaptativas*, que são acionadas no momento da aplicação de alguma regra definida pelo dispositivo adaptativo. As funções adaptativas é que possibilitam uma forma de alterar o conjunto de regras do dispositivo adaptativo.

As regras adaptativas são formuladas associando-se a uma regra não-adaptativa do dispositivo subjacente duas funções adaptativas. Uma função adaptativa a ser executada antes e a outra depois da mudança de configuração do dispositivo. Caso não se queira executar nenhuma ação antes e/ou depois da mudança de configuração, associa-se a regra não-adaptativa uma função adaptativa nula.

Durante a operação do dispositivo adaptativo, uma vez escolhida uma regra adaptativa a ser aplicada, executa-se inicialmente a função adaptativa anterior, aplica-se a regra a ela associada, e por fim, executa-se a função adaptativa posterior.

Na próxima seção, apresenta-se um mecanismo com adaptatividade básica para a seleção de planos de compilação em compiladores JIT.

V. O Mecanismo Adaptativo

Em geral, os planos de compilação utilizados pelo compilador otimizador em compiladores JIT são definidos de forma estática e aplicados em todas as unidades de código que são compiladas. Com a utilização de técnicas de otimização adaptativa, apresentadas na seção III, os planos de compilação são selecionados dinamicamente a partir das características da unidade de código, através de informações de seu perfil de utilização. Entretanto, essas técnicas ditas adaptativas não alteram o comportamento do dispositivo, isto é, do compilador JIT. Elas apenas selecionam, dentro de um conjunto fixo de planos de compilação, o melhor plano para uma determinada unidade de código. Portanto, de acordo com a terminologia apresentada em [5], essas técnicas ditas adaptativas seriam melhor definidas como técnicas de um *processo configurável*.

Mesmo as técnicas que utilizam aprendizado de máquina podem ser classificadas como configuráveis. Em geral, os trabalhos relacionados ao aprendizado de máquina em compiladores utilizam técnicas de aprendizado supervisionado [14]. Com o uso destas técnicas, o aprendizado é feito através de exemplos fornecidos por algum supervisor externo em uma fase de treinamento. No caso do uso de aprendizado supervisionado em compiladores, o agente aprendiz é treinado para selecionar as melhores otimizações em relação à plataforma de execução

e ao conjunto de programas usados para o treinamento. Caso se queira utilizar uma nova plataforma de execução ou o perfil de utilização do programa se altere, o processo de treinamento deve ser refeito. Portanto, após o processo de treinamento inicial, é sempre selecionado, para uma determinada unidade de código, o mesmo plano de compilação.

A proposta deste trabalho é de um mecanismo adaptativo que permite o aprendizado contínuo na seleção de planos de compilação, isto é, permite que seja selecionado, para uma determinada unidade de código, diferentes planos de compilação ao longo do tempo, dependendo não somente das características da unidade de código a ser compilada como também do seu histórico de execução e dos planos de compilação usados anteriormente.

A realização deste mecanismo adaptativo é feita utilizando um modelo baseado em aprendizado por reforço [15] [16]. Aprendizado por reforço é uma forma de aprendizado de máquina em que um agente aprende através de sua interação com um ambiente. Ele permite que o aprendizado seja contínuo, através de um processo de tentativa e erro. O agente não é ensinado por meio de exemplos fornecidos por um supervisor externo. O uso de aprendizado por reforço em compiladores pode permitir que o processo de aprendizado seja contínuo e adaptável ao perfil de execução do programa ou também a mudanças de ambiente como, por exemplo, novas plataformas de *hardware* ou sistema operacional.

A Figura 2 apresenta os componentes básicos do dispositivo adaptativo. A camada subjacente é constituída pelo compilador JIT e os módulos apresentados na seção II. A camada adaptativa é constituída por um agente aprendiz, baseado em aprendizado por reforço, responsável selecionar o melhor plano de compilação para uma determinada unidade de código. Ao selecionar um plano de compilação, o mecanismo adaptativo pode alterar o plano de compilação previamente especificado no compilador JIT, alterando assim a regra de sua aplicação.

Um componente de *Controle* é adicionado ao *Compilador JIT*, na camada subjacente, com o objetivo de criar um mecanismo de interação entre o compilador JIT e o mecanismo adaptativo. Isso permite reduzir a quantidade de alterações necessárias no compilador JIT para o acoplamento do mecanismo adaptativo.

Os estímulos de entrada do dispositivo subjacente são as informações coletadas em época de compilação, pelo módulo *offline profiler*, e também pelas informações que são coletadas em época de execução, pelo módulo *online profiler*. No momento em que o compilador JIT identifica uma unidade de código que deve ser (re)compilada, ele aciona o componente *Controle*, que então passa as informações recebidas para o mecanismo adaptativo. Neste momento, o acionamento do mecanismo adaptativo é equivalente à chamada de uma função adaptativa anterior.

O mecanismo adaptativo então cria um determinado plano de compilação a partir das informações recebidas e de sua configuração atual e envia esse plano para o *Controle* que é responsável por alterar a regra de aplicação do plano de compilação no compilador JIT. O compilador JIT então compila a unidade de código de acordo com o plano de compilação recebido. Após a unidade de código ser executada, são obtidas informações de perfil dessa execução, que são

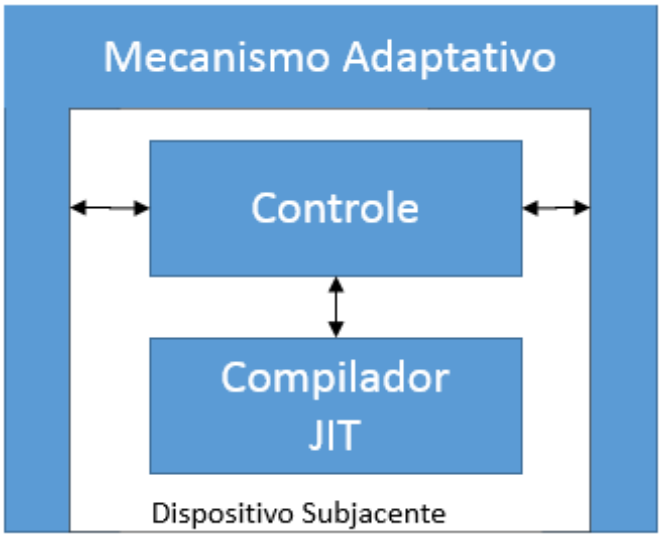


Figura 2. Componentes básicos do mecanismo adaptativo.

enviadas para o *Controle* e deste para o mecanismo adaptativo. Essas informações são utilizadas para atualizar as regras de inferência do modelo baseado em aprendizado por reforço e é equivalente à chamada de uma função adaptativa posterior.

VI. Conclusão

Este trabalho apresentou uma proposta de um mecanismo adaptativo, baseado em aprendizado por reforço, para a seleção de planos de compilação em compiladores JIT. O objetivo deste mecanismo é realizar um processo de aprendizado contínuo na seleção do melhor plano de compilação para uma determinada unidade de código permitindo que o processo de seleção se adapte ao histórico de utilização da unidade de código ou a mudanças na plataforma de execução do programa.

Como trabalhos futuros pretende-se implementar o mecanismo adaptativo utilizando a máquina virtual Jikes RVM [17] e realizar experimentos para a avaliação do modelo de aprendizado proposto com o objetivo de verificar se com o uso do mecanismo adaptativo houve alguma redução do tempo de execução das unidades de código e/ou do tempo de execução global do programa.

Referências

[1] P. A. Kulkarni, "Jit compilation policy for modern machines," *ACM SIGPLAN Notices*, vol. 46, no. 10, pp. 773–788, 2011.

[2] M. Arnold, S. J. Fink, D. Grove, M. Hind, and P. F. Sweeney, "A survey of adaptive optimization in virtual machines," *Proceedings of the IEEE*, vol. 93, no. 2, pp. 449–466, 2005.

[3] G. S. Oliveira and A. F. da Silva, "Compilação just-in-time: Histórico, arquitetura, princípios e sistemas," *Revista de Informática Teórica e Aplicada*, vol. 20, no. 2, pp. 174–213, 2013.

[4] D. Grune, K. van Reeuwijk, H. E. Bal, C. J. Jacobs, and K. Langendoen, *Modern Compiler Design*, 2nd ed. New York, NY, USA: Springer Science & Business Media, 2012.

[5] J. J. NETO, "Um glossário sobre adaptatividade," in *3º Workshop de tecnologia Adaptativa–WTA, São Paulo*, 2009.

[6] K. Hoste, A. Georges, and L. Eeckhout, "Automated just-in-time compiler tuning," in *Proceedings of the 8th Annual IEEE/ACM International Symposium on Code Generation and Optimization*, ser. CGO '10. New York, NY, USA: ACM, 2010, pp. 62–72.

[7] J. Cavazos and M. F. O'boyle, "Method-specific dynamic compilation using logistic regression," *ACM SIGPLAN Notices*, vol. 41, no. 10, pp. 229–240, 2006.

[8] 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.

[9] 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*. London, UK: Springer-Verlag, 2002, pp. 41–50.

[10] S. Kulkarni and J. Cavazos, "Mitigating the compiler optimization phase-ordering problem using machine learning," *ACM SIGPLAN Notices*, vol. 47, no. 10, pp. 147–162, 2012.

[11] H. Leather, E. Bonilla, and M. O'boyle, "Automatic feature generation for machine learning–based optimising compilation," *ACM Transactions on Architecture and Code Optimization*, vol. 11, no. 1, pp. 14:1–14:32, Feb. 2014.

[12] 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).

[13] J. J. NETO, "Adaptatividade: Generalização conceitual," in *3º Workshop de tecnologia Adaptativa–WTA, São Paulo*, 2009.

[14] T. M. Mitchell, *Machine Learning*. New York, NY, USA: McGraw-Hill, 1997.

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

[16] A. S. Mignon and R. L. A. Rocha, "epsilon-greedy adaptativo," in *Memórias do VIII Workshop de Tecnologia Adaptativa – WTA 2014*, 2014, pp. 57–62.

[17] M. Arnold, S. Fink, D. Grove, M. Hind, and P. F. Sweeney, "Adaptive optimization in the jalapeño jvm," *ACM SIGPLAN Notices*, vol. 35, no. 10, pp. 47–65, Oct. 2000.

Parte IV

Transcrição da mesa redonda

A transcrição da mesa redonda está em fase de revisão. Em breve, esta seção será substituída pelo texto adequado.