MEMÓRIAS DO WTA 2009

TERCEIRO 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
2009

WTA 2009 – Terceiro Workshop de Tecnologia Adaptativa

MEMÓRIAS DO WTA 2009

TERCEIRO 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
2009

FICHA CATALOGRÁFICA

Workshop de Tecnologia Adaptativa (3. : 2009 : São Paulo)
Memórias do WTA'2009 : terceiro Workshop de Tecnologia Adaptativa / Laboratório de Linguagens e Técnicas Adaptativas. -- São Paulo : s.n., 2009.
p.

ISBN 978-85-86686-51-1

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. Laboratório de Linguagens e Técnicas Adaptativas II. t.

CDD 004

SUMÁRIO

TUTORIAL

COMUNICAÇÕES

SHORT PAPERS

APRESENTAÇÃO

WTA'2009

A terceira edição do WTA realizou-se em São Paulo, BRASIL, nos dias 29 e 30 de janeiro de 2009, na Escola Politécnica da Universidade de São Paulo, por meio do Departamento de Engenharia de Computação e Sistemas Digitais.

As contribuições encaminhadas na forma de artigos relacionados à Tecnologia Adaptativa, nas seguintes áreas, abrangeram, de forma não exclusiva, os tópicos abaixo:

TÓPICOS

- Teoria da Computação
- Autômatos
- Computação inspirada na biologia
- Compiladores
- Gramáticas
- Processamento de Linguagem Natural
- Jogos eletrônicos
- Tomada de decisões
- Inferência Gramatical

COMISSÃO DE PROGRAMA

Almir Rogério Camolesi (Assis, SP, Brasil)
Amaury Antônio de Castro Junior (Coxim, MS, Brasil)
Aparecido Valdemir de Freitas (São Caetano do Sul, SP, Brasil)
César Alberto Bravo Pariente(São Paulo, SP, Brasil)
Hemerson Pistori (Campo Grande, MS, Brasil)
Marcus Vinicius Midena Ramos (São Paulo, SP, Brasil)

COMISSÃO ORGANIZADORA

André Riyuiti Hirakawa (São Paulo, SP, Brasil)
Cinthia Itiki (São Paulo, SP, Brasil)
Italo Santiago Vega (São Paulo, SP, Brasil)
João José Neto, Chair (São Paulo, SP, Brasil)
Ricardo Luis de Azevedo da Rocha (São Paulo, SP, Brasil)

Memórias do WTA 2009: Terceiro Workshop de Tecnologia Adaptativa

R. L. A. Rocha

Resumo — **Esta publicação do Laboratório de Linguagens e Técnicas Adaptativas do Departamento de Engenharia de Computação e Sistemas Digitais da EPUSP é uma coleção de textos produzidos para o WTA 2009, o Terceiro Workshop de Tecnologia Adaptativa, realizado em São Paulo nos dias 29 e 30 de Janeiro de 2009. A exemplo da edição de 2008, este evento contou com uma presença maciça 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.**

Palavras-chave — **Adaptatividade, Autômatos adaptativos, Comportamento Auto-modificável, Sistemas Adaptativos, Tecnologia Adaptativa.**

I. INTRODUÇÃO

Com muita satisfação apresentamos neste documento estas memórias com os artigos apresentados no WTA 2009 – Terceiro Workshop de Tecnologia Adaptativa, realizado na EPUSP nos dias 29 e 30 de janeiro de 2009.

Constatou-se o terceiro sucesso desse evento pela efetiva participação de uma centena de pesquisadores, pelo recebimento de 30 artigos, dos quais 5 foram selecionados completos e 8 resumos. Os artigos mais bem avaliados foram aprimorados por seus autores para publicação na revista IEEE Latin American Transactions.

Na presente edição do WTA observaram-se algumas evoluções em relação à do ano anterior:

- **Chamada aberta de trabalhos**: facilitou-se assim o envolvimento e a aproximação de interessados externos;
- **Diversificação dos participantes e dos trabalhos:** foram recebidas e aceitas excelentes contribuições técnicas, oriundas de fora de S.Paulo e do Brasil;
- **Prestigiosos apoios:** da SBC (Sociedade Brasileira de Computação), SPC (Sociedad Peruana de Computación), IEEE (The Institute of Electrical and Electronics Engineering), EPUSP (Escola Politécnica da USP) e PCS (Dep. de Eng. de Computação e Sistemas Digitais).

II. Esta Publicação

Estas memórias espelham o conteúdo apresentado no WTA 2009.

R. L. A. Rocha é pesquisador do Laboratório de Linguagens e Técnicas Adaptativas, Escola Politécnica da USP, São Paulo/SP, Brasil; fone: 55 11-3091-5583; e-mail: luis.rocha@poli.usp.br.

As quatro seguintes áreas dominaram o WTA 2009:

- **Algoritmos**
- **Ambiente de execução adaptativo**
- **Aplicações on-line**
- **Processamento de linguagem natural**

A exemplo do que foi feito no ano anterior, todo o material referente aos trabalhos apresentados no evento estarão acessíveis no portal do WTA 2009, 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.

III. 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 2010.

AGRADECIMENTO

Às instituições que apoiaram o WTA 2009, à comissão organizadora, ao pessoal de apoio e a tantos colaboradores voluntários, cujo auxílio propiciou o êxito que tivemos a satisfação de observar.

São Paulo, fevereiro de 2009.
Prof. Dr. Ricardo Luis de Azevedo da Rocha
LTA – PCS - EPUSP

Créditos

São muitos os que deram ao nosso evento um auxílio decisivo, que viabilizou sua realização e permitiu que tivesse todo o êxito que apresentou.

Nosso agradecimento inicial a todos os que, de modo formal, institucionalmente apoiaram o WTA 2009:

- SBC – Sociedade Brasileira de Computação;
- SPC – Sociedade Peruana de Computação;
- IEEE – The Institute of Electrical and Electronics Engineering;
- EPUSP – Escola Politécnica da USP;
- PCS – Dep. de Engenharia de Computação e Sistemas Digitais.

Gostaríamos de agradecer ao IEEE, através do Profa. Dra. Mirela Sechi Moretti Annoni Notare, pela parceria com o nosso evento, e pela publicação dos melhores trabalhos na revista IEEE Latin American Transactions.

Agradecemos também, à Microsoft, que, através da pronta ação do Prof. Dr. Jorge Luís Risco Becerra, prestou ao evento uma importante colaboração mediante a cessão dos equipamentos utilizados na mostra de software do evento.

Registramos a seguir nosso reconhecimento público nominal àqueles que, direta ou indiretamente, contribuíram, com suas idéias e seu trabalho dedicado, para a organização e para a viabilização final desta edição do WTA (ordem alfabética):

- Prof. Dr. Almir Rogério Camolesi, a quem devemos: a divulgação do evento; a coleta das inscrições para o evento; a organização dos horários das apresentações; a criação e manutenção do novo portal do WTA e pela participação no tutorial deste ano;
- Prof. Amaury Antônio de Castro Júnior, a quem devemos: a gestão junto à SBC para apoio ao evento; a atualização do conteúdo do portal do LTA, especialmente das publicações; a criação do novo portal do LTA, com recursos mais automáticos que permitirão uma atualização mais rápida e segura;
- Prof. Dr. João José Neto, pela competência com que aceitou nosso convite para proferir o mini-curso sobre os aspectos conceituais da Adaptatividade e sua generalização com o qual tantos elogios logrou ganhar de todos os participantes

Agradecemos finalmente a boa vontade da qual resultou o importante auxílio prestado ao WTA 2009 pelos seguintes profissionais (ordem alfabética):

- Leia Sicília, que secretariou o evento, pela permanente alegria e boa vontade que demonstrou, antes e durante seus trabalhos na recepção do WTA 2009;
- Nilton Araújo do Carmo, pelos trabalhos de apoio técnico prestados durante o evento, inclusive pela gravação de todas as imagens dos trabalhos do WTA 2009;
- Arqt. Regina Gauer José, pela graça que deu ao evento através do seu trabalho de criação e execução artística dos cartazes utilizados na sua divulgação;

TUTORIAL

Um Glossário sobre Adaptatividade

J. J. Neto

Resumo — **Este material complementa o tutorial sobre adaptatividade e temas afins, que foi apresentado em São Paulo, em 29 de janeiro de 2009, por ocasião do WTA 2009 – Terceiro Workshop de Tecnologia Adaptativa. Procura concentrar os principais conceitos, e apresentar uma versão atualizada da terminologia técnica referente ao tema, levando em consideração os mais recentes avanços científicos e tecnológicos da área. Pretende-se com isso proporcionar, aos interessados na adaptatividade, um panorama atualizado da terminologia em uso, visando à uniformização do vocabulário empregado nas futuras publicações dessa área.**

Palavras-chave — **Adaptatividade, Dispositivos Adaptativos, Comportamento Auto-modificável, Sistemas Adaptativos, Tecnologia Adaptativa.**

IV. *Conceitos e terminologia*

Na pesquisa empreendida pelo LTA, desde sua criação, o tema de interesse central tem sido o estudo da teoria e das aplicações de dispositivos e de sistemas que apresentam comportamento variável, e, em especial, daqueles conhecidos como adaptativos os quais exibem comportamento autonomamente auto-modificável.

A implementação de tais sistemas é realizada, geralmente, na forma de softwares preparados para exibirem alterações dinâmicas em sua funcionalidade.

Tais alterações funcionais são as responsáveis pela realização computacional desses comportamentos dinâmicos, autonomamente auto-modificáveis, ou seja, do conceito da adaptatividade.

Há muitos estudos de sistemas e dispositivos com 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.

A finalidade desta seção é a de discutir o significado de alguns termos técnicos, muito utilizados nas publicações sobre tecnologia adaptativa, mas cujo emprego correto tem apresentado, com freqüência, dúvidas e mal-entendidos.

O texto que segue relata resultados dos últimos esforços efetuados em uma tentativa de esclarecer o significado dos principais desses termos, em busca de uma padronização da terminologia, através de uma proposta concreta de uniformização do uso de algumas das palavras e expressões mais problemáticas.

A. *Adaptatividade, Autômatos e Tecnologia Adaptativa*

O foco principal da terminologia a que se refere este particular estudo reside no comportamento variável dos softwares, e softwares auto-modificáveis são os que maior interesse apresentam para essa área.

João José Neto é professor associado do Departamento de Engenharia de Computação e Sistemas Digitais da EPUSP, e foi o coordenador geral do evento WTA 2008 – Segundo Workshop de Tecnologia Adaptativa. (joao.jose@poli.usp.br) fone: (11) 3091-5402

A implementação de tais sistemas de software costuma ser feita através de programas que não tenham impedimentos para efetuarem autonomamente modificações em sua própria funcionalidade.

A atenção terminológica principal neste trabalho está, por isso, focalizada no significado associado às palavras que designam fenômenos, técnicas e métodos ligados à alteração dinâmica do comportamento de programas auto-modificáveis.

Convém identificar, como causas de freqüentes deslizes terminológicos amiúde constatados em publicações sobre o assunto, certas ocorrências historicamente relacionadas a uma série de mal-entendidos quanto aos significados de alguns termos técnicos muito usados em publicações sobre a adaptatividade.

ADAPTATIVIDADE E AUTÔMATOS ADAPTATIVOS [1,2,4] – O primeiro grande mal-entendido é a errônea identificação entre o estudo do conceito de adaptatividade e o estudo de certos temas fundamentais da teoria da computação, como é o caso dos autômatos, em particular, dos autômatos adaptativos.

Pelo simples fato de os autômatos adaptativos terem sido os primeiros dispositivos adaptativos a surgirem e a serem objetos de estudo e publicação, provavelmente essa tenha sido a causa mais remota desse equívoco e de muitas de suas implicações.

Uma seqüela nefasta advinda desse mal-entendido pode ser facilmente identificada notando-se a maneira como tantos potenciais usuários de sistemas automodificáveis têm se mantido sistematicamente à margem da área.

Essa totalmente desnecessária rejeição do emprego da adaptatividade e de suas vantagens talvez ocorra porque desconheçam, não dominem ou não tenham afinidade com as bases teóricas da computação, particularmente com autômatos e linguagens formais.

Neste ponto, convém deixar bem claro que basta o bom-senso para que se possa identificar a conveniência de um comportamento dinamicamente variável em uma aplicação, sendo totalmente dispensável para isso um conhecimento muito profundo da teoria da computação.

Para a aplicação das técnicas e métodos baseados no conceito da adaptatividade, basta que se tenha um certo domínio da atividade de modelagem, e para isso, se esteja minimamente à vontade com algum tipo de formulação matemática com a qual se possa desenvolver modelos rigorosos das aplicações desejadas.

Nessa ocasião, pode-se então buscar, na tecnologia que nasceu dos fundamentos matemáticos da adaptatividade, recursos tecnológicos para a elaboração de soluções modernas e criativas para as aplicações desejadas.

Já para compreender em profundidade as bases científicas em que se apóiam todos os fundamentos da adaptatividade, certamente se torna conveniente que se tenha uma bagagem conceitual menos superficial sobre cada assunto que seja fundamento das atividades ligadas à computação e aos computadores, especialmente: matemática discreta, linguagens formais, autômatos, modelos de computação, computabilidade e complexidade computacional.

AUTÔMATOS (DE PILHA ESTRUTURADOS) ADAPTATIVOS [1,2] – Outra confusão muito freqüente é a que envolve as designações "autômato adaptativo" e "autômato de pilha estruturado adaptativo". Embora ambas tenham sido usadas com exatamente o mesmo significado, encontram-se, nas publicações, poucas ocorrências da forma longa, cujo uso provavelmente tenha sido evitado não apenas por causa de

sua extensão, como também por não ter sido muito empregada nos primeiros textos publicados sobre o assunto.
Na época das publicações mais antigas sobre tais autômatos, não se suspeitava que poderia haver necessidade do nome completo para distinguir esse dispositivo de tantos outros, que um dia estariam sendo também utilizados. Dessa forma, a história consagrou o termo "autômato adaptativo" como sinônimo de "autômato de pilha estruturado adaptativo".
Para designar outros autômatos adaptativos que não sejam derivados dos autômatos de pilha estruturados, recomenda-se usar, por exemplo, nomes tais como "autômato finito adaptativo", procurando com isso conservar, para o termo "autômato adaptativo" o uso consagrado pela tradição.
TECNOLOGIA ADAPTATIVA [4] – 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.
Estão atualmente em uso diversos procedimentos, métodos, técnicas e ferramentas, centradas no conceito da Adaptatividade, e que constituem partes integrantes da crescente Tecnologia Adaptativa.
O alcance da Tecnologia Adaptativa é, portanto, muitíssimo mais amplo que o dos simples Autômatos Adaptativos, ou de qualquer outro artefato conceitual adaptativo particular.
Portanto, a Tecnologia Adaptativa não deve de forma alguma ser identificada com qualquer particular dispositivo adaptativo, já que cada um deles não passa de uma particular instância do formalismo geral em que foi desenvolvida a sua formulação.
Os estudos teóricos desenvolvidos no âmbito dos fundamentos da Tecnologia Adaptativa não se restringem, como muitos acreditam, apenas aos autômatos, estendendo-se também a gramáticas, programas, tabelas, e muitos outros dispositivos adaptativos.
É preciso notar ainda que nem qualquer uma dessas formulações em particular, nem todo o seu conjunto, nem mesmo aliados às aplicações que deles se desenvolveram, são suficientes para formar o que se denomina "Tecnologia Adaptativa".
A Tecnologia Adaptativa abrange certamente tudo isso, mas compreende, adicionalmente, todo o universo das técnicas, métodos, ferramentas e aplicações que encontram na Adaptatividade as raízes de seu funcionamento diferenciado.
Dessa maneira, são consideradas pertencentes ao âmbito da Tecnologia Adaptativa não apenas atividades especificamente ligadas à área de computação, mas também as de qualquer outra área de interesse, desde que em sua concepção fundamental esteja fortemente atuante o conceito da Adaptatividade.
ADAPTATIVIDADE E DISPOSITIVO ADAPTATIVO [3,4] **–** A presença dessa adaptatividade obviamente independe de qualquer formalismo especial que tenha sido empregado para sua formulação ou representação, tenha ela sido feita na forma de um autômato, de um conjunto de cláusulas declarativas, de um programa de computador, de uma tabela de decisões, ou de qualquer outra.
Apesar da obviedade dessa afirmação, persiste a infundada crença de que a adaptatividade não poderia ser obtida, aplicada ou utilizada sem que fosse projetado, construído e empregado um autômato ou algum outro dispositivo adaptativo, e que isso obrigatoriamente exigiria que seus autores e usuários fossem profundos conhecedores das teorias, métodos e técnicas associadas a tais formalismos.
Torna-se por isso necessário esclarecer que os autômatos, assim como tantos outros formalismos em uso, representam apenas casos particularíssimos dentre os inúmeros dispositivos passíveis de incorporarem características de auto-modificação autônoma, e que, portanto, sua utilização na implementação da adaptatividade em aplicações computacionais não passa de uma simples conveniência conjuntural.
O que mais importa, em suma, é a variabilidade dinâmica do comportamento do sistema ou do programa, e, caso se deseje modelar essa característica com a ajuda de qualquer formulação matemática específica cabível (como é o caso dos autômatos), isso é mera circunstância secundária, tendo-se total liberdade para fazê-lo, desde que a atenção não se desvie da adaptatividade que se deseja implementar.

V. Glossário

Apresenta-se neste glossário o significado e o uso de diversos termos técnicos específicos da área, que, por diversas razões, apresentam-se às vezes com conotações variadas, aplicados em situações não muito claras nem homogêneas, podendo isso dar margem a dúvidas quanto ao significado exato dos textos em que são empregados.
Os sentidos associados a tais termos são comentados, um de cada vez, procurando-se definir da forma mais rigorosa possível o significado de cada um no contexto das publicações sobre tecnologia adaptativa, sempre de acordo com a acepção adotada no âmbito das pesquisas realizadas no LTA [4].

DISPOSITIVO – refere-se a qualquer artefato abstrato que, a partir de uma configuração inicial conhecida e fixa, opera migrando sucessivamente de uma configuração para outra, em resposta a sucessivos estímulos de entrada recebidos, e de acordo com um conjunto de regras que define seu comportamento.
O termo "dispositivo" refere-se, portanto, a qualquer abstração que descreva um fenômeno, um programa, um processo ou algo similar, cuja operação ou comportamento dependa exclusivamente dessas mudanças sucessivas de configuração.
Neste sentido, portanto, pode ser tratado como dispositivo qualquer programa, qualquer fenômeno, qualquer processo que se consiga descrever como coleção de cláusulas do tipo "ao receber um estímulo X na configuração Y, a nova configuração passa a ser Z".
CONFIGURAÇÃO – define-se como sendo a configuração de um dispositivo, em algum momento de sua operação, uma representação, naquele momento, do conteúdo de todos os elementos variáveis relevantes à operação do dispositivo.
A configuração de um dispositivo compreende, portanto, todos os elementos modificáveis que o compõem, incluindo: os valores de todas as suas variáveis de estado, os conteúdos de todos os elementos de memória do dispositivo, a situação instantânea de todos os seus elementos dinâmicos, incluindo topologias, regras e algoritmos que possam ser alterados em operação, a situação de utilização de seus estímulos de entrada, áreas de trabalho, suas saídas já realizadas, etc.
CONFIGURAÇÃO INICIAL – corresponde à configuração na qual o dispositivo deve ser encontrado por ocasião do início de seu funcionamento. Deve ser sempre a mesma, e compreender a cadeia de entrada completa, variáveis de estado e memórias auxiliares, todas em seus valores iniciais pré-estabelecidos.
CONFIGURAÇÃO FINAL – é aquela atingida pelo dispositivo ao término de sua operação.
DISPOSITIVOS ACEITADORES – em dispositivos aceitadores, a configuração final costuma ser indicativa de êxito ou de fracasso do dispositivo em atingir alguma meta preestabelecida.
Assim, em aplicações à triagem de cadeias, caso o dispositivo tenha determinado que a cadeia de entrada atende às convenções consideradas aceitáveis, então ela será aceita, caso contrário, será rejeitada.
Estando o dispositivo em alguma de suas configurações finais de aceitação, em geral a cadeia de entrada deverá estar esgotada, e o conteúdo das variáveis de estado e das memórias auxiliares deverá atender a alguma convenção preestabelecida.
Em configurações de rejeição, ao menos uma de tais condições não será atendida.

DISPOSITIVOS CLASSIFICADORES – são aqueles em que a triagem tem por finalidade a classificação das cadeias de entrada de acordo com múltiplos critérios.
Haverá assim, neste tipo de dispositivos, diversas configurações finais de aceitação, cada qual correspondente ao atendimento de uma das múltiplas condições estabelecidas. Nesse caso, a rejeição da cadeia somente ocorre quando a cadeia não for capaz de atender a qualquer dos critérios de classificação válidos para esse dispositivo.
ESTÍMULO OU SÍMBOLO DE ENTRADA – cada um dos elementos de entrada capazes de provocar uma mudança de configuração no dispositivo é chamado estímulo.
ALFABETO DE ENTRADA – é o conjunto de todos os possíveis estímulos que podem ser utilizados por um dado dispositivo.
CADEIA DE ENTRADA – é qualquer seqüência de estímulos que alimente o dispositivo, nele provocando mudanças de configuração pela aplicação das leis codificadas no conjunto de regras que define o comportamento do dispositivo.
REGRA – lei que preside as mudanças de configuração de um dispositivo, a partir de sua configuração corrente, em resposta a um estímulo recebido.
Encontrando-se em certo instante em uma dada configuração, e recebendo algum estímulo de entrada, o dispositivo migra para uma nova configuração, que é determinada com base exclusiva no estímulo recebido e no conjunto de regras condicionais, do tipo "**se ... então**", que define completamente o comportamento do dispositivo.
Note-se que, em cada diferente configuração, a reação do dispositivo a um particular estímulo recebido pode ser diferente, já que as regras aplicadas determinam os movimentos não só a partir dos estímulos recebidos como também da configuração corrente do dispositivo.
Como todo dispositivo tem seu comportamento regido por um conjunto dessas leis, cada regra corresponde a uma descrição parcial desse comportamento. Somente o conjunto de todas as regras é que determina de forma completa o comportamento do dispositivo.
MOVIMENTO OU PASSO DE OPERAÇÃO – por movimento entende-se o efeito da aplicação de uma das regras que definem o funcionamento do dispositivo, e cada movimento do dispositivo é também chamado passo de operação desse dispositivo.
Assim sendo, cada movimento realizado pelo dispositivo é determinado por alguma regra, cuja aplicação, a partir da configuração corrente do dispositivo e do estímulo recebido, determina a sua próxima configuração.
OPERAÇÃO DO DISPOSITIVO, OU COMPUTAÇÃO – corresponde ao processamento completo de uma cadeia de entrada por um dispositivo: dada uma cadeia de entrada completa, e encontrando-se o dispositivo em sua configuração inicial, estímulos sucessivos vão sendo extraídos dessa cadeia de entrada, provocando um novo movimento no dispositivo.
Assim, a cada passo do dispositivo, estando ele em uma dada configuração, e recebendo um estímulo de entrada, é consultado o conjunto de regras que define seu comportamento, para determinar qual deverá ser sua próxima configuração.
Pela aplicação adequada das regras, uma sucessão de movimentos ocorrerá no dispositivo, provocando uma correspondente seqüência de mudanças de configuração, até que seja atingida alguma configuração final, encerrando a computação da cadeia de entrada inicialmente proposta.
SISTEMAS ADAPTATIVOS GUIADOS POR REGRAS – conseguindo-se descrever todo o comportamento de um fenômeno na forma de uma coleção de regras condicionais, tem-se o necessário para especificar os chamados "sistemas guiados por regras", que constituem o substrato dos sistemas adaptativos guiados por regras.
Para tanto, é preciso transformar as regras passivas de mudança de configuração, em regras ativas que, além de determinar as mudanças de configuração para o dispositivo, se torne também capaz de alterar o próprio comportamento do mesmo.
Torna-se assim possível, se necessário, estender regras não-adaptativas da forma "ao receber um estímulo X na configuração Y, a nova configuração passa a ser Z", para torná-las capazes de definir comportamentos auto-modificáveis, ou seja, adaptativos.
Isso pode ser feito facilmente incorporando-lhes uma componente adaptativa, resultando cláusulas da forma: "ao receber um estímulo X na configuração Y, a nova configuração passa a ser Z, e também o conjunto de regras se altera pela remoção/ substituição/ inserção as regras R_1, ... , R_n".
Todas essas cláusulas podem ser codificadas, por exemplo, na forma de estruturas de dados, e depositadas em uma área do programa passível de sofrer alterações por parte do próprio programa.
Nessas condições, nada poderá impedir que esse programa modifique esses dados, podendo assim, por exemplo, remover dessa área aqueles que codificam cláusulas consideradas desnecessárias, apagá-las, substitui-las por outras, alterar algum dos elementos que as compõem, etc.
A disponibilidade desse recurso proporciona a qualquer sistema ou programa todos os requisitos básicos para a materialização da sua própria auto-modificação, ou seja, para a incorporação, com uma relativa facilidade, do conceito da adaptatividade.
ADAPTATIVIDADE – é a propriedade que apresenta um sistema, dispositivo ou processo computacional, que lhe permite, sem a interferência de agentes externos – mesmo do próprio operador – tomar a decisão de modificar dinamicamente, de forma autônoma, seu próprio comportamento, em resposta apenas à sua configuração corrente e ao estímulo de entrada recebido.
DISPOSITIVO ADAPTATIVO – é todo dispositivo cujo comportamento incorpore o conceito da adaptatividade. Um dispositivo adaptativo guiado por regras é regido por um conjunto de cláusulas da forma "ao receber um estímulo X na configuração Y, a nova configuração passa a ser Z, e também o conjunto de regras se altera pela remoção/ substituição/ inserção as regras R_1, ... , R_n".
As regras adaptativas especificam, para cada possível configuração do dispositivo, não apenas uma nova configuração, mas também as eventuais alterações de comportamento, a ela associadas, as quais caracterizam a adaptatividade do dispositivo, e são representadas pelos mecanismos de automodificação expressos nas regras adaptativas, que determinam o comportamento dinâmico do dispositivo.
PROCESSO ADAPTATIVO – é aquele que apresenta um comportamento ativo, responsável por modificar espontaneamente seu próprio funcionamento, por força de decisão tomada exclusivamente pelo próprio processo, com base apenas na sua configuração e no seu estímulo de entrada correntes, sofrendo assim alterações comportamentais, por vezes intensas, à medida que for avançando em sua operação.
Um processo adaptativo se caracteriza por ser auto-modificável, sendo as alterações que sofre exclusivamente ditadas pelas próprias regras que definem o comportamento desse processo, podendo portanto ser realizadas, por exemplo, através de uma auto-reestruturação dos elementos computacionais que o implementam.
Em processos adaptativos, podem ocorrer, portanto, mudanças de comportamento em tempo de execução, e em conseqüência, enquanto operam, softwares adaptativos costumam permitir alterações tanto em seus dados quanto a em seu próprio código.
É característico de um software adaptativo tomar livremente a decisão de alterar seu próprio comportamento, em função ou em decorrência apenas da sua própria configuração na ocasião.
PROCESSO ADAPTÁVEL – diferindo dos processos adaptativos, cuja dinâmica comportamental é ativa, os processos adaptáveis apresentam-se com um comportamento modificável reativo, como acontece em softwares configuráveis ou selecionáveis.
Em outras palavras, o comportamento desse tipo de processos pode ser escolhido, em época de execução, pela ação de intervenções

externas, tais como comandos do operador, eventos específicos, condições especiais detectadas, etc.

Em sistemas com essa característica, escolhe-se o comportamento que se deseja para o processo, e em reação, ocorre uma alteração de comportamento, de acordo com a escolha realizada, sem a participação ativa do sistema, como seria o caso em processos adaptativos.

Em geral, processos adaptáveis oferecem a seus usuários um repertório de sub-sistemas operantes, os quais, tipicamente, são selecionados por algum agente externo ao processo, através, por exemplo, da ação de comandos de múltipla escolha.

Não ocorre, portanto, qualquer modificação de código ou de dados, nem mesmo alterações comportamentais reais em época de execução, e sim apenas a seleção de alguma funcionalidade pré-existente, que implemente o comportamento desejado em cada ocasião.

PROCESSO PARAMÉTRICO – costuma, em geral, apresentar um comportamento pré-determinável, ajustável a situações específicas, tirando para isso proveito da possibilidade de uma simples escolha apropriada de seus argumentos.

A implementação de programas que apresentam esta característica é feita, em geral, empregando-se procedimentos (funções, sub-rotinas, métodos, etc.) com parâmetros.

Embora simples, a chamada de tais procedimentos, cuja personalização pode ser feita pela escolha de argumentos adequados, mostra-se suficientemente poderosa para propiciar a seu usuário um mecanismo eficaz para a escolha de um particular comportamento desejado.

Verifica-se com facilidade que, neste tipo de processos, não ocorrem alterações comportamentais dinâmicas, apenas um simples ajuste do programa em execução, obtido exclusivamente pela alteração dos seus dados, sem quaisquer modificações de seu código.

PROCESSO COM COMPORTAMENTO SELETIVO – caracteriza-se por permitir que, em operação, seu comportamento seja estaticamente comutado entre um certo número fixo e pré-existente de alternativas, oferecidas na forma de um conjunto de opções.

Do ponto de vista de implementação, tudo se passa como se houvesse, nos processos com comportamento seletivo, um grande elemento de seleção múltipla, que disponibiliza, para escolha, diversos módulos relativamente independentes, os quais realizam os diferentes comportamentos para o processo, e que podem ser selecionados de acordo com a necessidade do seu usuário final.

Não há, tampouco neste caso, qualquer modificação comportamental dinâmica, mas apenas a seleção de um dos comportamentos disponibilizados, a partir de um repertório de possibilidades.

Pode-se afirmar que se trata de uma forma muito rudimentar de adequação do comportamento do processo às necessidades do usuário, sem envolver quaisquer alterações, quer no código, quer nos dados, já que aqui também não ocorre qualquer alteração dinâmica de comportamento, apenas uma simples seleção.

PROCESSO CONFIGURÁVEL – essa designação refere-se àqueles processos que tenham sido projetados para disponibilizar, de uma vasta gama de funcionalidades possíveis, subconjuntos escolhidos caso a caso, estritamente de acordo com as necessidades de cada um de seus inúmeros usuários.

Processos configuráveis costumam ser muito usado em grandes sistemas, com um elevado número de usuários, e que disponibilizem uma grande variedade de funcionalidades.

Nessa situação, um particular cliente dificilmente se interessa pelo uso de todas as funcionalidades potencialmente disponíveis, sendo então adequado que cada qual adquira apenas os direitos de acesso daquelas que lhe sejam especificamente convenientes.

Para que o processo possa ser assim personalizado, costuma-se arquitetá-lo de forma tal que apresente um configurador, encarregado da tarefa de adequá-lo às necessidades de cada cliente, e para isso, operações de escolha ou de mascaramento de comportamentos podem ser utilizadas por esse módulo para facilitar a composição do software de acordo com a conveniência de cada cliente.

Dessa maneira, em uma fase preliminar da utilização de um software configurável, permite-se ao usuário adquirir todas as funções que lhe sejam úteis, e, de posse de tal informação, o módulo configurador pode então gerar um programa personalizado, com o exato comportamento (estático) desejado, o qual passa finalmente à sua fase de operação, agora já com sua funcionalidade restrita ao conjunto dos comportamentos selecionados.

Embora com essa técnica sejam promovidas, em um software, variações comportamentais de fato, estas nunca se manifestam durante a operação final do processo configurável.

Apesar disso, para um dado software configurável, tais variações podem ser notadas com muita facilidade observando-se os comportamentos diferenciados exibidos por diversas de suas instâncias, que tenham sido independentemente configuradas.

É necessário, todavia, insistir que, apesar de existir nesses casos uma real variação de comportamento de programa para programa, essa variação não é dinâmica, ocorrendo apenas na fase de configuração, portanto totalmente fora da época de execução desses programas.

Assim, em geral, o código que implementa essa classe de processos é personalizado, caso a caso, a partir de módulos, porém, uma vez montado e entrando em operação, não mais se modifica.

Dessa forma, mesmo que dois usuários adquiram o mesmo software, se tiverem optado pela utilização de funcionalidades diferentes, cada um deles terá à sua disposição, para todos os efeitos, repertórios diferentes de comportamentos em seus programas.

PROCESSO MULTIFUNCIONAL – forma outra categoria de programas de comportamento variável, projetados para permitirem, a cada execução, a escolha estática de um comportamento particular, selecionado dentre um conjunto fixo de comportamentos disponibilizados, pré-existentes e prontos para serem diretamente executados.

Este tipo de programa, construído definitivamente uma única vez, incorpora e disponibiliza ao seu usuário todas as funcionalidades possíveis, sendo em geral implementado na forma de uma única seleção múltipla, eventualmente hierárquica, abrangendo todas as partes do programa responsáveis pela realização dos variados comportamentos alternativos previstos para o programa.

Pela natureza desse tipo de programas, não é possível a ocorrência de alterações comportamentais dinâmicas, já que o controle de sua operação se restringe à seleção da funcionalidade a ser executada a cada utilização.

Processos multifuncionais são, portanto, completamente estáticos, e não efetuam modificações em seu código nem nos seus dados de controle, pois sequer sofrem alterações comportamentais.

Hoje em dia, é muito popular o uso de equipamentos multi-funcionais tais como, por exemplo, uma impressora que também é scanner, fax, telefone, relógio, etc., ou então um telefone celular que também é terminal de internet, máquina fotográfica, jogo eletrônico.

Trata-se de equipamentos preparados para realizarem muitas tarefas diferentes, porém em cada instante tais equipamentos executam, na maior parte dos casos, uma única dessas funcionalidades, em atendimento a uma seleção de função.

Para o usuário, obtém-se o efeito da variação comportamental através da solicitação de execução, dentre as funções disponíveis, daquela que confira ao programa o comportamento que se mostre mais adequado na ocasião.

PROCESSO PROGRAMÁVEL – oferece aos seus usuários a possibilidade de especificar, de programar e/ou de reprogramar sua funcionalidade.

Para obter esse efeito há duas formas clássicas, ambas envolvendo modificações de uma representação lógica, ou especificação, do comportamento que se deseja para o processo.

A primeira dessas alternativas emprega uma máquina virtual, encarregada de interpretar, na forma de um script, uma especificação que descreva o comportamento desejado para o processo.

Nesse caso, tal script toma a forma de uma estrutura de dados, localizado em uma área de memória acessível para modificações, o que permite ao processo programável alterá-lo com facilidade durante a sua execução, ajustando-o assim, de forma dinâmica ao comportamento mais adequado em cada ocasião.

Inicialmente especificado pelo usuário através de uma linguagem apropriada, esse script é convertido por um pré-processador para um formato adequado, e depositado, na forma de estruturas de dados, em uma área de trabalho.

Isso permite que o programa extraia dessas estruturas, na forma de estruturas de dados interpretáveis, informações simbólicas que descrevem de forma rigorosa o comportamento que se deseja para o processo, podendo então um programa interpretador, no papel de máquina virtual, interpretá-los sempre que necessário.

Adicionalmente, estando assim todo o comportamento do processo representado na forma de estruturas de dados, estas poderão ser alteradas sempre que se desejar, promovendo o efeito de uma funcionalidade dinâmica, totalmente controlável pelo usuário.

Vale notar a real variação de funcionalidade que pode ser obtida dessa forma, e que essa variação não resulta de alterações na área de código do processo, mas de modificações no conteúdo da parte de sua área de dados que codifica o comportamento do programa.

Como uma segunda alternativa para a realização de processos programáveis, é possível utilizar diretamente a própria máquina real, processando diretamente seu código nativo.

Nesse caso, o comportamento do processo fica representado na lógica do próprio código que se executa na máquina, e assim, somente através de modificações diretas do próprio código do programa é que se pode, neste caso, obter a funcionalidade dinâmica desejada.

Essa manobra nem sempre se mostra trivial na maior parte dos sistemas computacionais modernos, os quais costumam impor restrições de acesso à área de código, com a finalidade de impedir, ou ao menos, dificultar a prática da automodificação de programas.

Nas duas formas citadas de implementação, os processos programáveis alcançam funcionalidade genuinamente dinâmica, que se manifesta pela variação comportamental do programa em execução, em conformidade com o comportamento especificado, seja na forma de script ou de um código nativo.

PROCESSO AUTO-PROGRAMÁVEL – é todo aquele que permite que sua própria funcionalidade seja especificada, programada ou re-programada, a qualquer momento.

Isso lhes permite, dessa maneira, que registrem automaticamente, não apenas para uso próprio, mas também para que seja posta em execução, uma informação que codifique, em cada oportunidade, o comportamento que se deseja para esses processos.

Uma prática que permite obter esse efeito é garantir que estejam adequadamente codificadas, na forma de estruturas de dados, as seqüências de operações que descrevem o comportamento desejado.

Para que programas desse tipo possam ser de fato utilizados em aplicações complexas, pode-se dotá-lo adicionalmente de um recurso extremamente poderoso: o programa, por algum meio a seu alcance, gera, ele próprio, segundo sua própria lógica, e a partir de decisões tomadas pelo próprio programa, estruturas de dados que descrevam seu próprio funcionamento.

Dessa maneira, tais especificações tornam-se disponíveis para uso pelo próprio processo, a qualquer momento, e isso permite que uma máquina virtual, integrante do próprio processo auto-programável, as interprete e assim realize o comportamento que elas especificam.

Neste caso, o efeito obtido é o de uma funcionalidade genuinamente dinâmica e autônoma, embora para isso não seja necessário promover quaisquer alterações no código nativo do processo auto-programável, e sim apenas nas áreas de dados que codificam o seu comportamento.

Para que se possa tirar maior proveito ainda desse tipo de arquitetura, mostra-se oportuno que processos auto-programáveis, assim construídos, operem em colaboração com procedimentos de Inteligência Artificial.

Dado o poderoso potencial de suporte que representam, os procedimentos de Inteligência Artificial podem potencializar significativamente o alcance dos processos auto-programáveis aos quais estiverem incorporados.

Em geral, costumam ser acoplados aos processos para serem acionados em ocasiões estratégicas, como ocorre em situações de tomada de decisões e, em especial, naquelas ocasiões em que se necessite estabelecer quais alterações comportamentais dinâmicas devem ser promovidas no processo auto-programável.

Isso contribui significativamente para a obtenção de programas de alta complexidade, e de alcance muito vasto, não apenas devido ao comportamento dinâmico assim obtido, como também pelo aumento da expressividade que isso empresta aos processos auto-programáveis, especialmente àqueles que realizam atividades de elevada complexidade.

PROCESSO ADAPTATIVO – é um processo auto-programável especial, que permite a obtenção do efeito de funcionalidade dinâmica autonomamente auto-modificável com a ajuda dos recursos da auto-programação dinâmica obtida pela alteração do próprio código nativo, que é executado diretamente por uma máquina física, sem o auxílio de interpretadores.

Neste caso, a funcionalidade dinâmica resulta de uma alteração real do próprio código executável de máquina, que representa o processo, portanto, essa dinâmica de comportamento decorrente de ações emanadas da execução do próprio programa.

Disso resulta, em tempo de execução, uma real modificação do programa em operação, e, em conseqüência, uma correspondente alteração do comportamento desse programa, enquanto ele próprio vai sendo executado e, possivelmente, até mesmo provocando, em si próprio, alterações comportamentais adicionais.

Não se trata, portanto, de uma ação externa que modifica a conduta do programa, mas de uma real e deliberada alteração de comportamento provocada *on line* pelo próprio programa, durante sua própria execução.

Portanto, as atividades de criação de programas, que codificam processos com código auto-modificável, exigem que se leve em consideração que programas adaptativos devem ser capazes de promover alterações dinâmicas em seu próprio código.

Isso pode ser obtido de várias formas, tais como:

a) pela ativação explícita de funções de uma biblioteca específica, que adequadamente acionadas, promovam a alteração do código.
b) usando máquinas virtuais para interpretar programas adaptativos, codificados em alguma linguagem de baixo nível auto-modificável, programas esses capazes de alterar uma área de dados na qual esteja descrito o comportamento do programa. Portanto esta seria, na realidade, uma implementação híbrida, baseada no conceito de processo auto-programável, descrito anteriormente.
c) mediante o emprego de recursos de auto-referência e de auto-modificação, que sejam disponibilizados por alguma linguagem de alto nível voltada à codificação de programas adaptativos. Esta pode ser uma linguagem de programação nativamente adaptativa (especialmente projetada para essa finalidade, com recursos sintáticos adequados para expressar de forma adequada os fatos adaptativos do programa), ou então alguma extensão adaptativa de alguma linguagem convencional de programação (implementada com o auxílio de expansores de macros, de pré-processadores destinados à resolução de extensões sintáticas adequadas, ou ainda de compiladores específicos, projetados particularmente para o processamento de tais linguagens estendidas). Diversos esforços têm sido canalizados para a investigação deste assunto, tendo como meta a obtenção de linguagens de programação voltadas para a codificação confortável de programas de natureza automodificável.

VI. Observações Finais

O glossário aqui apresentado é constituído de um conjunto representativo dos principais termos envolvendo conceitos que têm sido alvos de pesquisas e desenvolvimentos recentes na área.

Seu conteúdo ainda está longe de ser exaustivo, pois o levantamento aqui relatado se limita a considerar os termos mais controvertidos ou empregados de maneira menos uniforme nas publicações disponíveis sobre o assunto.

Assim sendo, dada a importância, para todos os que se interessam por este ramo do conhecimento, de se dispor de um material completo desta natureza, solicita-se a todos os interessados que ofereçam suas contribuições para que, a partir delas e de suas complementações, o presente glossário venha a ampliar-se, em futuro próximo, pela incorporação também de outros termos técnicos da área, já consagrados.

Alcança-se assim uma abrangência cada vez maior, para que venha a tornar-se em breve um material de referência de grande utilidade, não apenas para todos os que já estudam, pesquisam e utilizam a adaptatividade em suas atividades, como também uma forma de disseminação do conceito entre todos aqueles que ainda não tiveram a oportunidade de trabalhar diretamente com esse importante conceito.

VII. Referências

[1] M.V.M. Ramos; J.J. Neto, I.S. Vega, Linguagens Formais, ISBN 978-85-7780-453-5, Porto Alegre, Ed. Bookman, 2009.

[2] J.J. Neto, Contribuições à Metodologia de Construção de *Compiladores*, Tese de Livre Docência – Escola Politécnica da Universidade de São Paulo, São Paulo, 1993.

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

[4] http://www.pcs.usp.br/~lta/ Página web do LTA – Laboratório de Linguagens e Técnicas Adaptativas.

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.

Adaptatividade: Generalização Conceitual

J. J. Neto

Resumo — **Este material refere-se ao conteúdo do mini-curso tutorial de mesmo nome, apresentado em 29 de janeiro de 2009 em São Paulo, por ocasião do WTA 2009 – Terceiro Workshop de Tecnologia Adaptativa. Faz-se aqui o estudo de alguns dos mais importantes aspectos da realização computacional das abstrações expressas através de formulações adaptativas, com a ajuda de ferramentas recentemente disponibilizadas. Pretende-se com isso proporcionar, aos interessados em utilizar a adaptatividade na prática, uma visão atualizada da situação em que se encontram os recursos teórico e tecnológico hoje disponíveis, e da maneira como se relacionam.**

Palavras-chave — **Adaptividade, Dispositivos Adaptativos, Comportamento Auto-modificável, Sistemas Adaptativos, Tecnologia Adaptativa.**

VIII. *Introdução*

O tema central em torno do qual costumam girar as discussões acerca dos assuntos ligados à adaptatividade é a forma como pode ser realizada a variabilidade funcional dos programas.

Em maior ou menor intensidade, quase todos os programas modernos bem projetados, e de porte razoável, têm sido construídos de tal forma que possam executar, de forma versátil, uma certa variedade de operações distintas.

De um para outro programa, podem-se constatar diferentes graus de visibilidade e de acesso do usuário à diversidade de funções que tais programas oferecem. Mais ainda, de programa para programa, variam muito as técnicas empregadas em sua construção.

Tais técnicas procuram dotar esses programas da possibilidade de proporcionarem a seus usuários o efeito da funcionalidade variável.

Programas pequenos e de uso restrito costumam ser projetados especificamente para apresentarem uma **funcionalidade constante**, sem opções de variação. Seu código, naturalmente estático, costuma ser muito simples. Seu comportamento, em conseqüência, é fixo e inflexível. A grande vantagem de programas dessa categoria é a simplicidade, que proporciona um baixo custo, mas uma eventual exigência posterior de recursos adicionais acarreta a necessidade de reprogramação, que permita incorporar as novas funcionalidades.

Programas ainda de pequeno porte, porém de propósito pouco menos restrito costumam ser projetados de tal forma que proporcionem mais flexibilidades para seus usuários. Apesar de executarem também uma única funcionalidade fixa, esta é materializada na forma de um procedimento paramétrico. Argumentos adequados instanciam os seus parâmetros, personalizando de forma prática seu comportamento a cada ativação dos procedimentos que os implementam.

Todos os que já se serviram de algum programa, procedimento, sub-rotina ou método paramétrico sabem que a parametrização agrega ao programa um potencial enorme, em relação a outro, que não se utilize de parâmetros.

Para que um programa não paramétrico possa executar operações diferentes daquela que ele realiza, torna-se necessário reescrevê-lo, ainda que na forma de uma cópia modificada. Por sua vez, basta fornecer diferentes conjuntos de dados a um programa paramétrico para que ele execute atividades diferentes, em correspondência aos particulares dados recebidos, sem que sejam promovidas quaisquer alterações em seu comportamento original.

Assim, neste caso particular, os parâmetros assumem o papel de agentes de personalização do comportamento do programa, a despeito de não ser variável a funcionalidade dos procedimentos com código estático.

Muitos programas oferecem uma variedade de funcionalidades fixas, freqüentemente parametrizadas, e permitem selecionar e executar uma delas de cada vez.

Uma situação ligeiramente mais complexa é a dos programas que executam diversas funcionalidades. Neste caso, seções diferentes são criadas na estrutura do programa, e se destinam a executar operações diferentes. Através de um seletor, pode-se escolher dentre essas diversas funcionalidades, aquela que se deseja executar.

Cada uma dessas seções pode, por sua vez, ser paramétrica, e com isso torna-se possível combinar as duas formas mencionadas de variação de funcionalidade, e dessa maneira, através de parâmetros e de dados de entrada que sejam especificamente usados para selecionar, torna-se possível uma escolha confortável da exata funcionalidade que se deseja do programa, a cada vez que este é executado.

Dessa forma, obtém-se um programa estático, com um conjunto constante de funcionalidades, mas cujo comportamento pode ser selecionado através de parâmetros, proporcionando assim uma grande gama de combinações.

Aumentando um pouco mais a complexidade do programa, pode-se acrescentar à parametrização a possibilidade de mascaramento de funcionalidades predefinidas.

Escolhendo-se um valor conveniente para a máscara, bloqueia-se ou habilita-se o acesso do usuário a cada uma das funções de um conjunto fixo que o programa potencialmente oferece, controlando assim o acesso do programa às diversas funcionalidades disponíveis.

Isso permite que um mesmo hardware e um mesmo programa básico possam ser utilizados de formas diferentes pelos diversos usuários, conforme a visibilidade que lhes é dada das funcionalidades potenciais do equipamento.

Assim, apesar de todo cliente possuir o software completo, ele poderá utilizar apenas os módulos realmente adquiridos, e aos demais, embora existam, não lhe será dada permissão de uso.

Tem-se aí, portanto, a simulação de uma funcionalidade variável, através do uso conjunto dos recursos da parametrização e da seleção, apresentados nos casos anteriores.

Quando se compra, por exemplo, um eletrodoméstico ou um equipamento complexo, quase sempre se recebe um hardware contendo um programa completo, capaz de executar todas as funções que foram projetadas para ele, mas ao usuário é dado acesso apenas às pertencentes ao subconjunto escolhido na ocasião da compra.

Assim, através da máscara, o fornecedor impede, segundo critérios preestabelecidos, o acesso a determinadas funções e o libera a outras, conforme a configuração a que cada particular cliente tem direito.

Sofisticando um pouco mais a estrutura dos programas, através do emprego do conceito de máquina virtual, com código interpretado, torna-se possível efetuar a simulação de (re)programações dinâmicas de funcionalidade, em programas mais complexos.

Imagine-se um programa como o mencionado anteriormente, mas desta vez equipado com um interpretador (máquina virtual) que é capaz de simular um certo conjunto de operações.

Seu usuário poderá escolher, dentre tais operações, aquelas que o ajudem a configurar o sistema, de forma que fique adequado aos seus propósitos. Esse conjunto de operações pode ser utilizado ativando-se

cada uma delas por meio de um script, organizado na forma de um programa.

Através desse script, o próprio programa se torna assim capaz de especificar quais operações devem ser executadas, bem como a ordem em que elas devem ser efetuadas. É possível, dessa maneira, memorizar esse script em uma área de dados, e na época da execução, o interpretador, operando sobre eles, faz com que tais dados se comportem como se fossem as instruções de um programa.

Resulta desse procedimento a execução interpretada do programa correspondente a esse script, a qual é viabilizada explorando-se as características associadas ao conceito de máquina virtual. Com isso, confere-se um comportamento programável ao software assim constituído, e isto era exatamente o importante passo que faltava ser dado para se dotar esse software da valiosa característica da adaptatividade.

Tornava-se conveniente incorporar um mecanismo que permitisse ao programa alterar o seu comportamento, e que tal modificação pudesse ser feita justamente através de transformações convenientes no script que determina a sua programação. Dispondo-se de tal recurso, o operador ganha a possibilidade de, agindo externamente, modificar a programação do sistema, determinando dessa maneira o seu comportamento imediatamente antes da execução do script.

Até aqui não se trata ainda de adaptatividade verdadeira, mas de uma técnica muito similar, e que proporciona muita versatilidade a um software, cujo comportamento prévio já podia ser parametrizado, simulado e selecionado, e que passa agora a ser também programado.

A seguir, nessa sucessão de complexidades crescentes dos programas, rumo à adaptatividade, o passo natural consiste em implantar nos programas recursos adicionais que proporcionem condições para que a funcionalidade desses programas passe a ser dinamicamente variável.

Isso significa que, durante a sua execução, os programas devem dispor de meios para criarem automaticamente novas funcionalidades através do emprego, direto ou indireto, da técnica da auto-modificação de código.

O caso anteriormente apresentado refere-se a um programa com funcionalidade estaticamente variável: embora seja possível modificar o script, para que este seja em seguida executado, tal script, uma vez alterado pelo usuário, permanecerá invariável durante toda a execução restante do programa.

Se, adicionalmente, for concedida ao próprio programa a possibilidade de acessar e modificar essa área de script, torna-se viável a obtenção de programas capazes de simular um comportamento dinâmico, apesar de, na realidade, seu código não ser obrigatoriamente dinâmico ou auto-modificável.

O esquema aqui apresentado opera da seguinte maneira: tem-se uma máquina virtual que interpreta instruções, e um script, codificado na forma de uma seqüência de dados, os quais representam operações de um programa a ser executado.

Os códigos assim formados vão sendo sucessivamente lidos e interpretados, e as operações correspondentes vão sendo executadas pela máquina virtual, e portanto a atividade que o script codifica acaba sendo efetuada passo a passo, dessa maneira.

É possível ainda disponibilizar no programa, para ser utilizada pelo seu próprio usuário final, uma funcionalidade que propicia uma atividade de reprogramação, efetuada por intervenção do próprio usuário final, o qual se torna assim capaz de alterar esse programa.

O passo decisivo para a adaptatividade pode ser dado em seguida, ao se proporcionar ao próprio programa a possibilidade de acionar essa funcionalidade de reprogramação.

Procedendo desta maneira, a alteração dinâmica de código daí decorrente deixa de ser resultado de ação humana, e portanto poderá ser conseqüência de uma atividade pré-programada ou então pode decorrer de decisões automáticas tomadas por procedimentos auxiliares de inteligência artificial incorporados ao programa.

Logo, através da ação do próprio programa, possivelmente com o auxilio de um procedimento de auto-modificação, auxiliado por um outro de tomada de decisão, ou de mais alguma lógica especial, acaba-se fechando finalmente esse ciclo, permitindo que o programa seja modificado por iniciativa e por ação do próprio programa, sem auxilio externo.

Nesse cenário, tendo-se um script, que o próprio programa possa modificar, script esse que determine integralmente o comportamento do programa, consegue-se finalmente obter, pela primeira vez, softwares que exibem um verdadeiro comportamento dinamicamente variável.

Convém notar, porém, que apesar de o código propriamente dito que o implementa continuar não sendo automodificável, o resultado da prática acima descrita é um comportamento autenticamente automodificável, embora ainda simulado.

É possível, assim, ter um código que seja alternativamente nativo ou virtual, e com isso se obtém um programa dinâmica ou estaticamente variável, ou seja, um software que, em tempo de execução, permita alterar ou não seu próprio código, respectivamente

O estudo apresentado até aqui sobre softwares com funcionalidade dinâmica levou-nos a percorrer uma seqüência gradativa de arquiteturas de programas, que cobre desde programas muito simples, de comportamento totalmente estático, até programas sofisticados, com comportamento totalmente dinâmico.

Toda a pesquisa que vem sendo realizada, tendo como alvo os programas adaptativos, orienta-se pela meta de desenvolver programas cujo comportamento possa ser calibrado em função da necessidade, ganhando-se com isso a possibilidade de ter uma funcionalidade que seja ao mesmo tempo dinâmica e também programável.

EXEMPLOS ILUSTRATIVOS – Por meio de programas que apresentam comportamento variável, podem ser elaboradas inúmeras aplicações de grande utilidade prática.

A modelagem rigorosa de tais aplicações pode empregar formulações matemáticas clássicas, eventualmente transformadas em versões adaptativas, mediante a adição de elementos que tornem o comportamento por elas descrito alterável durante sua operação.

Um exemplo, do dia-a-dia, mostra a situação em que pessoas e veículos modificam seu comportamento em função de alterações no ambiente em que estão mergulhados.

Só para citar uma dessas situações, a simples mudança periódica do estado de um semáforo de trânsito afeta dinamicamente o comportamento dos veículos e dos pedestres que trafegam na via: aqueles bloqueados, para os quais a mudança foi de vermelho para verde passam a poder trafegar livremente, o inverso ocorrendo para os demais, para os quais a mudança foi para vermelho, e cujo tráfego, antes livre, passa a estar bloqueado.

Programas reativos constituem outro exemplo, quando as ações por eles tomadas em resposta a um dado estímulo puderem variar em função dos seus antecedentes e da situação em que eles se encontram no instante em que o estímulo for recebido.

De forma ampla, programas em geral também costumam exibir uma característica similar: eles recebem dados como estímulos de entrada a serem tratados, e em função de sua situação interna, produto de sua história, podendo reagir ao estímulo de acordo com essa situação.

Outro exemplo, significativo e intuitivo, é o de uma simulação típica, como a de um jogo eletrônico, a de um brinquedo eletronicamente controlado de um parque de diversões, a de uma cabine de comando, ou de tantas outras atividades complexas.

Dado algum jogo, cujo comportamento original esteja definido na forma de um conjunto de regras, se à aplicação de uma ou mais dessas regras puder ser associada alguma alteração do conjunto das regras que o definem, tem-se aí criada uma realização concreta de um comportamento dinamicamente variável para o software resultante.

No novo simulador assim obtido, o conceito da adaptatividade pode ser identificado se for constatada a singular característica, assim

adquirida, de poder alterar espontaneamente seu próprio comportamento.

IX. Dispositivos adaptativos – formulação estendida

Discorre-se a seguir acerca de uma extensão à formulação clássica dos dispositivos adaptativos gerais [1], [2], [3], que com ela passam a levar em conta diversos avanços conceituais e estruturais recém-desenvolvidos nessa área.

Inicia-se descrevendo a componente não-adaptativa, presente em todos esses dispositivos na forma de um dispositivo subjacente guiado por regras, o qual serve como suporte para a construção da versão adaptativa.

Em seguida, apresenta-se uma proposta, cuja finalidade é a de dotar de adaptatividade esse dispositivo subjacente, acoplando-se-lhe uma camada adaptativa, a qual difere, em relação à formulação tradicional, essencialmente por apresentar-se estruturada de forma hierárquica, e em vários níveis.

A. Dispositivos Não-adaptativos Guiados por Regras

Os dispositivos abstratos guiados por regras têm seu comportamento totalmente descrito pelo conjunto, finito e fixo, de regras condicionais de movimentação, da forma clássica "**se** ... **então**".

Essas regras representam leis de mudança de configuração para o dispositivo, as quais determinam seu funcionamento. Na formulação aqui proposta, foi preservado seu formato tradicional.

Em operação, o dispositivo percorre sucessivas configurações, enquanto permanecer recebendo estímulos de entrada, exatamente conforme foi anteriormente descrito.

B. Consideração sobre Determinismo

Determinados dispositivos guiados por regras podem exibir uma atraente propriedade, a de apresentarem uma única possibilidade de movimento a partir de qualquer configuração em que se encontrem, ao longo de toda a sua operação.

A essa propriedade dá-se o nome de determinismo, e qualquer dispositivo que a apresente é dito determinístico.

Conseqüentemente, dispositivos determinísticos apresentam um eficiente comportamento dinâmico, pois fluem livremente, de configuração em configuração, até que esgotem os estímulos recebidos, quando encerram então sua operação.

Outros tipos de dispositivos, por seu lado, permitem mais de um movimento válido a partir de alguma de suas configurações, e nesse caso, denominam-se dispositivos não-determinísticos.

A realização prática mais natural de um dispositivo não-determinístico é obviamente aquela em que os possíveis movimentos sejam processados em paralelo, independentemente uns dos outros.

Em computadores nos quais não se disponha de tantos processadores quantos seriam exigidos por esse tipo de implementação, existe a possibilidade de seqüencializar as alternativas, de forma tal que seja processada apenas uma por vez, em cada processador disponível.

No limite, recai-se no tradicional processamento seqüencial, com um só processador, e nessa realidade a busca da seqüência mais adequada em cada caso recai em uma busca exaustiva, por tentativa e erro.

Ao contrário dos dispositivos determinísticos, os não-determinísticos exigem, a cada passo de operação, a execução de uma tomada de decisão, para que seja selecionado, dentre todas as possibilidades válidas de movimentação, um particular movimento, aquele que se mostrar mais adequado, a partir da particular configuração corrente.

Tais escolhas nem sempre são triviais, e sua aplicação sistemática durante a operação dos dispositivos não-determinísticos costuma prejudicar, em geral sensivelmente, a eficiência da operação do mesmo, constituindo isso uma das razões mais sérias pelas quais na prática se recomende que, na medida do possível, seja evitado o uso de não-determinismos.

C. Formulação dos Dispositivos Guiados por Regras

Nesta seção é apresentada a formulação adotada para os dispositivos (não-adaptativos) guiados por regras.

Um dispositivo não-adaptativo se caracteriza por não permitir qualquer tipo de alteração dinâmica em suas componentes, estruturais ou funcionais.

DISPOSITIVO NÃO-ADAPTATIVO – FORMULAÇÃO

Um dispositivo D, não-adaptativo, guiado por regras, pode ser descrito como uma quíntupla, conforme a formulação seguinte:

$$D = (C, R, S, c_0, A)$$

onde:

- C é o conjunto de todas as possíveis configurações de D;
- S é o conjunto de todos os estímulos válidos de entrada;
- $c_0 \in C$ é a única configuração inicial de D;
- $A \subseteq C$ é o conjunto de todas as configurações de aceitação de D;
- R é uma relação de mudança de configuração para D:

$$R \subseteq C \times (S \cup \{ \varepsilon \}) \times C,$$

cujos elementos constituem as regras que definem o dispositivo D:

$$r = (c_j, s, c'_j) \in R, \text{ com } c_j, c'_j \in C; s \in S,$$

as quais podem ser denotadas na forma

$$(c_j, s) \rightarrow c'_j,$$

realçando ser s o estímulo que, aplicado ao dispositivo D na configuração c_j, leva-o à sua nova configuração c'_j.

$$c_j \Rightarrow_s c'_j$$

A linguagem definida por D é representada por L(D):

$$L(D) = \{w \in S^* \mid c_j \Rightarrow_w^* c, c \in A\},$$

onde x* denota o fecho de Kleene de um conjunto ou operador x.

Dessa maneira, as regras, denotadas como triplas (c_j, s, c'_j), podem ser facilmente transcritas como cláusulas condicionais, da forma:

se a configuração corrente for c_j e o estímulo recebido for s,
então o dispositivo D poderá alterar sua configuração para c'_j.

Ocorrerá um movimento no dispositivo toda vez que uma dessas regras for aplicada, levando o dispositivo a migrar de sua configuração corrente para alguma outra, de acordo com aquilo que estiver determinado na regra aplicada.

Como a operação do dispositivo D depende total e exclusivamente desse conjunto de regras, ou, conseqüentemente, do conjunto de cláusulas condicionais equivalente, então pode-se dizer que o dispositivo D tem a sua operação determinada (ou guiada) pelo conjunto de regras que o define, ou, simplesmente, que se trata de um dispositivo guiado por regras.

D. Operação do Dispositivo Não-Adaptativo

O algoritmo seguinte descreve o funcionamento de um dispositivo (eventualmente não-determinístico) não-adaptativo.

Com essa descrição, estabelece-se um ponto de partida para a formulação, em seções subseqüentes, da operação de dispositivos mais complexos, com capacidade de auto-modificação.

DISPOSITIVO NÃO-ADAPTATIVO – OPERAÇÃO

1. Atribuir valores iniciais aos elementos:
   - $t = 0$ (o passo de execução corrente inicial é o passo 0)
   - $c_t = c_0$ (configuração inicial, fixa).

- $w_t = w_0 = w = s_1\ s_2\ s_3\ \ldots$ (seqüência inicial, completa, de estímulos de entrada).

2. Se $w_t = \varepsilon$ (seqüência de estímulos esgotada), desviar para o passo 4, caso contrário, extrair de w_t o próximo estímulo de entrada $s = s_{t+1}$, e então, incrementar t.
3. Obter, a partir de R, o sub-conjunto CR de regras compatíveis com o estímulo s e a configuração corrente c_t.
   - Se CR = ∅, rejeitar w.
   - Se CR={(c_t, s, c')}, mover D para a nova configuração $c_{t+1} = c'$, e desviar para o passo 2.
   - Se CR = {$r_k = (c_t, s, c'_k) \mid c'_k \in C, k = 1, \ldots, n, n>1$}, aplicam-se, em paralelo, as regras r_k, levando D simultaneamente às configurações $c'_1, c'_2, \ldots, c'_n$. Para cada um desses n casos, prosseguir no passo 2.
4. Caso $w_t = \varepsilon$ e D esteja nessa ocasião em alguma configuração $c_{t+1} \in F$, então D aceita w, caso contrário, D rejeita w. Em qualquer caso, encerra-se aqui a execução deste algoritmo.

A linguagem aceita pelo dispositivo será, então, o conjunto de todas aquelas seqüências de estímulos que, a partir de uma configuração inicial fixa, possam levar o dispositivo a alguma configuração final de aceitação, numa sucessão de movimentos determinados pela seqüência de regras aplicadas.

E. Adaptatividade Básica

Um dispositivo AD com adaptatividade básica se reduz a um dispositivo não-adaptativo subjacente, acoplado a um mecanismo capaz de alterar o conjunto de regras que define seu comportamento.

O mecanismo que converte um dispositivo não-adaptativo D em um dispositivo AD com adaptatividade básica incorpora um conjunto AA de funções especiais, denominadas funções adaptativas, acionáveis quando da aplicação das regras que definem o dispositivo adaptativo AD.

As funções adaptativas do conjunto AA proporcionam ao dispositivo AD uma forma de alterar o seu próprio conjunto das regras, pois os valores associados aos elementos variáveis que definem dispositivos adaptativos podem ser alterados a cada passo da sua operação.

Assim, resultam dispositivos adaptativos cujos comportamentos são descritos como conjuntos de regras, tendo à disposição uma variedade de funções adaptativas (incluindo a função adaptativa nula), as quais, convenientemente acopladas às regras do dispositivo subjacente, são executadas sempre que for necessário alterar o conjunto de regras.

Dispositivos não-adaptativos podem ser convertidos trivialmente em seus dispositivos adaptativos equivalentes, bastando usá-los como dispositivos subjacentes, e associando a função adaptativa nula a todas as suas regras.

Dispositivos assim obtidos são adaptativos no nome e na formulação, e, embora de fato não se automodifiquem, guardam matematicamente total compatibilidade com os dispositivos estritamente adaptativos, permitindo assim unificá-los no seu tratamento formal.

A introdução do conceito de funções adaptativas torna potencialmente mutável o comportamento de um dispositivo adaptativo, tornando conveniente o uso de um conjunto variável de regras RA, cujo valor RA_t depende do passo t de operação do dispositivo AD, exprimindo assim a dinâmica de seu comportamento. RA_t reflete, pois, o comportamento instantâneo do dispositivo adaptativo, ao seu t-ésimo passo de operação, e representa o conjunto de regras que nessa ocasião definem o comportamento do dispositivo.

Para formular as regras adaptativas do dispositivo AD, cada regra em RA_t associa a uma regra não-adaptativa do dispositivo subjacente duas funções adaptativas, a serem executadas uma antes, e outra, depois da mudança de configuração.

O emprego de uma regra $r \in R$, do dispositivo D subjacente, pela camada adaptativa AM(t) equivale à utilização de regras adaptativas da forma (β, r, α), cuja estrutura explicita o relacionamento entre o dispositivo subjacente e a camada adaptativa básica, aqui discutida.

Uma vez escolhida uma dessas regras adaptativas para ser aplicada, executa-se inicialmente sua função adaptativa anterior β, aplica-se a regra r, e por fim, executa-se sua função adaptativa posterior α.

Existe uma conjuntura particular que merece atenção, na operação dos dispositivos adaptativos: é a que ocorre quando, ao ser utilizada uma regra adaptativa, executando-se a função adaptativa anterior β, esta aplica à própria regra adaptativa uma operação de remoção, que a elimina do conjunto R(t) de regras.

Nesse caso, perde o sentido completar a aplicação da regra, pois esta já deixou de existir, e assim, o mais adequado em tal situação será descontinuar a aplicação daquela regra, reiniciando-se para o dispositivo o ciclo de aplicação de regras, agora escolhidas do conjunto R(t+1), resultante da remoção da regra em questão.

É importante alertar que a incorporação da adaptatividade a um dispositivo, por viabilizar sua automodificação, deve induzir seu usuário a tomar cuidados que garantam ao dispositivo manter-se permanentemente no controle, a despeito das alterações provocadas em seu próprio comportamento.

Cuidados similares se aplicam ao próprio software que implementa o dispositivo adaptativo, pois é real o perigo de perda do controle sobre o comportamento do dispositivo sempre que este, em operação, tiver a liberdade de provocar automodificações irrestritas.

O mecanismo adaptativo deve, portanto, ser projetado de forma tal que, apenas nas ocasiões mais apropriadas, seja promovida a execução de funções adaptativas que não comprometam a integridade operacional do dispositivo.

Do conhecimento do autor, não foi publicado nenhum método, para o desenvolvimento de camadas adaptativas, que garanta a perene preservação dessa integridade na operação de um dispositivo adaptativo, e isso constitui, sem dúvida, um interessantíssimo tema de pesquisa.

F. O Mecanismo Adaptativo Básico

A base da operação dos dispositivos dotados de adaptatividade básica torna-se assim bastante simples. Esse mecanismo conceitual rege o funcionamento dos dispositivos com adaptatividade básica, e se aplica a qualquer dispositivo, bastando para isso instanciar, no particular formato do dispositivo desejado, as regras mencionadas na formulação genérica a seguir:

MECANISMO ADAPTATIVO BÁSICO

Posicionar inicialmente o dispositivo AD em sua configuração inicial $c_0 \in C$, e extrair, da seqüência de entrada $w \in S^*$, o primeiro estímulo s a ser tratado.

1. Extrair, a partir do conjunto corrente de regras RA_t do dispositivo adaptativo, o sub-conjunto CR das regras compatíveis, para o estímulo s na configuração corrente c_t.
   São consideradas regras compatíveis todas as regras da forma $ra = (\beta b, r, \alpha a)$ cuja regra subjacente não-adaptativa $r = (c_t, s, c')$ referencie ao mesmo tempo a configuração corrente c_t e o estímulo corrente s.
   Para aplicar uma regra compatível $ra = (\beta b, (c_t, s, c'), \alpha a) \in CR$:
   - executar primeiro a chamada (*call*) βb da função adaptativa anterior (b). Caso isso faça com que a própria regra ra seja removida do conjunto RA_t, a regra correntemente em aplicação deixará de existir, perdendo portanto o sentido sua execução, devendo-se então voltar ao passo 1 para a escolha de outra regra compatível. Caso contrário, prosseguir.
   - aplicar a correspondente regra não-adaptativa subjacente (c_t, s, c'), o que leva o dispositivo AD à configuração c'.
   - finalizar executando a chamada (call) αa da função adaptativa posterior (a).

2. Aplicar simultaneamente todas as regras ra_j desse sub-conjunto CR (se houver mais de uma):
   - Se CR = ∅, encerrar a tentativa de reconhecimento em curso. Encerrar o algoritmo, rejeitando w se não houver outras tentativas paralelas em andamento.
   - Se CR = { ra = (βb, (c_t, s, c'), αa) }, conjunto unitário, aplicar, deterministicamente, a (única) regra ra, movendo AD para a nova configuração c_{t+1} = c'.
   - Se CR = { ra_k = (βb_k, (c_t, s, c'_k), αa_k) | $c'_k \in C$, sendo βb_k, αa_k chamadas (*calls*) de funções adaptativas pertencentes ao conjunto AA; k = 1, … , n, n>1}, aplicam-se, não-deterministicamente, todas as regras $ra_k \in$ CR.
3. Não havendo novos estímulos, encerrar o algoritmo, aceitando w caso tenha sido atingida uma configuração de aceitação. Caso contrário, extrair da seqüência de entrada um novo estímulo s, incrementar t e voltar ao passo 1 para que seja iniciado mais um passo de operação do dispositivo AD.

É importante notar que um mecanismo dessa natureza pode ser considerado como sendo o alicerce principal da adaptatividade, pois é ele que provê a dinâmica comportamental dos dispositivos, podendo ser identificado com facilidade em todos os fenômenos nos quais se manifestem atividades de automodificação.

G. Formulação de Dispositivos com Adaptatividade Básica

Dispositivos com adaptatividade básica são dispositivos adaptativos cuja capacidade de modificação dinâmica fica restrita à alteração do conjunto de regras adaptativas, que determina seu comportamento. Nesta seção é apresentada a formulação adotada para dispositivos guiados por regras, portadores de adaptatividade básica, os quais se caracterizam por permitirem alterações dinâmicas apenas nos conjuntos de regras que definem seus respectivos comportamentos.

DISPOSITIVO ADAPTATIVO BÁSICO – FORMULAÇÃO

Um dispositivo AD, guiado por regras, dotado do recurso de adaptatividade básica, pode ser descrito como uma sêxtupla, conforme a formulação seguinte:

AD = (C, RA, S, AA, c_0, RA_0, A), onde:

- C é o conjunto de todas as possíveis configurações de AD;
- $c_0 \in C$ é a configuração inicial, única, de AD;
- $A \subseteq C$ é o conjunto das configurações de aceitação de AD;
- S é o conjunto dos estímulos válidos de entrada de AD;
- AA é o conjunto, fixo, de funções adaptativas de AD;
- RA_0 é um conjunto, fixo, de regras adaptativas iniciais de AD;
- RA é o conjunto de regras adaptativas de AD;

Merecem destaque os seguintes detalhes, referentes aos elementos dessa formulação:

- Os elementos do conjunto AA são as funções adaptativas φ, que apresentam os seguintes componentes (no caso mais trivial, φ qualquer pode ser a função adaptativa nula λ):
  - Cabeçalho da função adaptativa φ, contendo:
    - Nome φ, pelo qual a função adaptativa é referenciada.
    - Ênupla ordenada (ρ_1, ρ_2, ..., ρ_p), de p≥0 parâmetros formais, similares às de funções em linguagens de programação.
    - Conjunto {υ_1, υ_2, ..., υ_v} de v ≥ 0 variáveis, cujo valor é preenchido uma só vez por ações adaptativas de consulta.
    - Conjunto {χ_1, χ_2, ..., χ_g} de g ≥ 0 geradores, cujo valor é preenchido com novos valores a cada chamada da função adaptativa.
  - Corpo ($\beta\mu$, Δ, $\alpha\nu$) da função adaptativa φ, compreendendo:
    - Chamada (*call*) $\beta\mu$ de uma função adaptativa anterior (μ), cuja execução esteja prevista para antes da aplicação do conjunto de ações adaptativas elementares que formam o núcleo da função adaptativa φ.
    - O núcleo, a parte essencial da função adaptativa φ, que consiste de um conjunto invariável Δ = {δ_1, δ_2, ..., δ_e}, contendo e ≥ 0 ações adaptativas elementares δ_i, 0≤i<e. Representa a coleção das alterações a serem aplicadas ao dispositivo adaptativo AD toda vez que a função adaptativa φ for executada.
      Cada ação adaptativa elementar δ_i, 0≤i<e, é denotada no formato seguinte:
      $$\delta_i = \otimes [\, r \,].$$
      O símbolo ⊗ representa um operador, aplicado à regra r e ao conjunto RA_t de regras que definem o comportamento do dispositivo AD (conjunto RA, com o conteúdo que nele se encontra no passo t de operação de AD), podendo ser um dos seguintes:
      - \+ (inserção): inclui uma nova regra r em RA_t.
      - – (remoção): elimina a regra r do conjunto RA_t.
      - ? (consulta): pesquisa o conjunto RA_t usando como padrão de busca a regra r indicada, e preenchendo variáveis e parâmetros com os resultados da busca.
    - Chamada (*call*) $\alpha\nu$ de uma função adaptativa posterior (ν), cuja execução esteja prevista para após a aplicação do conjunto das ações adaptativas elementares definidas no conjunto Δ.
  - Para completar esta descrição da formulação das funções adaptativas, resta definir o formato para as chamadas (*calls*) de funções adaptativas φ, (denotadas na presente publicação como $\beta\varphi$ ou $\alpha\varphi$, conforme se trate de ação adaptativa anterior ou posterior, respectivamente) com p ≥ 0 parâmetros formais ρ_1, ρ_2, ..., ρ_p .
    Sendo θ_1, θ_2, ..., θ_p os argumentos correspondentes, onde os θ_i representam valores posicionalmente correspondentes aos parâmetros formais ρ_i (i = 1, ... , p) aos quais se referem, uma chamada (*call*), por exemplo, $\alpha\varphi$, assume o formato seguinte:
    $$\alpha\varphi = \varphi\,(\theta_1, \theta_2, ..., \theta_p).$$
    Obviamente, para p = 0, tem-se uma função sem parâmetros, e neste caso o formato da sua chamada (*call*) se reduz a:
    $$\alpha\varphi = \varphi\,(\,).$$
- O conjunto RA é variável por ação das funções adaptativas (elementos de AA), assumindo sucessivamente os valores RA_0, RA_1, ..., RA_t, (t ≥ 0) a cada passo do processamento de AD.
  Os valores RA_i assumidos por RA (i ≥ 0) representam, na ocasião do passo i do dispositivo AD, a relação de mudança de configuração que define o comportamento do dispositivo nessa oportunidade:
  $$RA_i \subseteq AA \times (C \times (S \cup \{\varepsilon\}) \times C) \times AA,$$
  O produto cartesiano (C × (S∪{ε}) × C) representa o universo das possíveis regras r_j do dispositivo não-adaptativo subjacente.
  Os elementos ra_j são as regras adaptativas cujo conjunto RA_i define o comportamento do dispositivo AD ao passo i de operação, e têm a forma:
  $$ra_j = (\beta b_j, r_j, \alpha a_j),$$

onde βb_j, $\alpha a_j \in AA$ são, respectivamente, as chamadas (*calls*) anteriores e posteriores, possivelmente paramétricas, de funções adaptativas b_j, $a_j \in AA$.

A regra não-adaptativa subjacente r_j tem a forma

$$r_j = (c_j, s, c'_j) \in R, \text{ com } c_j, c'_j \in C; s \in S,$$

podendo também ser denotada como

$$(c_j, s) \rightarrow c'_j,$$

realçando ser s o estímulo que, aplicado ao dispositivo na configuração c_j, leva-o à sua nova configuração c'_j.

$$c_j \Rightarrow_s c'_j$$

A linguagem definida por D é representada por L(D):

$$L(D) = \{w \in S^* \mid c_i \Rightarrow_w^* c, c \in A\},$$

onde o x* denota o fecho de Kleene de um conjunto ou operador x.

Sendo AA o conjunto (finito, fixo) de funções adaptativas do dispositivo, e R_t o conjunto das regras (descritas em sua formulação não-adaptativa) que definem seu dispositivo subjacente na ocasião do seu t-ésimo passo de execução ($t \geq 0$), o seu mecanismo adaptativo básico AM_t nesse passo de operação pode ser interpretado como:

$$AM_t \subseteq AA \times R_t \times AA,$$

determinando, naquela ocasião, o conjunto de regras (adaptativas) de operação do dispositivo com adaptatividade básica.

As regras adaptativas obtidas pela utilização desse mecanismo adaptativo básico assumem a forma ar = (βb, r, αa), conforme formulado anteriormente, com $r \in R_t$, sendo que βb e αa, que denotam chamadas (*calls*) das funções adaptativas b, $a \in AA$, representam as chamadas (*calls*) das funções adaptativas previstas para serem executadas respectivamente antes e depois da aplicação da regra $r \in R_t$.

O princípio de operação desse mecanismo adaptativo básico é o que foi apresentado: partindo-se da configuração inicial do dispositivo, aplicam-se ciclicamente as regras adaptativas, até que não haja movimento possível ou até que, esgotados os estímulos fornecidos ao dispositivo, seja atingida alguma particular configuração de término de operação (eventualmente indicativa de sucesso ou fracasso do processamento da seqüência de estímulos do dispositivo).

H. Operação de um Dispositivo com Adaptatividade Básica

Um dispositivo dotado de adaptatividade básica opera de acordo com o algoritmo esboçado a seguir. Sua lógica é relativamente simples, e espelha o funcionamento anteriormente discutido para os dispositivos guiados por regras, aos quais tenha sido acrescentada uma camada adaptativa básica:

DISPOSITIVO ADAPTATIVO BÁSICO – OPERAÇÃO

1. Atribuir valores iniciais aos elementos:
   - $t = 0$ (passo inicial 0).
   - $c_t = c_0$ (configuração inicial, fixa).
   - $w_t = w_0 = s_1 s_2 s_3 \ldots$ (cadeia de estímulos de entrada).
2. Se $w_t = \varepsilon$ (seqüência de estímulos esgotada), desviar para o passo 7, caso contrário, incrementar t e então extrair de w_t o próximo estímulo s_t.
3. Para cada uma das configurações correntes c_t que estejam sendo paralelamente processadas, obter de C o sub-conjunto CA de regras adaptativas a ela compatíveis.
   - Se $CA = \varnothing$, rejeitar w.
   - Se $CA = \{ (\beta b_p, (c_t, s, c'_p), \alpha a_p) \}$, aplicar deterministicamente a única regra nele contida conforme os passos 4, 5 e 6, levando AD a uma nova configuração c_{t+1}.
   - Se $CA = \{ ar_k = (\beta b_k, (c_t, s, c'_k), \alpha a_k) \mid nr_k = (c_t, s, c'_k)$ é uma regra do dispositivo não-adaptativo subjacente, $c'_k \in C, k = 1, \ldots, n, n>1\}$, aplicam-se em paralelo as regras ar_p desse conjunto, conforme os passos 4, 5 e 6, levando AD em paralelo às correspondentes configurações c_{t+1} do conjunto $\{c'_1, \ldots, c'_n\}$.
4. Se $\beta b_p = \lambda(\,)$ (a ação adaptativa nula) desviar para o passo 5; Se não, aplicar inicialmente βb_p. Caso a regra ar_p resulte removida pela aplicação de βb_p, desviar para o passo 3, descontinuando ar_p. Caso contrário AD terá atingido uma configuração estável, e neste caso, desviar para o passo 2.
5. Aplicar a regra nr_p à configuração intermediária corrente, levando AD a uma nova configuração estável $c_{t+1} = c'_p$.
6. Se $\alpha a_p = \lambda(\,)$ (a ação adaptativa nula) desviar para o passo 7; Se não, aplicar finalmente a ação adaptativa αa_p, e então, desviar para o passo 2.
7. Caso algum $c_{t+1} \in F$ tenha sido atingido, então AD aceita w, caso contrário, AD rejeita w; em qualquer caso, encerra-se aqui a execução deste algoritmo.

I. Adaptatividade Multinível Hierárquica

Descreve-se nesta seção o conceito de adaptatividade hierárquica e multinível, pelo qual se acrescenta à adaptatividade básica a possibilidade de alteração controlada das funções adaptativas e também sua hierarquização [4].

Segundo essa nova organização, os mecanismos responsáveis pela automodificação dos dispositivos adaptativos utilizam-se, para isso, de várias camadas hierarquicamente estruturadas, de forma que cada camada adaptativa tem acesso direto de modificação apenas à sua camada vizinha, de hierarquia imediatamente inferior.

Dessa maneira, confinam-se as alterações possíveis, evitando-se modificações indiscriminadas nos mecanismos adaptativos, restringindo o acesso apenas ao nível vizinho inferior, o que reduz significativamente a complexidade do dispositivo como um todo.

A introdução da hierarquia não limita de forma alguma o dispositivo resultante, pois continuam viáveis quaisquer acessos necessários a camadas não vizinhas, desde que se guarde o protocolo de encaminhar convenientemente a informação necessária através das camadas intermediárias.

Assim sendo, um dispositivo com adaptatividade multinível hierárquica pode ser formulado indutivamente, e é constituído essencialmente dos mesmos elementos que formam os dispositivos com adaptatividade básica:

- Seu dispositivo não-adaptativo subjacente, o qual constitui a base da sua formulação indutiva, e nessa hierarquia, representa uma camada de adaptatividade de nível zero.
- O mecanismo adaptativo hierárquico multi-nível, que permite que sejam causadas alterações no conjunto de regras que define o dispositivo não-adaptativo subjacente, promovendo assim sua adaptatividade por meio de um comportamento dinâmico e hierárquico.
- A camada adaptativa, que se apresenta como uma hierarquia de níveis, na qual cada nível se sobrepõe ao nível imediatamente anterior, e assim, o primeiro deles, de primeiro nível, se sobrepõe diretamente ao dispositivo subjacente não-adaptativo, ou seja, de nível zero.
- As funções adaptativas dos dispositivos com adaptatividade hierárquica multinível, as quais, além do usual acesso à modificação do conjunto de regras adaptativas do próprio nível, podem também reprogramar as funções adaptativas do nível imediatamente inferior, quando existirem.
- A associação das chamadas de funções adaptativas às regras do mesmo nível, para execução sempre que a regra for aplicada, que é efetuada em cada nível, pelo mecanismo adaptativo.
- O nível de adaptatividade: um dispositivo adaptativo é dito de nível N quando suas regras adaptativas apresentam nível máximo N de adaptatividade.

Da mesma forma como foi feito no caso dos dispositivos com adaptatividade básica, os dispositivos subjacentes de nível zero são obtidos a partir dos dispositivos não-adaptativos por um simples ajuste notacional, o mesmo podendo ser feito para outros níveis, compatibilizando a apenas a notação, sem modificar o seu comportamento.

J. Mecanismo Adaptativo Hierárquico de Nível N

Nesta seção, descreve-se um mecanismo adaptativo hierárquico HM de nível N. Trata-se de uma variante do mecanismo adaptativo básico AM, anteriormente proposto, ao qual foram incorporados os elementos necessários à formulação de dispositivos com estrutura adaptativa hierárquica e multinível. Essencialmente, as alterações atingiram a notação, que agora explicita o nível da camada adaptativa a que se refere, e introduziram uma definição indutiva das camadas adaptativas, responsável pela formulação da estrutura hierárquica desejada.

MECANISMO ADAPTATIVO HIERÁRQUICO

- Um mecanismo adaptativo hierárquico HM, de nível $N \geq 0$, no seu passo $t \geq 0$ de operação, é composto de N+1 camadas adaptativas de nível k:
  $$HM(k, t) \mid 0 \leq k \leq N.$$
- Assim, pode-se definir (indutivamente) o mecanismo adaptativo HM caracterizando cada uma de suas camadas adaptativas HM(k, t) de nível k, $0 \leq k \leq N$, no passo t:
  - Para $k = 0$, sendo λ a função adaptativa nula (sem parâmetros), e R_t o conjunto de regras do dispositivo subjacente (não-adaptativo) em seu passo t de operação, define-se a camada de nível 0 do mecanismo adaptativo HM:
    $$HM(0, t) = \{\lambda\} \times R_t \times \{\lambda\}.$$
    Dessa forma, sendo $r \in R_t$ uma regra do dispositivo não-adaptativo HM(0,t) subjacente, pode-se denotar a regra r^0 equivalente, em sua formulação adaptativa de nível zero:
    $$r^0 = (\lambda( ),\ r,\ \lambda( )).$$
  - Para $0 < k \leq N$, sendo HA(k,t) o conjunto de todas as funções adaptativas de nível k, na forma em que se encontrarem no passo t de operação do dispositivo HD (para $0 < k \leq N$):
    $$HM(k, t) \subseteq HA(k, t) \times HM(k-1, t) \times HA(k, t).$$
    Sendo b^k, $a^k \in HA(k,t)$ e $r^{k-1} \in HM(k-1,t)$ uma regra adaptativa de nível k-1, HM(k,t) constrói uma regra r^k de nível k, associando-lhe ações adaptativas anteriores e posteriores βb^k e αa^k, respectivamente chamadas (*calls*) das funções adaptativas b^k e a^k, de nível k (para $0 < k \leq N$):
    $$r^k = (\beta b^k, r^{k-1}, \alpha a^k).$$
    Observe-se que, de acordo com a natureza das funções b^k e a^k (eventualmente vazias), é possível que as correspondentes chamadas βb^k e αa^k sejam paramétricas.
    Note-se ainda que, sendo HD um dispositivo de nível N, não podem existir funções adaptativas de nível $k > N$, logo, tampouco poderão existir as correspondentes camadas adaptativas HM(k, t):
    $$HM(k, t) = \varnothing \times HM(k-1, t) \times \varnothing = \varnothing \text{ , para } k > N.$$
  - As regras que definem um dispositivo adaptativo hierárquico multinível podem, portanto, ser assim denotadas:
    $$(\beta b_N, (..., (\beta b_1, (\lambda( ), (c, s, c'), \lambda( )), \alpha a_1), ...), \alpha a_N)$$
    onde:
    - (c, s, c') se refere a uma regra do dispositivo subjacente não-adaptativo de HD.
    - $r^0 = (\lambda( ), (c, s, c'), \lambda( ))$ é essa mesma regra, expressa na formulação adaptativa equivalente de nível zero.
    - $r^k = (\beta b_k, r^{k-1}, \alpha a_k)$ agrega, à regra r^{k-1} de nível k-1, uma camada adaptativa de nível k ($1 \leq k \leq N$).
- Cada camada HM(k,t) de nível k ($0 < k \leq N$), do mecanismo adaptativo HM, dispõe de um conjunto HA(k,t) de funções adaptativas do nível k, com $\lambda \in HA(k,t)$, com as quais é possível:
  - Alterar para HM(k,t+1) o conjunto HM(k,t) das regras adaptativas de seu próprio nível k ($0 < k \leq N$), através de ações de inclusão e de exclusão de regras.
  - Alterar o funcionamento de alguma função adaptativa $\varphi^{k-1}(t) \in HA(k-1,t)$, do nível imediatamente inferior k-1, resultando assim uma nova versão $\varphi^{k-1}(t+1)$ dessa função, e acarretando, como conseqüência, a obtenção de uma versão modificada desse mesmo conjunto de funções ($1 < k \leq N$):
    $$HA(k-1,t+1) = (HA(k-1,t) - \{\varphi^{k-1}(t)\}) \cup \{\varphi^{k-1}(t+1)\}.$$

  Observe-se que, nunca existindo funções adaptativas em níveis de adaptatividade $k > N$ (por definição), o conjunto HA(N,t+1) será o próprio HA(N,t), ou seja, HA(N,t) não varia, para todo $t \geq 0$.
  Note-se também que, embora a presente reformulação não contemple a criação dinâmica de novas funções adaptativas, limitando-se apenas à alteração do comportamento de funções adaptativas já existentes, não há motivo para que tal restrição seja preservada, podendo incorporar-se, se necessário em futuras reformulações dos dispositivos adaptativos.

- Está faltando ainda definir melhor as funções φ adaptativas de nível k, que compõem os conjuntos HA(k,t). Como essas funções podem ser alteradas dinamicamente pelo mecanismo adaptativo, será usada a notação φ^k_t para designar uma função adaptativa φ, tal como se encontra no passo t da operação de HD.
  - Uma função adaptativa qualquer, utilizada pelo dispositivo HD, apresenta os seguintes elementos:
    - Nome φ.
    - Nível k da camada adaptativa HM(k,t) a que pertence.
    - Passo t da operação do dispositivo HD, que a utiliza.
    - Denota-se φ^k_t, como é referenciada doravante neste texto.
    - Seqüência $(\rho_1, \rho_2, ..., \rho_p)$ de $p \geq 0$ parâmetros formais, similares aos utilizados pelos procedimentos e funções encontrados nas linguagens usuais de programação.
    - Conjunto $\{\upsilon_1, \upsilon_2, ..., \upsilon_v\}$ de $v \geq 0$ variáveis, componentes básicos da formulação, aos quais são associados valores, uma só vez, como resultado da execução de operações elementares de consulta ou de remoção especificadas na função adaptativa φ^k_t.
    - Conjunto $\{\chi_1, \chi_2, ..., \chi_g\}$ de $g \geq 0$ geradores, componentes básicos da formulação, aos quais são associados valores, uma só vez, toda vez que a função adaptativa φ^k_t é chamada.
    - Chamada (*call*) βb^k de uma função adaptativa anterior b^k, do mesmo nível k de adaptatividade, para ser executada antes do conjunto das ações adaptativas elementares que constituem a parte essencial da função adaptativa φ^k_t.
    - Conjunto $\Delta^k = \{\delta^k_1, \delta^k_2, ..., \delta^k_{ek}\}$ contendo $e^k \geq 0$ ações adaptativas elementares δ^k_i, $1 \leq i < e^k$, todas do mesmo nível k de adaptatividade. Indica a coleção das alterações específicas a serem aplicadas a cada execução da função adaptativa φ^k_t pela camada adaptativa HM(k,t) de nível k, às regras do dispositivo adaptativo hierárquico HD.
      Note-se que este conjunto Δ^k não é obrigatoriamente constante, podendo ser modificado durante a operação de HD pela eventual iniciativa de funções adaptativas do nível k+1 (para $1 \leq k < N$).
      Nas fórmulas a seguir, o símbolo $\otimes$ denota um dos operadores seguintes, aplicado ao conjunto apropriado –

conjunto CA de regras, definido em HM(k,t), ou então, conjunto Δ^{k-1} de ações adaptativas elementares, que caracteriza uma função adaptativa $\varphi^{k-1}{}_t$:

- \+ inserção: inclui um novo elemento no conjunto.
- – remoção: elimina do conjunto o elemento indicado.
- ? consulta: pesquisa o conjunto usando como padrão de busca o elemento indicado, e preenchendo variáveis e parâmetros com os resultados da busca.

Cada uma das ações adaptativas elementares, $\delta^k{}_i$, $0 \le i < e^k$, de nível k, pode apresentar-se em uma das duas formas seguintes:

- As que aplicam o operador $\otimes$ diretamente à regra r^k do conjunto CA de regras do mesmo nível k, do dispositivo HD (para $0 < k \le N$):
$$\delta^k{}_i = \otimes [\, r^k \,].$$
- As que operam sobre o comportamento das funções adaptativas de nível k-1 (para $1 < k \le N$):
$$\delta^k{}_i = \otimes \varphi^{k-1}{}_t [\, \delta^{k-1}{}_j \,],$$
onde $\delta^k{}_i \in \Delta^k$ e $\delta^{k-1}{}_j \in \Delta^{k-1}$, e se refere a $\varphi^{k-1}{}_t$.
Neste caso, uma função adaptativa de nível k, em execução, promove a aplicação da ação adaptativa elementar $\delta^k{}_i \in \Delta^k$, a qual, por sua vez, aplica o operador $\otimes$ à ação adaptativa elementar $\delta^{k-1}{}_j$ destacada entre colchetes nesta notação, e referente à definição da função adaptativa $\varphi^{k-1}{}_t$, de nível k-1.

- Chamada (*call*) $\alpha a^k{}_t$ de uma função adaptativa posterior $a^k{}_t$, do mesmo nível k de adaptatividade, para ser executada após o conjunto Δ^k das ações adaptativas elementares que compõem a parte essencial da função adaptativa $\varphi^k{}_t$.

- No caso mais trivial, uma função adaptativa qualquer $\varphi^k{}_t$, de nível $0 < k \le N$, pode eventualmente ser a função adaptativa sem parâmetros nula λ, cuja execução não exerce qualquer alteração sobre o dispositivo HD.
- Para completar a formulação, apresenta-se uma definição de formato para as chamadas (*calls*) $\alpha a^k{}_t$ das funções adaptativas $a^k{}_t$, com $p \ge 0$ parâmetros formais $\rho_1, \rho_2, ..., \rho_p$.
Sendo $\theta_1, \theta_2, ..., \theta_p$ os argumentos correspondentes, onde os θ_i denotam valores (que podem referir-se a quaisquer instâncias de elementos componentes da formulação do dispositivo HD em questão) posicionalmente associados aos respectivos parâmetros formais ρ_i (i = 1, ... , p), uma chamada (*call*) αa^k assume o formato seguinte:
$$\alpha a^k{}_t = a^k{}_t (\theta_1, \theta_2, ..., \theta_p).$$
Obviamente, quando p = 0, isso indica que $a^k{}_t$ é uma função adaptativa sem parâmetros.
Por exemplo, quando $a^k{}_t$ é a função adaptativa nula λ, o formato da sua chamada (*call*) $\alpha\lambda$ se reduz a:
$$\alpha\lambda = \lambda( ).$$

K. Dispositivo com Adaptatividade Hierárquica de Nível N

A exemplo do que foi visto para dispositivos com adaptatividade básica, é possível descrever um dispositivo HD, guiado por regras, dotado do recurso de adaptatividade hierárquica, como uma sêxtupla, conforme a formulação anteriormente utilizada:

(C, RA, S, AA, c_0, RA_0, A),

onde, embora todos os elementos tenham a mesma conotação conceitual anteriormente estudada, os elementos que envolvem as funções adaptativas devem ser reinterpretados:

- RA e RA_0 são constituídos de regras que apresentam agora formato hierárquico multinível.
- AA passa a conter funções adaptativas capazes de modificar o funcionamento de outras funções adaptativas.

Por questão de clareza, não será usada a sêxtupla acima nesta formulação, pois torna-se conveniente evidenciar a camada adaptativa do novo dispositivo, realçando o caráter estratificado do seu formalismo hierárquico.

DISPOSITIVO ADAPTATIVO HIERÁRQUICO – FORMULAÇÃO

Formula-se a seguir um dispositivo guiado por regras com adaptatividade hierárquica:

HD = (ND_0, HM, N), onde:

- $N \ge 0$ representa o nível de adaptatividade do dispositivo HD. Note-se que:
  - para N=0, esta formulação pode ser vista como uma denotação alternativa geral para dispositivos adaptativos, aplicável até mesmo quando HD não é adaptativo.
  - Para N=1, essa formulação oferece uma notação alternativa que se aplica ao caso particular em que HD se refere a um simples dispositivo com adaptatividade básica.
- ND_0 representa um dispositivo subjacente inicial (em formulação não-adaptativa) ao qual se está incorporando a adaptatividade:
ND_0 = (C, R_0, S, c_0, A), onde:
  - C é o conjunto de todas as possíveis configurações do dispositivo ND_0
  - R_0 representa o conjunto das regras (em formulação não-adaptativa) que definem, no passo de operação 0 de HD, o comportamento do dispositivo subjacente ND_0, cujas alterações são promovidas pelo mecanismo adaptativo HM.
  - S é o conjunto dos estímulos que geram as movimentações (mudanças de configuração) do dispositivo ND_0.
  - $c_0 \in C$ é a configuração inicial (única) do dispositivo ND_0.
  - $A \subseteq C$ é o conjunto de todas as configurações finais (de aceitação) do dispositivo ND_0.
- HM é o mecanismo adaptativo hierárquico multinível, conforme definido anteriormente, o qual, atrelando em camadas estratificadas, chamadas de funções adaptativas às regras do dispositivo subjacente ND_0, capacita HD a efetuar auto-modificações comportamentais por dois diferentes caminhos: pela modificação do conjunto de regras ou então pela alteração do comportamento de funções adaptativas existentes.

L. Operação de um Dispositivo com Adaptatividade Hierárquica de nível N

Um dispositivo dotado de adaptatividade hierárquica de nível N opera de acordo com o algoritmo esboçado a seguir:

DISPOSITIVO ADAPTATIVO HIERÁRQUICO – OPERAÇÃO

1. Posicionar em seus valores iniciais os elementos:
   - $t = 0$ (passo inicial 0).
   - $c_t = c_0$ (configuração inicial, fixa).
   - $w_t = w_0 = s_1 s_2 s_3 \ldots$ (cadeia de estímulos de entrada).
   - o conjunto inicial de regras adaptativas hierárquicas
   - os conjuntos de ações adaptativas elementares associadas às funções adaptativas, em cada nível

2. Se $w_t = \varepsilon$ (seqüência de estímulos esgotada), desviar para o passo 7, caso contrário, incrementar t e então extrair de w_t o próximo estímulo s_t.
3. Para cada uma das configurações correntes c_t que estejam sendo paralelamente processadas, determinar, a partir dos conjuntos HM(k, t) ($0 < k \leq N$), os sub-conjuntos CA correspondentes de regras adaptativas de nível k, a elas compatíveis. Em cada caso,
   - Se CA = ∅, rejeitar w.
   - Se CA = { $(\beta_p, (c_t, s, c'_p), \alpha_p)$ }, aplicar essa única regra deterministicamente de acordo com os passos 4, 5 e 6, levando HD a uma nova configuração c_{t+1}.
   - Se CA = { $ar_k = (\beta_k, (c_t, s, c'_k), \alpha_k) \mid nr_k = (c_t, s, c'_k)$ é uma regra do dispositivo não-adaptativo subjacente, $c'_k \in C$, k = 1, … , n, n>1}, aplicam-se em paralelo as regras ar_p desse conjunto, conforme os passos 4, 5 e 6, levando HD em paralelo às configurações c_{t+1} do conjunto { $c'_1, c'_2, \ldots, c'_n$ }.
4. Se $\beta_p = \lambda(\,)$ (a ação adaptativa nula) desviar para o passo 5; Se não, aplicar inicialmente β_p. Caso a regra ar_p resulte removida pela aplicação de β_p, desviar para o passo 3, descontinuando ar_p. Caso contrário HD terá atingido uma configuração estável, e neste caso, desviar para o passo 2.
5. Aplicar a regra nr_p à configuração intermediária corrente, levando HD a uma nova configuração estável $c_{t+1} = c'_p$.
6. Se $\alpha_p = \lambda(\,)$ (a ação adaptativa nula) desviar para o passo 7; Se não, aplicar finalmente a ação adaptativa α_p, e então, desviar para o passo 2.
7. Caso algum $c_{t+1} \in F$ tenha sido atingido, então HD aceita w, caso contrário, HD rejeita w; em qualquer caso, encerra-se aqui a execução deste algoritmo.

EXEMPLO

Apresenta-se a seguir um pequeno caso particular de autômato finito adaptativo de nível 3. Destina-se apenas a ilustrar o que foi apresentado, e não constitui aplicação prática, que seria muito mais complexa, tornando proibitivo apresentá-la neste texto. Neste exemplo, nenhuma das funções adaptativas utilizadas é paramétrica:

R_0 = { (λ(), (1, a, 2), λ()), (λ(), (2, b, 1), λ()), (λ(), (2, ε, 3), λ()) }
$A_0 = \varnothing$
$B_0 = \varnothing$

R_1 = { (λ(), (λ(), (1, a, 2), λ()), a_1()),
(b_1(), (λ(), (2, b, 1), λ()), λ()),
(λ(), (λ(), (2, ε, 3), λ()), λ()) }
A_1 = { a_1 }
B_1 = { b_1 }

ações adaptativas elementares da função adaptativa a_1:
{ - [(λ(), (λ(), (1, a, 2), λ()), a_1 ())],
+ [(λ(), (λ(), (2, a, 1), λ()), λ())],
+ [(λ(), (λ(), (1, b, 2), λ()), λ())] }

ações adaptativas elementares da função b_1:
{ - [(b_1(), (λ(), (2, b, 1), λ()), λ ())],
+ [(λ(), (λ(), (2, b, 1), λ()), λ())],
+ [(λ(), (λ(), (1, a, 2), λ()), λ())] }

R_2 = { (λ(), (λ(), (λ(), (1, a, 2), λ()), a_1()), a_2()),
(λ(), (b_1(), (λ(), (2, b, 1), λ()), λ()), λ()),
(λ(), (λ(), (λ(), (2, ε, 3), λ()), λ()), λ()) }
A_2 = { a_2 }
$B_2 = \varnothing$

ações adaptativas elementares da função adaptativa a_2:
{ - b_1 [+ [(λ(), (λ(), (2, b, 1), λ()), λ())]],
+ a_1[- [(λ(), (λ(), (1, b, 2), λ()), λ ())]],
- [(λ(), (b_1(), (λ(), (2, b, 1), λ()), λ()), λ())] }

R_3 = { (λ(), (λ(), (λ(), (λ(), (1, a, 2), λ()), a_1()), a_2()), λ()),
(b_3(), (λ(), (b_1(), (λ(), (2, b, 1), λ()), λ()), λ()), λ()),
(λ(), (λ(), (λ(), (λ(), (2, ε, 3), λ()), λ()), λ()), λ()) }
$A_3 = \varnothing$
B_3 = { b_3 }

ações adaptativas elementares da função adaptativa b_3:
{ - b_2 [+ a_1 [- [(λ(), (λ(), (1, b, 2), λ()), λ())]]],
- [(λ(), (λ(), (λ(), (λ(), (1, a, 2), λ()), a_1()), a_2 ()), λ())]
}

X. Observações Finais

As formulações apresentadas nesta publicação, para os dispositivos guiados por regras – não-adaptativos, com adaptatividade básica e com adaptatividade hierárquica – permitem exprimir e tratar de maneira uniforme os elementos dinâmicos das formulações referentes a dispositivos automodificáveis.

Correspondem a generalizações de ampliação das formulações que têm sido tradicionalmente utilizadas, permitindo reutilizá-las sem que se torne necessário aplicar-lhes modificações significativas, mantendo portanto uma relativa compatibilidade com as notações em uso ao mesmo tempo que se oferecem meios para a expressão de notações estratificadas, voltadas para formulações hierárquicas que podem servir-se de funções adaptativas com comportamento variável.

Maiores informações sobre trabalhos desenvolvidos sobre os formalismos adaptativos se encontram na página web do Laboratório de Linguagens e Técnicas Adaptativas [5].

XI. Referências

[1] Neto, J.J. - Contribuição à metodologia de construção de compiladores. São Paulo, 1993, 272p. Tese (Livre-Docência) Escola Politécnica, Universidade de São Paulo.

[2] Neto, J.J. - Adaptive automata for context-dependent languages. ACM SIGPLAN Notices, v.29, n.9, p.115-24, 1994.

[3] Neto, J.J – Adaptive devices – general formulation and case study – CIAA 2001 - 6th International Conference on Implementation and Application of Automata - Lecture Notes In Computer Science; Vol. 2494, pp. 234-250 – Pretoria, South Africa, 2001

[4] Neto, J.J. - Adaptive Technology and its Applications - in Dopico, J.R.R., J.D. de la Calle, A.P. Sierra (editors) - Encyclopedia of Artificial Intelligence - IGI Global, 2009, pp. 37-44 ISBN 978-1-59904-849-9 (hardcover) and 978-1-59904-850-5 (ebook)

[5] http://www.pcs.usp.br/~lta/ Página web do LTA – Laboratório de Linguagens e Técnicas Adaptativas.

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.

COMUNICAÇÕES

Adaptive Dynamic Discovery Language

R. G. Dutra, R. Silva, M. Martucci Jr., W. Vicente Ruggiero

Resumo - O objetivo deste artigo é propor uma meta linguagem de tradução de regras, que permita a descoberta dinâmica de serviços WEB, baseada nos atributos não funcionais que caracterizam estes serviços. Como gerador das regras de produção desta linguagem, será utilizada a ferramenta *Adaptive Fuzzy Neural Tree Network* (*AFNTN*).

Como resultado desta meta linguagem espera-se a criação de documentos no padrão XML, que possam ser utilizados por outras linguagens de definição, gerenciamento e oferta de serviços WEB, complementando estas linguagens com funções adaptativas para a descoberta dinâmica.

Palavras Chave – Linguagens de programação, Sistemas Adaptativos, Árvores de Decisão, Redes Neurais Artificiais, Lógica Difusa.

I. INTRODUÇÃO

Atualmente um grande problema para o gerenciamento de serviços WEB, é o processo de descoberta, procurando o serviço de acordo com as suas funcionalidades. Geralmente os desenvolvedores acessam algum registro em um repositório e a busca é realizada por meio de palavras chave, que possam traduzir a funcionalidade que se almeja. É o mesmo problema enfrentado nos sites de busca na WEB.

A solução para localizar um serviço de acordo com seus atributos não funcionais, tais como qualidade do serviço, performance, aspectos de segurança, entre outros, além dos atributos funcionais, é a combinação de algoritmos de prospecção de dados [21].

Para finalizar o processo de descoberta de serviços, é necessário traduzir as regras que classificam os mesmos em uma linguagem de programação aberta, como XML (Extensible Markup Language), que permita a leitura ou interpretação destes serviços por outros algoritmos para consumo dos mesmos.

Diversas linguagens de programação baseadas no padrão XML foram propostas na literatura [1], destacando-se entre elas:

- WSDL (*WEB Services Definition Language*), criada para descrever a especificação de serviços *WEB*, baseado em atributos funcionais. Atualmente bastante utilizada para a descoberta estática de serviços.
- OWL-S (*Ontology WEB Language for Services*), antigamente conhecida como DAML-S (*Darpa Agent Markup Language for WEB Services*) criada para descrever a especificação de servicos WEB, baseado em ontologias de atributos funcionais e não funcionais.
- WSOL (*WEB Services Offering Language*), criada para oferecer serviços *WEB*, baseado em atributos funcionais e não funcionais, para composição e coreografia de serviços.

Segundo [1], a linguagem WSOL é compatível com WSDL, acrescentando-se atributos não funcionais e permitindo composição dinâmica entre diversos serviços que possuem atributos funcionais similares, necessidade primordial da arquitetura SOA, não coberta pela linguagem OWL-S.

Entretanto, nenhuma das linguagens citadas ou encontradas na literatura até o presente momento, permitem a realização do processo dinâmico de descoberta de serviços *WEB*, baseado em características não funcionais.

Este artigo objetiva a exploração das ferramentas de descoberta dinâmica *(dynamic discovery),* visando a solução deste problema, através da combinação de métodos de Inteligência Artificial, tais como Redes Neurais Artificiais, Árvores de Decisão Adaptativas e Lógica *Fuzzy* em um modelo denominado de *Adaptive Fuzzy Neural Tree Network* (AFNTN) [22].

O problema alvo consiste em classificar serviços WEB, através dos atributos não funcionais que descrevem estes serviços conforme linguagem WSOL, de forma não supervisionada, ou seja, o número de classes e os atributos utilizados para definição das mesmas não é conhecido a priori, cabendo a AFNTN sua determinação e geração de regras de produção de uma linguagem de tradução para XML.

O conteúdo deste artigo está organizado de forma a apresentar inicialmente uma introdução sobre a teoria de linguagens formais de programação, seguida da descrição de mecanismos adaptativos para indução de árvores decisão e a descrição do algoritmo de aprendizado não supervisionado. Em seguida é discutido como tratar incerteza nos dados e a modelagem para implementação da ferramenta proposta neste trabalho. Nas seções finais são apresentados resultados e conclusões obtidas através da aplicação da linguagem gerada pela *AFNTN*.

I. CONCEITOS SOBRE LINGUAGENS DE PROGRAMAÇÃO

Serviços são componentes de software que representam um processo, entidade, atividade ou tarefa. Existem quatro tipos de Serviços:

- Básicos: que representam os elementos básicos de um processo de negócio, como Entidades e Tarefas básicas de negócios;
- Intermediários: são o único tipo de Serviço mais orientados a tecnologia em uma arquitetura orientada a serviços (SOA). Fornecem pontes, conversores ou funcionalidades adicionais aos demais serviços;

- Processos: são os serviços que representam de forma direta um processo ou atividade de negócio, do início ao fim.
- Públicos: extensão aos serviços do tipo Processo que possibilita sua exposição para clientes (usuários) que estejam fora das fronteiras de um determinado domínio.

Todo serviço, independente de seu tipo, é sempre composto por três partes principais. A primeira parte é um *contrato*, um acordo que é fechado entre os consumidores e seus provedores. O contrato explica os propósitos, contexto, regras de utilização, restrições, níveis de serviço esperados, além de apresentar uma definição formal da interface. Tal *interface* é implementada em separado e constitui o sendo segundo elemento de construção de um serviço, representando o único meio de comunicação com o mesmo. A terceira e última parte de um serviço é sua *implementação* propriamente dita, através da realização da lógica do negócio e acesso e manutenção de seus dados, de forma a atender todos os objetivos fixados no contrato.

Um *Repositório* de serviços armazena todos os *Contratos* dos Serviços disponíveis, o que o torna o ponto de partida para utilização destes. Além dos *Contratos*, o *Repositório* pode armazenar informações adicionais e mais específicas acerca dos serviços, como localização física, restrições de uso e segurança, etc. Apesar de ser apresentado por alguns autores como um elemento opcional em uma SOA, o *Repositório* pode ser fator crítico de sucesso em grandes implementações, principalmente naquelas que envolverem a disponibilização de serviços do tipo Público. Neste contexto, os serviços podem ser publicados na *World Wide Web*, sendo denominados de *WEB Services*.

O UDDI (*Universal Description, Discovery, and Integration*) especifica um mecanismo centralizado para os provedores de *WEB Services* anunciarem a existência de seus serviços e para os consumidores poderem localizar os serviços de seu interesse,ie, um *Repositório* de *WEB Services*. Este registro funciona como uma grande lista telefônica cujos serviços são catalogados.

O principal componente do UDDI é um arquivo XML (*Extensive Markup Language*) usado para descrever serviços *WEB*. A informação deste arquivo consiste em três componentes:

- ***Páginas brancas***: informações de contato (nome, endereço, URLs, etc.).
- ***Páginas amarelas***: categorização dos serviços baseados em taxonomias.
- ***Páginas verdes***: informações técnicas sobre o serviço publicado, como especificações do serviço WEB e sua localização.

A especificação do UDDI consiste em um documento no formato XML *Schema* para as mensagens SOAP (*Simple Object Access Protocol*) e a descrição da especificação da *API (Aplication Program Interface)* do UDDI. Juntos eles formam um modelo básico que permite a publicação de informação sobre um grande número de *WEB Services*.

O reconhecimento de padrões que permitam a classificação de atributos não funcionais de serviços WEB, necessita de um método padronizado para expressar instruções dentro de uma arquitetura SOA, ou seja, um conjunto de regras sintáticas e semânticas que fundamentam uma "linguagem formal" de programação.

Entende-se por "linguagens formais", os mecanismos formais para representação e especificação de linguagens, baseados na chamada "Teoria da Computação". As representações podem ser feitas por reconhecedores e geradores. Os reconhecedores são dispositivos formais que servem para verificar se uma sentença pertence ou não à determinada linguagem. São os autômatos finitos, autômatos de pilha e Máquina de Turing. Os sistemas geradores são dispositivos formais que permitem a geração sistemática de todas as sentenças de uma linguagem. A tabela 1 a seguir, ilustra os reconhecedores utilizados em função de cada tipo de gramática segundo a classificação de Chomsky [20].

Teoria de Autômatos: Linguagem formal e gramática formal			
Hierarquia Chomsky	Gramática	Linguagem	Reconhecedor
Tipo-0	Estrutura de frase	Recursivamente enumerável	Máquina de Turing
--	Estrutura de frase	Recursiva	Máquina de Turing
Tipo-1	Sensíveis ao contexto	Sensíveis ao contexto	Máquina de Turing com memória limitada
Tipo-2	Livre de contexto	Livre de contexto	Autômato com pilha
Tipo-3	Regular	Regular	Autômato finito

Tabela 1 – Classificação de Gramáticas segundo a hierarquia de Chomsky.

A classificação das gramáticas começam pelo tipo 0, com maior nível de liberdade em suas regras, e aumentam as restrições até o tipo 3. Cada nível é um super-conjunto do próximo. Logo, uma gramática de tipo n é conseqüentemente uma gramática de tipo n - 1.

Uma gramática G pode ser formalmente definida [2] por uma quádrupla (V, Σ, R, S), onde:

- V é um alfabeto;
- Σ é um alfabeto de símbolos terminais e um subconjunto de V;
- R é um conjunto de regras pertencente ao subconjunto de (V- Σ) x V*;
- S é o símbolo inicial.

Para o reconhecimento de gramáticas não determínicas, requer-se a utilização de autômatos não-determinísticos, uma vez que a trajetória de reconhecimento de uma sentença nem sempre é única, exigindo que sua operação seja, muitas vezes, efetuada por meio de tentativas e erros ("*backtracking*") [18]. Isto acarreta uma substancial perda de eficiência do reconhecedor, tornando anti-econômico o seu uso em aplicações práticas.

Segundo Neto [19], autômatos adaptativos podem ser empregados para a contrução de reconhecedores de gramáticas livres ou dependentes de contexto, determinísticos ou não. O reconhecimento consiste em uma análise sintática (*parsing*), definido como o processo de analisar uma sequência de entrada (lida de um arquivo de computador ou do teclado, por exemplo) para determinar sua estrutura gramatical segundo uma determinada gramática formal. Essa análise faz parte de um compilador, junto com a análise léxica e análise semântica.

A análise sintática transforma um texto na entrada em uma estrutura de dados, em geral uma árvore, o que é conveniente para processamento posterior e captura a hierarquia implícita desta entrada. Através da análise léxica é obtido um grupo de tokens, para que o analisador sintático use um conjunto de regras para construir uma árvore sintática da estrutura, como ilustrado na figura 1.1.

Fig 1.1 – Exemplo de análise sintática (*parsing*)

Em termos práticos, pode também ser usada para decompor um texto em unidades estruturais para serem organizadas dentro de um bloco, por exemplo. A vasta maioria dos analisadores sintáticos implementados em compiladores aceitam alguma linguagem livres de contexto para fazer a análise. Estes analisadores podem ser de vários tipos, como o LL, LR e SLR [17].

II. UTILIZAÇÃO DE DISPOSITIVOS ADAPTATIVOS

O problema de indução incremental de árvores de decisão para atributos discretos pode ser resolvido através da aplicação da tecnologia adaptativa, utilizando um dispositivo adaptativo descrito no algoritmo *AdapTree* [3].

Um dispositivo adaptativo [4] é constituído em duas partes: a primeira é algum dispositivo usual, definido através de regras (em particular, algum autômato, gramática ou qualquer outro dispositivo descrito através de um conjunto finito de regras estáticas), denominado seu dispositivo subjacente (tipicamente não-adaptativo); a segunda é um mecanismo adaptativo, cuja conexão ao formalismo subjacente proporciona-lhe todos os recursos complementares necessários para a realização das propriedades responsáveis pela auto-modificação autônoma que caracteriza os dispositivos adaptativos.

Um autômato adaptativo [5] de estados finitos (AAF) é um dispositivo adaptativo, que estende o poder de expressão do autômato de estados finitos (AF), através da capacidade de modificar a sua própria estrutura com a aplicação de regras adaptativas citadas anteriormente. Formalmente, um AAF pode ser definido conforme Quadro 2.1:

$M = \langle Q, \Sigma, q_0, F, \delta, Q_\infty, \Gamma, \Pi \rangle$

onde:

$Q \subseteq Q_\infty$ é um subconjunto finito de um conjunto, possivelmente infinito, de estados.

Σ é o alfabeto de entrada, finito e não vazio.

$q_0 \in Q$ é o estado inicial do autômato.

$F \subseteq Q$ é o conjunto de Estados Finais.

$\delta \subseteq (Q_\infty \times (\Sigma \cup \{\epsilon\}) \times Q_\infty)$ é a relação de transição.

Os três últimos elementos de M, responsáveis pela adaptabilidade do sistema, são definidos a seguir:

Q_∞ é um conjunto, possivelmente infinito, de estados.

$\Gamma \subseteq (\{+,-\} \times (Q_\infty \times (\Sigma \cup \{\epsilon\}) \times Q_\infty))$ é o conjunto de ações primitivas que podem ser executadas pelo AAF: incluir (+) ou excluir (-) transições [4].

$\Pi : (Q_\infty \times (\Sigma \cup \{\epsilon\}) \times Q_\infty) \rightarrow \Gamma^*$ é uma função que associa a cada transição em δ uma seqüência de ações adaptativas primitivas. Podemos associar uma seqüência vazia a uma determinada transição para indicar que esta é uma transição "normal", não adaptativa.

Quadro 2.1 – Definição formal de um AAF

Neste AAF, para toda a transição:

$$x \in \delta \text{ onde } \Pi(x) = \epsilon \qquad \text{(eq. 2.1)}$$

O autômato adaptativo se reduz, funcionalmente, a um simples autômato de estados finitos. Nos outros casos, antes de executar a transição, o AAF deverá efetuar as alterações adaptativas sugeridas pelas ações adaptativas associadas a mesma. Após executar as ações adaptativas, o AAF passa a operar com sua nova estrutura.

O dispositivo adaptativo no qual o *AdapTree* se baseia, pode ser visto como um AAF classificador, estendido para trabalhar com mais de duas classes de serviços. Neste contexto, a cada estado final é associado um elemento, que corresponderá a uma das classes possíveis. Ao receber um exemplo de treinamento o *AdapTree* cria um caminho ligando o estado inicial do autômato ao estado final correspondente à classificação deste exemplo.

Formalmente, dado n conjuntos disjuntos:

$A_1, A_2, ..., A_n$

O conjunto de treinamento é composto por:

$T = \{w_1, w_2, ..., w_m\}$

Onde:

$|w_i| = n,\ w_i = \alpha_1\alpha_2...\alpha_n \text{ com } \alpha_j \in A_j$

Para:

$1 \leq j \leq n$

$1 \leq i \leq m$

O último símbolo da cadeia é utilizado para representar a calsse do exemplo, sendo que |An|, determina a quantidade de classes do problema e n-1 é o total de atributos (variáveis) a serem considerados no processo de aprendizagem. Os conjuntos:

$A_1, A_2, ..., A_n$

Representam os domínios – valores permitidos – de cada atributo.

Para alternar o classificador do modo de treinamento para o modo de classificação, basta fornecer cadeias de caracteres de tamanho (n-1), excluindo dessa forma as classificações.

Caso não seja possível determinar, sintaticamente, a classe de uma cadeia de entrada, o *AdapTree* utiliza o mecanismo estatístico **ID3** [6], que fornecerá como resposta uma

estimativa baseado no ganho de entropia de informação, definindo a ordem dos atributos utilizada nesta classificação.

Porém, segundo Quinlan [7], um atributo com muitos valores possíveis, teria uma dispersão maior na distribuição de probabilidades desses valores, conseqüentemente maior ganho de entropia de informação. Para evitar esta distorção, que favorece o atributo com maior número de valores, introduziu-se no algoritmo **C4.5** [8], sucessor do ID3, o conceito de Razão do Ganho de Entropia de Informação.

O método **C4.5** tem sido largamente empregado para construir árvores de decisão (*decision tree* ou DT) que implementam classificadores de elevada performance. Contudo, este algoritmo só permite classes previamente definidas para classificação de atributos na fase de treinamento, não tendo a capacidade de interpolar ou deduzir novos padrões por inferência ou tratar dados imprecisos ou incertos. Como mostrado em [9] e [10], o **C4.5** somente pode delimitar hiperplanos paralelos aos eixos coordenados, fato que, em um espaço de atributos contínuos, implicaria em crescimento exponencial da DT resultante.

Diferentemente do **C4.5**, o *AdapTree* não particiona o conjunto de exemplos, ou seja, o princípio de otimização é global e não local. As árvores de decisão geradas pelo *AdapTree* não possuem um número excessivo de ramificações, devido a esta característica.

Adicionalmente, como o *AdapTree* é um algoritmo incremental, onde novos exemplos podem ser incorporados durante o processo de classificação, resolvendo adequadamente o problema de variação temporal de padrões, muito comum em atividades de *Data Mining*. A combinação do *AdapTree* e do mecanismo C4.5, permite maximizar a vantagem de ambos algoritmos, minimizando as desvantagens citadas.

No entanto, ambos *AdapTree* e C4.5 necessitam de um conjunto de treinamento para geração de uma DT cujo número de nós entre a raiz e folhas não seja excessivamente grande. Dessa forma, o *AdapTree* e o C4.5 são algoritmos que trabalham em modo supervisionado durante a fase de treinamento, ou seja, não são capazes de definir o número de classes que particionam o conjunto de treinamento.

Por outro lado, *Redes Neurais Artificiais* (RNA) têm sido empregadas em tarefas de classificação para determinação de padrões em modo supervisionado ou não supervisionado.

Classes complexas podem ser prospectadas através dos dados, a fim de gerar uma DT de forma mais rápida, eficiente e de simples visualização.

Evidentemente que a performance e qualidade dos padrões adquiridos estão diretamente correlacionados com o tipo e a arquitetura escolhida para a modelagem da RNA, como demonstram os estudos realizados comparando-se a arquitetura *Multi-Layered Perceptron* (MLP) com DT [7].

A seguir, será apresentado um algoritmo de RNA adaptativo, porém não baseado no paradigma de Autômatos Adaptativos, cuja vantagem é não necessitar de um conjunto de treinamento classificado a priori, como o *AdapTree*.

III. Redes Neurais Artificiais

O algoritmo *Self-Organizing Map* (SOM), desenvolvido por Teuvo Kohonen [11], é um dos modelos mais populares de RNA. O algoritmo da SOM é baseado em um aprendizado competitivo e não supervisionado, o que implica em um treinamento direcionado exclusivamente pelos dados, sendo que os neurônios que constituem o mapa competem entre si para adquirir padrões dos dados, se aproximando deles. Algoritmos supervisionados, como o *Multi-Layered Perceptron* (MLP), requerem uma classificação pré-definida para cada vetor de treinamento, além de depender fundamentalmente do número de camadas internas (*hidden units*) para um aprendizado com baixo erro de classificação e boa performance, limitações que não ocorrem na arquitetura SOM.

No algoritmo interativo da SOM [15], cada neurônio i é representado por um vetor n-dimensional de pesos conhecido como vetor protótipo ou vetor de referência m_i:

$$m_i = [m_{i1},...,m_{in}]^T \qquad \text{(eq. 3.1)}$$

Onde:

- n é igual ao número de dimensões do vetor de entrada;
- m_{in} é o n-ésimo peso do vetor m_i.

No algoritmo básico, as relações topológicas e o número de neurônios são previamente fixados durante a fase de estruturação da rede. O número de neurônios deve ser usualmente escolhido como o maior possível, assim como o tamanho da vizinhança de forma a buscar um equilíbrio entre performance computacional e um bom grau de generalização e granularidade.

Antes da etapa de treinamento não supervisionado da SOM, valores são escolhidos para os pesos componentes dos vetores de referência.

Em cada passo do treinamento, um vetor de entrada de amostra x dos dados é escolhido aleatoriamente e uma medida de similaridade é calculada entre este e todos os vetores de referência do mapa da SOM. A unidade do mapa que mais se assemelha, também conhecida na literatura como BMU (Best Matching Unit), denominada aqui como unidade vencedora ***c***, é a unidade que possui a menor medida distância, tipicamente distância Euclidiana, segundo a equação 3.2:

$$\|x - m_c\| = \min_i \{\|x - m_i\|\} \qquad \text{(eq. 3.2)}$$

Onde $\|\,.\,\|$ denota a medida de distância Euclidiana.

A regra de atualização para o vetor de referência de uma unidade i da SOM pode ser expressa pela eq. 3.3:

$$m_i(t+1) = m_i(t) + h_{ci}(t)\left[x(t) - m_i(t)\right] \qquad \text{(eq. 3.3)}$$

Onde:

- t denota o instante de tempo ou passo do treinamento;
- x(t) o vetor de entrada selecionado aleatoriamente dos dados em um instante t;

- $h_{ci}(t)$ a relação de vizinhança ao redor de uma BMU ***c*** em um instante t.

Apesar do algoritmo de treinamento ser relativamente simples, o tratamento matemático para obtenção de demonstrações de convergência é bastante complexo. Segundo Vesanto[15], a convergência em um mapa unidimensional ocorre na maioria dos casos. Neste caso, para um número elevado de neurônios cujo raio R final de vizinhança, densidade pontual para os vetores referência da SOM é proporcional à eq. 3.4:

$$p(x)^{\frac{2}{3}-\frac{1}{3R^2+3(R+1)^2}} \qquad \text{(eq. 3.4)}$$

Onde p(x) é a função de densidade de probabilidade das entradas x.

Nesta expressão, quando a dimensão dos vetores de entrada aumenta o expoente acima tende a unidade, assim como a distribuição do vetor de pesos se aproxima aos dados de treinamento e o algoritmo de treinamento converge.

Entretanto, a presença de imprecisão, incerteza ou ruído nos dados pode reduzir drasticamente a performance treinamento não supervisionado do algoritmo SOM, da mesma forma que induz árvores excessivamente grandes utilizando os algoritmos *AdapTree* e C4.5. Para tratar de forma eficaz a incerteza, torna-se necessário a utilização de um mecanismo de inferência baseado em lógica nebulosa.

IV. LÓGICA NEBULOSA

A lógica nebulosa (*fuzzy*) possibilita que seja abordado, de forma mais adequada, o problema referente à representação e manipulação de imprecisão ou incerteza.

Os sistemas baseados em lógica nebulosa foram criados por Zadeh[12], fundamentando-se na representação e manipulação de informações incertas e imprecisas tão comuns no cotidiano humano. Expressões tais como "quase", "muito" e "pouco" representam este tipo de imprecisão, que usualmente não pode ser tratada pelos sistemas da lógica clássica de forma simples.

Os sistemas especialistas *fuzzy* utilizam um conjunto de regras do tipo "*If-Then*", baseadas em variáveis nebulosas. Primeiramente as variáveis de entrada sofrem um processo de "*fuzzificação"*, ou seja, os conjuntos nebulosos das variáveis lingüísticas de entrada são ativados. Terminado este processo, efetua-se a inferência sobre o conjunto de regras nebulosas obtendo os valores das variáveis de saída. Finalmente, as variáveis de saída sofrem um processo de *"defuzzificação*".

Este processo consiste em converter os dados nebulosos para valores numéricos precisos. Para isto são utilizadas várias técnicas, tais como valor máximo, média dos máximos, média local dos máximos, centro de gravidade, ponto central da área, entre outros. Neste artigo, foi utilizado o método de chamado *Takagi-Sugeno* [13], ou simplesmente *Sugeno*, cuja saída representará um valor constante, representando uma classe definida pela RNA do tipo SOM.

V. APLICAÇÃO DA ADAPTIVE FUZZY NEURAL TREE NETWORK

Existem inúmeras formas de se combinar árvores de decisão adaptativas [1], RNA do tipo SOM e Lógica *Fuzzy* para compor a *Adaptive Fuzzy Neural Tree Network*, porém a escolha dos algoritmos, neste artigo, objetiva suportar as atividades de descoberta dinâmica de serviços. O processo de descoberta dinâmica, nesse contexto, consiste basicamente em interceptar as requisões de serviços provenientes de um consumidor e intermediar a procura nos possíveis provedores de serviços, baseado na classificação de atributos não funcionais dos mesmos.

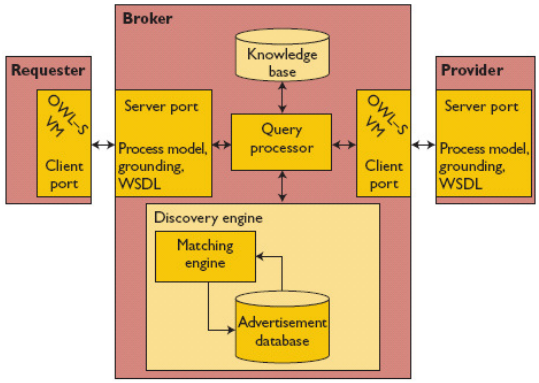


Fig. 5.1 . Intermediador de serviços WEB [14]

A figura 5.1 ilustra o modelo de um intermediador (*broker*), entre o consumidor (*requester*) e provedor (*provider*), utilizando características das linguagens OWL-S e WSDL.

Cabe a *AFNTN* realizar as tarefas de procura (*matching*), realizando uma busca sobre a base de dados (*advertising database*) que contém os atributos não funcionais dos serviços, conforme ilustrado na figura 5.2.

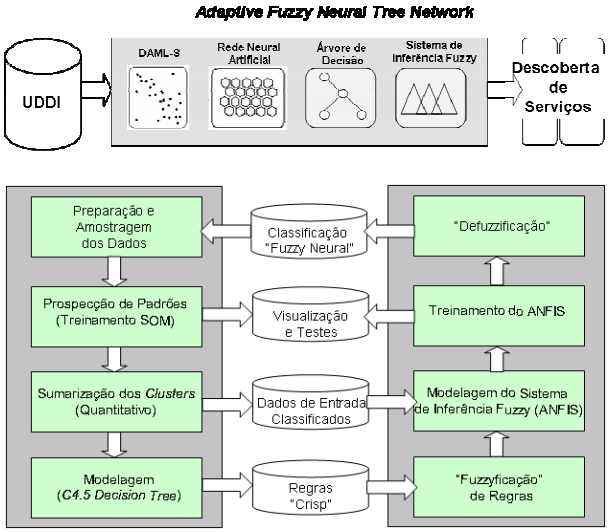


Fig. 5.2. Diagrama de Blocos da *Adaptive Fuzzy Neural Tree Network (AFNTN)*

A base de dados utilizada é composta de características de serviços WEB, geradas a partir dos *services profiles* utilizados pela linguagem OWL-S. A função do service profile é mapear

características de entidades e serviços de negócios em um único repositório. Nesta base, constam aproximadamente 500 *services profiles*, contendo os seguintes atributos não funcionais:

- **QoS (*Quality of Service*)** – índice de qualidade numérico que varia de 0 a 300 (valor adimensional).
- **Preço Médio** – valor cobrado pela utilização do serviço, recebe valores numéricos de 1 a 5000 (valor adimensional).
- **Volume de *Handover* Vertical** – número de vezes na qual um mesmo serviço foi disponibilizado por um provedor diferente, variando entre 1 e 1500 (valor adimensional).
- **Volume de *Handover* Horizontal** - número de vezes na qual diferentes serviços foram disponibilizados por um mesmo provedor, variando entre 1 e 5100 (valor adimensional).

Aplicando-se estes atributos e seus respectivos valores para os services profiles utilizados para classificação da RNA do tipo SOM, resultaram 4 classes que mapeiam os 500 *services profiles*.

Todas as funções utilizadas no treinamento do algoritmo da SOM e os resultados obtidos, basearam-se em funções previamente elaboradas em Matlab® versão 6.5 provenientes da SOMTOOLBOX 2.0 [15].

Estas classes foram utilizadas como entrada da árvore de decisão adaptativa, resultando na seguinte ontologia de classes de serviços, conforme ilustrado na figura 5.3.

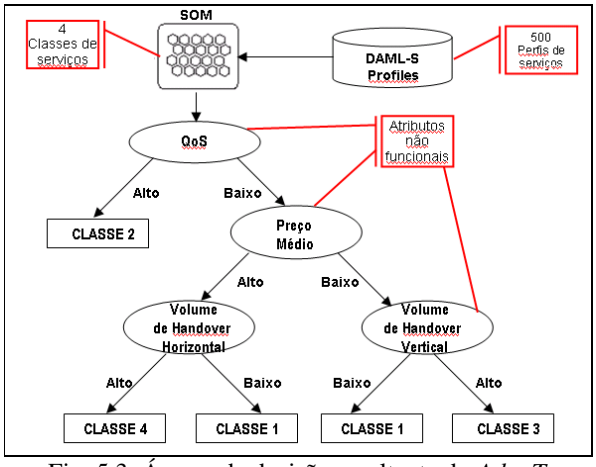


Fig. 5.3. Árvore de decisão resultante do *AdapTree* modificado

É possível inferir as seguintes regras no quadro 5.1, através da árvore de decisão ilustrada na fig. 5.3:

Regra 1. SE QoS:= "Alto" ENTÃO
Serviço PERTENCE Classe2
Regra 2. SE QoS:= "Baixo" E
Preço Médio:= "Baixo" E
Volume de Handover Vertical := "Alto" ENTÃO
Serviço PERTENCE Classe3
Regra 3. SE QoS:= "Baixo" E
Preço Médio:= "Alto" E
Volume de Handover Horizontal := "Baixo" ENTÃO
Serviço PERTENCE Classe3
Regra 4. SE QoS:= "Baixo" E
Preço Médio:= "Alto" E
Volume de Handover Horizontal := "Alto" ENTÃO
Serviço PERTENCE Classe4

Quadro 5.1 Regras inferidas pela árvore de decisão, gerada a partir do *AdapTree*

Os termos *fuzzy* "Alto" ou "Baixo" foram utilizados para "*fuzzificar*" as regras rígidas (*crisp*) geradas pela árvore de decisão.

A definição das funções de pertinência dos termos *fuzzy* será de responsabilidade do sistema adaptativo de inferência *fuzzy* (ANFIS), do tipo *Sugeno* ilustrado na fig. 5.4.

Os dados classificados e as regras *fuzzy* foram utilizadas como entrada do sistema de inferência *fuzzy* do tipo *Sugeno*, fornecido pela FUZZYTOOLBOX [16] do Matlab®.

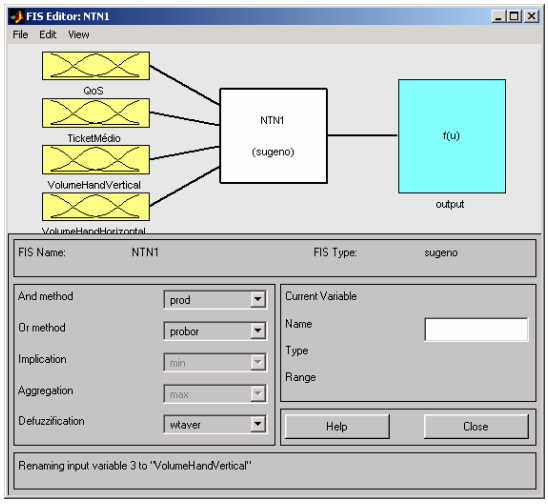


Fig. 5.4. Sistema de inferência *Fuzzy* Adaptativo

Após o treinamento supervisionado do ANFIS do tipo *Sugeno*, obteve-se o ajuste das funções de pertinência de entrada e saída, conforme Figura 5.5:

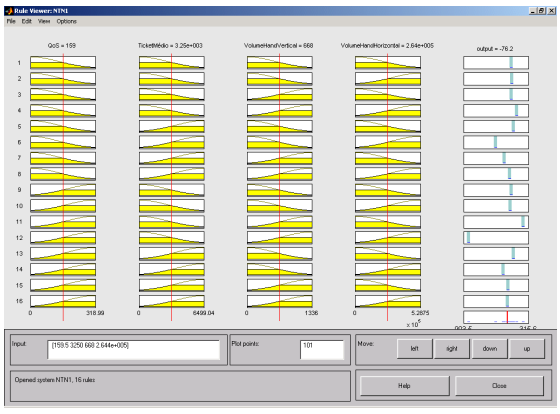


Fig. 5.5. Resultados do treinamento do ANFIS

O sistema de Inferência *Fuzzy* Adaptativo do tipo *Sugeno* (ANFIS) utiliza uma rede neural do tipo *Multi-Layer Perceptron* (MLP) [16] para ajuste das funções de pertinência.

A combinação das variáveis de entrada com as possíveis classes de saída é realizada através da criação de regras *fuzzy* do tipo "*E*", conforme quadro 5.2 a seguir:

```
SE Variável_Entrada_1 PERTENCE A Função_Pertinência_1
  E
    Variável_Entrada_2 PERTENCE A Função_Pertinência_1
  E ...
    Variável_Entrada_n PERTENCE A Função_Pertinência_1
ENTÃO
    Saída PERTENCE A Funcão_Pertinência_Classe_1
```

Quadro 5.2 Fragmento do arquivo de configuração léxica para o FLEX

As funções de pertinência delimitam superfícies de decisão, como ilustrado na Figura 5.6, onde as variáveis de entrada *Devoluções* e *Preço Médio* delimitam uma superfície tridimensional que, por sua vez, delimita a função de pertinência da classificação de saída (*output*), determinada pela RNA do tipo SOM.

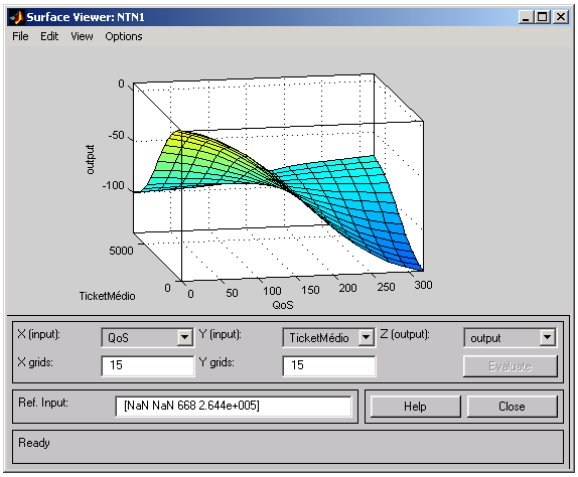


Fig. 5.6. Superfície de decisão formada pelas variáveis *Ticket Médio* e *QoS*

VI. Resultados da Análise Sintática e Léxica da Gramática Gerada pela *AFNTN*

GNU BISON é um *software livre* gerador de parser, escrito para o projeto GNU, e disponível para quase todos os sistemas operacionais. É compatível com o YACC, e oferece muitas melhorias se comparado com este *software*. Ele é utilizado em conjunto com o analisador léxico FLEX (FLEX lexical analyser).

Com a utilização dos programas BISON e FLEX foi implementado um analisador léxico e sintático para documentos que descrevem as regras *fuzzy* do tipo "E".

Como primeiro passo, foi configurado um arquivo com a descrição léxica da linguagem e que foi utilizado como entrada para o analisador léxico FLEX. O quadro 6.1 representa o segmento principal do arquivo de configuração utilizado pelo FLEX para definição dos *tokens* da linguagem.

```
se                      return RESERVED_WORD_IF;
e                       return RESERVED_WORD_AND;
pertence                return RESERVED_WORD_BELONG;
entao                   return RESERVED_WORD_SO;
:=                      return ASSIGNMENT_OPERATOR;
;                       return SEMICOLON;
[0-9]+                  return NUMBER_VAL;
[a-zA-Z]+[a-zA-Z0-9]*   return IDENTIFIER_OR_ALPHANUM_VAL;
\n                      /* ignore end of line */;
[ \t]+                  /* ignore whitespace */;
```

Quadro 6.1 Fragmento do arquivo de configuração léxica para o FLEX

Para a implementação do analisador sintático foi elaborado um arquivo de regras de sintaxe, segundo a formatação definida pelo BISON. Por esta configuração foram estabelecidas as regras sintáticas para a elaboração de programas contemplando as regras *fuzzy* do tipo "E". O quadro 6.2 representa o segmento principal do arquivo de configuração utilizado pelo BISON para a geração do analisador sintático.

```
commands:
        adaptative_service_discovery SEMICOLON
        |commands adaptative_service_discovery SEMICOLON
        ;
adaptative_service_discovery:
        attribute_pertinency_statements RESERVED_WORD_SO service_pertinency_statement
        ;
attribute_pertinency_statements:
        first_attribute_pertinency_statement
        |
        attribute_pertinency_statements RESERVED_WORD_AND attribute_pertinency_statement
        ;
first_attribute_pertinency_statement:
        RESERVED_WORD_IF attribute_pertinency_statement
        ;
attribute_pertinency_statement:
        IDENTIFIER_OR_ALPHANUM_VAL ASSIGNMENT_OPERATOR attribute_value
        RESERVED_WORD_BELONG IDENTIFIER_OR_ALPHANUM_VAL
        ;
attribute_value:
        NUMBER_VAL
        |
        IDENTIFIER_OR_ALPHANUM_VAL
        ;
service_pertinency_statement:
        IDENTIFIER_OR_ALPHANUM_VAL RESERVED_WORD_BELONG IDENTIFIER_OR_ALPHANUM_VAL
        ;
```

Quadro 6.2. Fragmento do arquivo de configuração sintática da linguagem utilizado pelo gerador BISON

Adicionalmente, foi utilizada uma extensão ao BISON, denominada YAXX (YAcc eXtension to XML). Com a ferramenta a YAXX foi possível a implementação de um analisador léxico e sintático que produz como saída uma representação XML do arquivo de regras utilizado como parâmetro de entrada. O quadro 6.3 apresenta um fragmento do arquivo XML gerado pelo YAXX.

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="yaxx.xsl"?><!DOCTYPE commands SYSTEM "yyyaxx.dtd">
<yaxx:commands xmlns:yaxx="urn:YAcc-Xml-eXtension">
<yaxx:commands>
<yaxx:adaptative_service_discovery>
<yaxx:attribute_pertinency_statements>
<yaxx:attribute_pertinency_statements>
<yaxx:attribute_pertinency_statements>
<yaxx:first_attribute_pertinency_statement>
<yaxx:RESERVED_WORD_IF>se</yaxx:RESERVED_WORD_IF>
<yaxx:attribute_pertinency_statement>
<yaxx:IDENTIFIER_OR_ALPHANUM_VAL>QoS</yaxx:IDENTIFIER_OR_ALPHANUM_VAL>
<yaxx:ASSIGNMENT_OPERATOR>:=</yaxx:ASSIGNMENT_OPERATOR>
<yaxx:attribute_value>
<yaxx:NUMBER_VAL>150</yaxx:NUMBER_VAL>
</yaxx:attribute_value>
<yaxx:RESERVED_WORD_BELONG>pertence</yaxx:RESERVED_WORD_BELONG>
<yaxx:IDENTIFIER_OR_ALPHANUM_VAL>Funcao_Pertinencia_1</yaxx:IDENTIFIER_OR_ALPHANUM_VAL>
</yaxx:attribute_pertinency_statement>
</yaxx:first_attribute_pertinency_statement>
</yaxx:attribute_pertinency_statements>
<yaxx:RESERVED_WORD_AND>e</yaxx:RESERVED_WORD_AND>
<yaxx:attribute_pertinency_statement>
<yaxx:IDENTIFIER_OR_ALPHANUM_VAL>PrecoMedio</yaxx:IDENTIFIER_OR_ALPHANUM_VAL>
<yaxx:ASSIGNMENT_OPERATOR>:=</yaxx:ASSIGNMENT_OPERATOR>
<yaxx:attribute_value>
<yaxx:NUMBER_VAL>100</yaxx:NUMBER_VAL>
</yaxx:attribute_value>
<yaxx:RESERVED_WORD_BELONG>pertence</yaxx:RESERVED_WORD_BELONG>
<yaxx:IDENTIFIER_OR_ALPHANUM_VAL>Funcao_Pertinencia_1</yaxx:IDENTIFIER_OR_ALPHANUM_VAL>
</yaxx:attribute_pertinency_statement>
</yaxx:attribute_pertinency_statements>
<yaxx:RESERVED_WORD_AND>e</yaxx:RESERVED_WORD_AND>
<yaxx:attribute_pertinency_statement>
<yaxx:IDENTIFIER_OR_ALPHANUM_VAL>VolHandoverVertical</yaxx:IDENTIFIER_OR_ALPHANUM_VAL>
<yaxx:ASSIGNMENT_OPERATOR>:=</yaxx:ASSIGNMENT_OPERATOR>
<yaxx:attribute_value>
<yaxx:NUMBER_VAL>200</yaxx:NUMBER_VAL>
</yaxx:attribute_value>
<yaxx:RESERVED_WORD_BELONG>pertence</yaxx:RESERVED_WORD_BELONG>
<yaxx:IDENTIFIER_OR_ALPHANUM_VAL>Funcao_Pertinencia_1</yaxx:IDENTIFIER_OR_ALPHANUM_VAL>
</yaxx:attribute_pertinency_statement>
</yaxx:attribute_pertinency_statements>
<yaxx:RESERVED_WORD_SO>entao</yaxx:RESERVED_WORD_SO>
<yaxx:service_pertinency_statement>
<yaxx:IDENTIFIER_OR_ALPHANUM_VAL>Servico_S</yaxx:IDENTIFIER_OR_ALPHANUM_VAL>
<yaxx:RESERVED_WORD_BELONG>pertence</yaxx:RESERVED_WORD_BELONG>
<yaxx:IDENTIFIER_OR_ALPHANUM_VAL>Classe_1</yaxx:IDENTIFIER_OR_ALPHANUM_VAL>
```

Quadro 6.3. Fragmento do arquivo XML produzido pelo analisador léxico e sintático gerado via ferramentas FLEX, BISON e YAXX

VII. CONCLUSÃO

A linguagem denominada de *Adaptive Dynamic Discovery Language*, gerada a partir da combinação de regras do tipo *"E"*, tem como princípio construtivo a utilização da recursividade, como ilustrado na figura 7.1, a seguir:

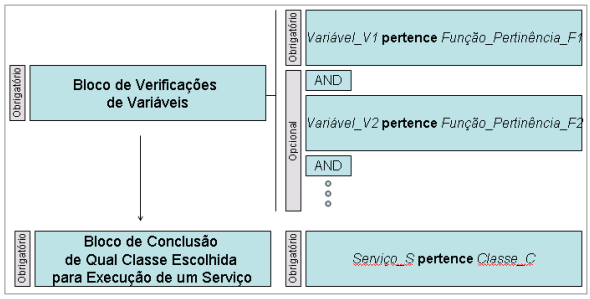


Fig 7.1 Estrutura da linguagem XML adaptativa

A recursão, nesta linguagem, foi necessária para definir qual a classe pertence um determinado serviço baseado nos parâmetros de entrada avaliados pela *AFNTN*, realizando as seguintes tarefas:

- Definição dos agrupamentos (*clusters*) de serviços através da utilização da Rede Neural Artificial do tipo SOM.
- Estruturação dos atributos não funcionais em um formato de árvore através da utilização da Árvore de Decisão Adaptativa (*AdapTree* modificado).
- Eliminação de incertezas nos dados através da utilização de um sistema de inferência *fuzzy* do tipo *Sugeno* (*ANFIS*), com a geração das regras de produção da linguagem proposta.

As regras de produção, geradas a partir do *ANFIS*, foram validadas de forma léxica e sintática gerando uma arquivo XML, conforme figura 7.2 a seguir

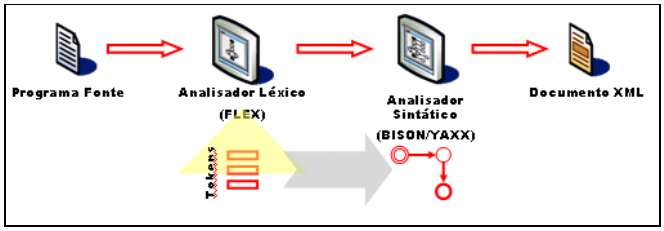


Fig. 7.2. Seqüência de passos para a validação da regras de produção da linguagem

Tal representação é bastante adequada para a implementação de serviços de busca (*discovery*) em um repositório que considere também atributos não funcionais, tornando-se um diferencial aos mecanismos atuais baseados em UDDI e WSDL, e permitirá a composição de serviços WEB através de linguagens como WSOL e OWL-S.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] PATEL, K. Improvements on WSOL Grammar and Premier WSOL Parser. Research Report. SCE-03-25 October 2003

[2] PAPADIMITRIOU, C. H.; LEWIS, H. R.; Elementos da teoria da computação. Bookmann, 2a. edição, 2004.

[3] PISTORI, H. e 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 2002. Montevideo, Uruguai, novembro, 2002

[4] PISTORI, H.; NETO, J. J.; PEREIRA, M.C.;Tecnologia Adaptativa em Engenharia da Computação. Estado da Arte e aplicações. Edição Revisada, 174p. São Paulo, 2003.

[5] PISTORI, H.; NETO, J. J.; PEREIRA, M. C. Adaptive Non-Deterministic Decision Trees: General Formulation and Case Study. INFOCOMP Journal of Computer Science, Lavras, MG, 2006 (accepted).

[6] CHENG, J.; FAYYAD, U. M.; IRANI, K. B.; QIAN, Z.; Improved decision trees: A generalized version of ID3. Proceedings of the Fifth International Conference on Machine Learning (pp. 100-106). Ann Arbor, MI: Morgan Kaufman, 1988.

[7] QUINLAN, J. R.; Comparing Connectionist and Symbolic Learning Methods. Basser Department of Computer Science; University of Sydney; Sydney NSW 2006;Australia. 1990.

[8] QUINLAN, J. R. C4.5 Programs for Machine Learning. Morgan Kaufmann. 1992.

[9] BRODLEY, C. E.; UTGOFF, P. E. Multivariate versus univariate decision trees. Technical report, Department of Computer Sciences University of Massachussetts. 1992.

[10] MITCHELL, T. Machine Learning. McGraw Hill. 1997.

[11] KOHONEN, T. Self-Organizing Maps. Springer-Verlag. 1995.

[12] ZADEH, L.A., "Fuzzy sets," Information and Control, Vol. 8, pp. 338-353, 1965.

[12] SUGENO, M., Industrial applications of fuzzy control, Elsevier Science Pub. Co., 1985.

[14] HUNHS; M. N.; Dynamic Discovery and Coordination of Agent Based Semantic WEB Services. IEEE internet computing . pp. 66-73, May-June, 2004.

[15] VESANTO, J.; ALHONIEMI, E.; HIMBERG, J.; PARHANKANGAS, J. Som Toolbox 2.0 BETA online documentation. Internet address http:// www.cis.hut.fi/projects/ somtoolbox. 1999.
[16] JANG, J.-S. R., ANFIS: Adaptive-Network-based Fuzzy Inference Systems, IEEE Transactions on Systems, Man, and Cybernetics, Vol. 23, No. 3, pp. 665-685, May 1993.
[17] MITCHELL P. M. Theory of Syntactic Recognition for Natural Languages. Cambridge: MIT Press, 1980. 335 p. ISBN 0262131498
[18] GILLES B., BRATLEY P.(1995). Fundamentals of Algorithmics. Prentice-Hall.
[19] NETO, J. J. Contribuições à metodologia de construções de compiladores. Tese de livre docência. EPUSP, 2003.
[20] CHOMSKY, N. "On certain formal properties of grammars". Information and Control (2): 137-167. 1959
[21] DUTRA, R. G.; CABRAL, E.; Aplicação de métodos de inteligência artificial em inteligência de negócios. Dissertação de Mestrado, Poli-USP, 2001.
[22] DUTRA, R. G.; MARTUCCI, M.; Adaptive Fuzzy Neural Tree Network. IEEE TLA Volume: 6, Issue: 5. 2008

Dutra, R., Engenharia Eletrônica pelo Instituto Tecnológico de Aeronáutica (1995), mestrado em Engenharia Elétrica pela Universidade de São Paulo (2001) e atualmente cursando doutorado na Politécnica da Universidade de São Paulo. Atualmente é gerente de projetos SAP, com ênfase em projetos de implementação de soluções de sistemas integrados e Inteligência de Negócios. Atuando principalmente nos seguintes temas: Sistemas de Automação, Sistemas distribuídos, Sistemas Abertos, Arquitetura Orientada a Serviços, Arquitetura Distribuída.

Silva, R., Bacharelado em Ciências da Computação pela Universidade de São Paulo (1998), pós-graduação lato senso em Administração de Empresas pela Fundação Getúlio Vargas (2005) e atualmente cursando mestrado na Politécnica da Universidade de São Paulo. Atualmente é gerente de projetos de consultoria em TI, com ênfase em projetos de implementação de soluções de sistemas na indústria de bancos. Atuando principalmente nos seguintes temas: Sistemas em Plataforma Web, Desenvolvimento Orientado a Objetos, Soluções com Tecnologia Java.

Martucci Jr., M., Engenharia Elétrica pela Universidade de São Paulo (1973), graduação em Bacharedado Em Física pela Universidade de São Paulo (1975), mestrado em Engenharia Elétrica pela Universidade de São Paulo (1977) e doutorado em Engenharia Elétrica pela Universidade de São Paulo (1982) . Atualmente é professor titular da Universidade de São Paulo. Tem experiência na área de Engenharia Elétrica , com ênfase em Eletrônica Industrial, Sistemas e Controles Eletrônicos. Atuando principalmente nos seguintes temas: Sistemas de Automação, Sistemas distribuídos, Sistemas Abertos, Arquitetura Hierarquizada, Arquitetura Distribuída.

Vicente Ruggiero, W., Diretor Presidente da Scopus Tecnologia Ltda, Engenheiro Eletricista pela Escola Politécnica da Universidade de São Paulo em 1971, Mestre em Engenharia Elétrica pela EPUSP-1975, PhD em Ciência da Computação pela UCLA-1978, Prof. Titular do Departamento de Computação e Sistemas Digitais da Escola Politécnica da USP, Diretor Técnico do Laboratório de Arquitetura e Redes de Computadores (LARC) da Escola Politécnica da USP e Investigador Principal do Projeto TIDIA – Aprendizado Eletrônico. Na Scopus é responsável pelo desenvolvimento de sistemas transacionais seguros na Internet e coordena o processo de inovação tecnológica. Autor de mais de 100 artigos técnicos publicados em revistas e anais de congressos nacionais e internacionais.

Armazenador Adaptativo de Palavras e Frases da Língua Portuguesa

(19 Dezembro 2008)

M. Marques

***Resumo*— Neste trabalho apresentaremos uma proposta de aplicativo adaptativo para tratamento e armazenamento de palavras ou frases da língua portuguesa, demonstrado através de teste a um conjunto limitado de frases. O aplicativo proposto é baseado nos conceitos de ambiente colaborativo de aprendizado por descoberta (Collaborative Discovery Learning Environment - CDLE) e no framework de dispositivos adaptativos dirigidos por regras (DADR). O CDLE permite a construção de um conhecimento próprio, por meio da execução de experimentos dentro de um domínio e por meio da inferência e adição de novas regras ou informações gramaticais. Também é proposto um novo paradigma de programação nomeado como programação orientada a conceito (POC).**

***Palavras chave*— Ambiente de Descobrimento e Aprendizado Colaborativo, Dispositivos Adaptativos Dirigidos por Regras, Processamento de Linguagem Natural, Tabela de Decisão Adaptativa.**

I. Introdução

Este documento descreve um aplicativo adaptativo para processamento de linguagem natural, mais especificamente para o português do Brasil. A base do aplicativo é o conceito de CDLE, como apresentado em [4], por meio do qual o aplicativo aprende novas construções gramaticais ou novas palavras e as incorpora em seu ambiente de conhecimento, por meio da produção de novas regras de reconhecimento de padrões. Para permitir o processo de aprendizagem, o aplicativo efetua a comparação das frases que ele recebe em seu ambiente de interação com o usuário, com as regras previamente definidas no aplicativo adaptativo. As regras são baseadas no *framework* DADR, que foi introduzido primeiramente em [4] e compreende um formalismo não adaptativo básico e um formalismo adaptativo que altera o funcionamento do autômato adaptativo, conforme definido por Neto como M = (ND, AM), em que ND é o mecanismo não adaptativo dirigido por regras pré-definidas e AM é o mecanismo adaptativo. Caso as regras pré-definidas ou incorporadas dinamicamente no aplicativo não consigam tratar a nova frase recebida como uma frase padrão do conhecimento já acumulado, o aplicativo fará a opção por inferir uma nova regra.

O restante do artigo é organizado como segue. A seção 2 descreve as principais características do aplicativo adaptativo para processamento de linguagem natural. A seção 3 apresenta a tabela de decisão adaptativa que é gerada para reconhecimento das frases utilizadas nos testes da seção 2. A seção 4 apresenta o diagrama padrão de funcionamento da aplicação adaptativa para processamento de linguagem natural. A seção 5 apresenta a conclusão do artigo e perspectivas de trabalhos futuros que podem ser feitos.

M. Marques exerce a docência no Departamento de Computação da Faculdade Sumaré e na Escola Técnica de São Paulo, que pertence ao Centro Paula Souza e pode ser contactado no email mario.marques@sumare.edu.br.

II. Características principais do aplicativo adaptativo

O aplicativo que foi desenvolvido tem a finalidade de validar o conceito de que é possível criar um mecanismo de reconhecimento de frases e dessa forma poderia ser classificado na categoria de armazenador de palavras, que precede ao desenvolvimento do módulo de reconhecimento gramatical, conforme definido em [3]. Os reconhecedores gramaticais treináveis, conforme em [2] são compostos de três módulos, mas precedidos de uma fase inicial de treinamento do autômato, em que ele é exposto a um corpus anotado para treinamento. Após a etapa de treinamento, o primeiro módulo efetua a atividade de etiquetar as palavras conhecidas, a partir do aprendizado realizado no corpus anotado. O segundo módulo efetua a marcação das palavras desconhecidas e o terceiro módulo faz o refinamento contextual. A diferença que existe entre o reconhecedor gramatical proposto por Brill, que efetua a classificação por meio de etiquetas de cada frase reconhecida e a proposta aqui apresentada, é a substituição do módulo de treinamento em corpus anotado, por um módulo de armazenamento de palavras, que serão obtidas por meio de interação com o usuário do ambiente CDLE. No caso em que a palavra já seja conhecida não será necessário acionar o módulo de marcação morfológica, mas se a palavra for desconhecida será acionado o primeiro módulo, que contém regras pré-definidas para efetuar a marcação morfológica da palavra. Caso não haja uma referência direta entre a nova palavra e as regras existentes no módulo 1, a palavra será encaminhada para o módulo 2 que por meio de consulta ao usuário efetuará a marcação da palavra e a inserção da palavra na lista de palavras conhecidas. Nos casos em que a palavra passe pelo módulo 2, a mesma ainda será submetida ao módulo 3 para verificação contextual.

O aplicativo foi construído apenas com uma estrutura sintática, a qual permite o reconhecimento de uma única estrutura de frase simples do português do Brasil e que corresponde a porção de regras fixas ou ND do autômato.. A frase que será reconhecida deve ter apenas três componentes: um sujeito como elemento inicial, um verbo como segundo elemento e um complemento como terceiro elemento, conforme definido por:

Frase → {sujeito verbo complemento}

A gramática implementada para reconhecimento de frases, prevê sempre a ocorrência de apenas um único elemento da gramática: sujeito (s), verbo (v) e complemento (c) e está definida por:

$L = \{\mathbf{F} \in \{s,v,c\} \mid \mathbf{F} = s,v,c,\ n=1\}$

O autômato que foi especificado para reconhecimento da gramática, prevê a existência de quatro símbolos possíveis:

- S que representa a transição para estado vazio;
- s que representa o elemento sujeito da frase;
- v que representa o verbo da frase;
- c que representa o complemento da frase.

Uma frase para ser considerada completa deve possuir os três elementos (s,v,c), separados por transições vazias da seguinte forma:

$G = (\{S, s,v,c\}, \{s,v,c,\}, \{S \rightarrow \epsilon, S \rightarrow sSvSc\}, S)$

Para realizar os testes com o aplicativo adaptativo foram utilizadas as seguintes frases:

Pássaro é azul.

Pássaro é verde.

O pássaro é bonito.

A partir da inserção da frase "Pássaro é azul", o autômato constrói a estrutura de reconhecimento especificada na figura 1.

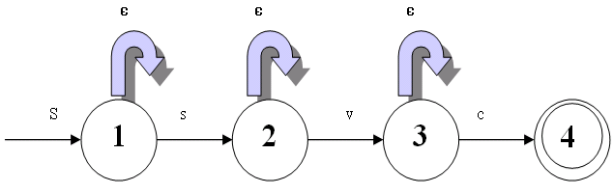


Fig. 1. O autômato em ação.

A partir da inserção da frase "Pássaro é verde" o autômato utiliza-se da mesma estrutura de reconhecimento especificada na figura 1.

A partir da próxima frase que é "O pássaro é bonito" o autômato terá dificuldades para reconhecer esta frase, pois ela contém quatro elementos e por este motivo solicitará o auxílio do usuário. A idéia básica é tentar reconhecer se as palavras da frase correspondem a palavras pré-cadastradas em frases anteriores e já catalogadas em categorias gramaticais componentes da estrutura da frase padrão. Dessa forma, ao analisar a cadeia de entrada, o autômato, por comparação, verifica que ele já conhece a palavra pássaro, que ele já catalogou como um substantivo e sujeito da frase, a palavra é que foi catalogada como um verbo, e o autômato não reconhece as palavras o e bonito. Com as informações que já são de conhecimento do autômato ele pergunta ao usuário qual é a categoria gramatical da palavra "o".

Como a resposta será a categoria de adjunto adnominal e artigo, o autômato então gera uma nova regra de reconhecimento, que corresponde à porção AM do autômato, em que é inserido o elemento a para representar a existência de um novo elemento, anterior ao substantivo, e que pode ser preenchido ou pode também aparecer vazio, conforme definido por:

$L = \{F \in \{a,s,v,c\} \mid F = s,v,c,\ a=*\ e\ s,v,c=1\}$

Uma nova gramática também é gerada, com a inclusão da possibilidade da existência de um elemento anterior. Dessa forma o autômato pode reconhecer frases que sejam formadas por três (s,v,c) ou quatro elementos (a,s,v,c), separados por transições vazias, conforme:

$G = (\{S, a, s, v, c\}, \{a, s, v, c,\}, \{S \rightarrow \epsilon, S \rightarrow aSsSvSc\}, S)$

A nova máquina de reconhecimento é ilustrada na figura 2, em que o elemento a é opcional na verificação da frase.

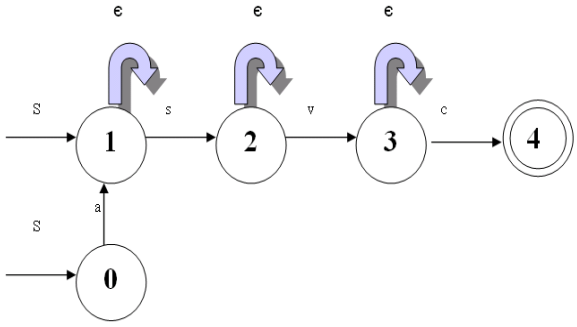


Fig. 2. O novo autômato.

Após a realização da ação adaptativa as novas palavras aprendidas devem ser armazenadas no léxico construído para posterior utilização. Specia e Rino [6] relatam à existência de duas abordagens mais usualmente utilizadas no armazenamento de itens lexicais, do ponto de vista da tecnologia da informação: base de dados ou arquivos seqüenciais. Do ponto de vista lingüístico as opções apontadas em [6] correspondem às citadas formas canônicas e as formas originais. No modelo em que a forma canônica é armazenada é definido um processo em que a partir da forma canônica são geradas as formas analisadas, com as devidas flexões de gênero, número, espécie, grau, modo, tempo, etc. Já no modelo de armazenamento das formas originais todas as variações de cada palavra são armazenadas. Neste trabalho, do ponto de vista lingüístico, utilizamos o armazenamento das formas canônicas das palavras e do ponto de vista computacional propomos um novo método, baseado no conceito de programação orientada a conceito (POC).

A POC é um novo paradigma para desenvolvimento de programas computacionais, pois propõe a identificação de cada palavra da língua portuguesa como um elemento componente da linguagem de programação POC. Em seu modelo conceitual, a POC é bastante similar à especificação de programação orientada a objetos (POO). As estruturas básicas da POC são

- Estrutura;
- Conceito.

A estrutura é a forma de implementação das estruturas de frase aprendidas pelo autômato adaptativo e, para cada nova estrutura de frase aprendida, uma nova estrutura é criada automaticamente pelo aplicativo adaptativo e carregada, automaticamente, como parte integrante do conjunto de frases conhecidas.

O conceito corresponde à especificação dos diferentes conceitos que podem ser aprendidos pelo aplicativo adaptativo. Cada conceito é composto de 4 elementos:

- Nome do conceito;
- Métodos;
- Atributos;
- Definição.

Cada conceito deve possuir um nome, por exemplo, o conceito de homem terá o nome homem. Além disso, cada conceito em POC possui métodos e atributos, que guardam estreita similaridade com os seus correspondentes em POO. Já a definição do conceito identifica de forma única ou coletiva cada conceito e também cada conceito deve pertencer a um conjunto. Por definição, a POC possui dois grandes conjuntos : o conjunto dos seres conscientes (C) e o conjunto dos seres não-conscientes (NC).

A categorização dos elementos da frase (conceitos) com o uso de teoria dos conjuntos, tem como base a definição de uma função, que permita localizar a qual conjunto a palavra da frase pertence. Por exemplo, a frase "O pássaro tem penas", deve ser definida por uma função f:A→B, biunívoca e definida sobre B. Os elementos da frase são inseridos no conjunto A, ou seja, A= {o, pássaro, tem, penas} e o conjunto B como B= {C, NC}, de modo que a função seja f(x) =x, sendo que se o x resultante for maior do que 1, x>1, ele corresponde ao conjunto NC, subconjunto não-animado (NA), caso x<=1, temos duas possibilidades, sendo que podemos classificá-lo como C ou NC e animado (A). Dessa forma, todas as palavras aprendidas pelo autômato serão classificadas e armazenadas, permitindo o tratamento de ambigüidade lingüística.

III. A TABELA DE DECISÃO ADAPTATIVA

A tabela de decisão adaptativa (TDA), conforme definida em [5] é o mecanismo utilizado para implementar o controle das ações que serão executadas por um dispositivo computacional que é composto por duas partes: um conjunto de regras pré-definidas e um conjunto de ações adaptativas, as quais podem ser representadas pela fórmula TDA = (ND, AM). Como já descrito anteriormente, a porção ND correspondem às regras não-adaptativas e a porção AM correspondem às ações adaptativas.

Na figura 3 temos o modelo conceitual da tabela de decisão adaptativa, conforme definido por Neto. Pode-se verificar que a tabela é composta das regras pré-definidas na parte superior da tabela relacionadas como tabela de decisão subjacente e as ações adaptativas na parte inferior da tabela relacionadas como funções adaptativas.

Cada coluna deve receber uma letra que indica qual tipo de ação deverá ser executada naquela coluna e na figura 3, extraída de [8], esta informação está identificada com o nome Tag. Os tipos de Tag possíveis são H ? + - S R E.

O Tag H indica o cabeçalho da função adaptativa e pode conter os seguintes caracteres:

A, para indicar se a regra deve ser executada antes da função adaptativa subjacente;

B, para indicar se a regra deve ser executada depois da função adaptativa subjacente;

P, para indicar um parâmetro formal;

V, para indicar uma variável;

G, para indicar um gerado.

Os caracteres ? + - indicam respectivamente as funções de consulta, adição e exclusão de uma ação adaptativa.

As colunas com os caracteres S R E indica respectivamente a primeira ação que deverá ser executada, a execução de uma regra normal não-adaptativa e o final da execução do dispositivo.

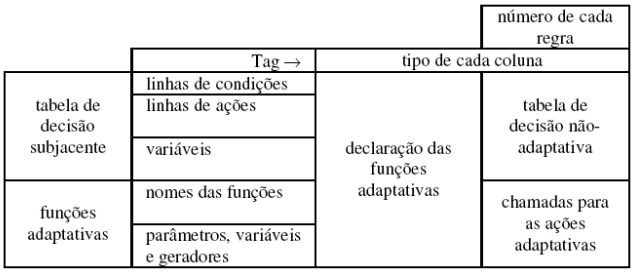


Fig. 3. Modelo conceitual da tabela de decisão adaptativa.

IV. DESCRIÇÃO DO APLICATIVO

O aplicativo que foi desenvolvido para validar a aplicabilidade do modelo aqui proposto foi construído na linguagem de programação Java, portanto baseado no paradigma de programação orientada a objeto, mas com algumas adaptações, devido à inexistência de uma linguagem de programação POC.

O aplicativo é composto de uma interface gráfica, que permite a interação do usuário com o módulo de interpretação de linguagem natural, para viabilizar, ao usuário, a digitação de frases e a resposta do usuário a questões colocadas pelo aplicativo.

O aplicativo desenvolvido neste artigo é o primeiro módulo da proposta de construção de um aplicativo de aprendizado e interação em linguagem natural, que deverá ser composto dos seguintes módulos: (conforme ilustrados na figura 4)

- Interface com o usuário - é o ambiente computacional em que o usuário pode interagir com o armazenador adaptativo, por meio da digitação de frases.
- Armazenador - cada frase digitada é avaliada para verificação da estrutura da frase e reconhecimento das palavras digitadas.
- Refinador Contextual - verifica a melhor maneira de classificar a estrutura da frase e das palavras, caso seja necessária à avaliação de novas estruturas ou palavras desconhecidas.
- Analisador Morfológico - contem as regras iniciais de classificação morfológica das palavras e, por meio de interação com o refinador contextual, pode ter novas regras inseridas através de processo de aprendizado adaptativo.
- Analisador Sintático e Analisador Semântico - tem o mesmo comportamento e interação do módulo Analisador Morfológico.
- Gerador de Produções - responsável por avaliar e produzir as frases que serão enviadas ao usuário, com o intuito de obter mais informações sobre frases recebidas do usuário ou para expressar as informações armazenadas.

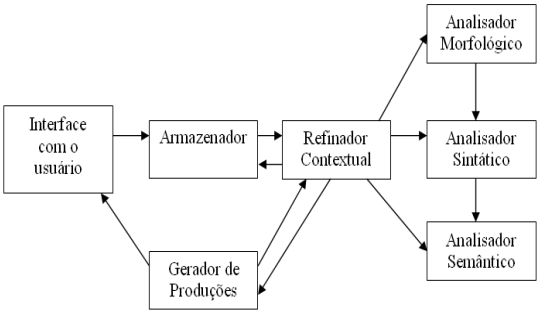


Fig. 4. Módulos do aplicativo adaptativo de aprendizado de linguagem natural.

O aplicativo é composto pela classe PhraseAnalyzer que efetuará primeiramente a separação das palavras da frase digitada por meio de seu método slicer. Após obter a quantidade de palavras da frase o método queryStructure fará a verificação da existência da estrutura recebida no aplicativo adaptativo. Caso a quantidade de elementos da frase seja incompatível com a estrutura pré-armazenada inicialmente no dispositivo (3 elementos), será acionado, automaticamente, o método createNewStructure. Este método executará a ação adaptativa exemplificada na figura 2 e criará uma nova estrutura que permitirá o armazenamento de uma frase com mais elementos (no caso do exemplo utilizado neste trabalho 4 elementos). Nos casos em que a estrutura da frase já exista ou no caso em que seja necessário criar um nova estrutura, as palavras da frase serão encaminhadas ao método queryWord para verificação da existência das palavras no aplicativo adaptativo e caso alguma palavra ainda não seja conhecida pelo aplicativo será acionado o método createNewWord que fará a inclusão da nova palavra no aplicativo.

O algoritmo do aplicativo, com o uso da tabela de decisão adaptativa, é o seguinte:

1) estando na configuração inicial da tabela executa a regra S;

2) enquanto houver símbolo a ser lido na cadeia de entrada faça;

2.1) executa a divisão da frase em seus componentes;

2.2) procura regra aplicável;

2.2.1) Se existir regra aplicável aceita a cadeia, senão executa o passo 2.5;

2.3) verifica se as palavras são todas conhecidas.

2.3.1) Se existir palavra desconhecida inclui a palavra na lista de palavras conhecidas;

2.4) fim enquanto.

2.5) executa a função adaptativa;

2.5.1) inclui nova regra;

2.5.2) volta ao passo 2.2;

V. V. CONCLUSÃO E PERSPECTIVAS FUTURAS

A partir dos experimentos realizados observou-se que os resultados obtidos foram adequados ao esperado. Os testes realizados contaram com frases de apenas 3 ou 4 elementos, e nos dois casos o aplicativo conseguiu verificar as palavras de cada frase e efetuar o armazenamento de novas palavras e gerar uma nova estrutura de reconhecimento de frase com 4 elementos.

A programação foi efetuada com a utilização da linguagem de programação Java devido a sua ampla utilização, por ser código portável e permitir a reutilização em trabalhos futuros. Trabalhos ainda precisam ser feitos para especificar e desenvolver uma linguagem que implemente a POC.

Cabe ressaltar que foi desenvolvido no presente trabalho apenas os módulos referentes à interface com o usuário e o módulo armazenador, ficando para trabalhos futuros o desenvolvimento dos módulos refinador contextual, morfológicos, sintáticos e semânticos.

As perspectivas de trabalhos futuros e potencial da solução aqui modelada e desenvolvida são muito grandes, podendo ampliar o escopo de outras soluções similares já desenvolvidas e que propõe sempre a identificação de palavras por meio de comparação com corpus anotado, sendo que a presente proposta sugere a possibilidade de iniciar o processo de identificação de palavras a partir do aprendizado de novas regras e novas palavras, por meio da interação direta com o usuário, tornando o processo de aprendizado mais próximo ao análogo processo de aprendizado humano.

REFERÊNCIAS

Periodicals (Artículos de revista):

[1] B.F.T. Azevedo e O.L. Tavares Um ambiente inteligente para aprendizagem colaborativa. XII SBIE 2001 – Simpósio Brasileiro de Informática na Educação, UFES, 2001.

[2] E. Brill A corpus-based approach to language learning. Thesis (PhD) - Department of Computer and Information Science of the University of Pennsylvania, Philadelphia, 1993, 154 p.

[3] C.E.D. Menezes e J.J. Neto. Um método híbrido para a construção de etiquetadores morfológicos, aplicado à língua portuguesa, baseado em autômatos adaptativos. Anais da Conferencia Iberoamericana en Sistemas, Cibernética e Informática, 19-21 de Julio, 2002, Orlando, Florida.

[4] J.J..Neto Uma solução adaptativa para reconhecedores sintáticos. Technical report, PCSPOLI-USP, São Paulo, Brasil, 1988.

[5] 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.

[6] L. Specia e L.H.M. Rino, O desenvolvimento de um léxico para a geração de estruturas conceituais UNL. Série de Relatórios do Núcleo Interinstitucional de Lingüística Computacional – NILC-TR-02-14. São Carlos, SP, setembro de 2002.

[7] C. Y. O. Taniwaki, e J.J. Neto. Autômatos Adaptativos no Tratamento Sintático de Linguagem Natural Boletim Técnico PBT/PCS, Escola Politécnica, São Paulo, 2001

[8] A. Tchemra, Tabela de decisão adaptativa:simulação de um autômato adaptativo. WTA 2008 – Segundo Workshop de Tecnologia Adaptativa. São Paulo, 2008, págs. 5-8.

Mario Marques nasceu em São Paulo, em 7 de Outubro de 1967. Graduou-se na Universidade Mackenzie e na Universidade de São Paulo e formou-se mestre no IPT.

Exerce atividades profissionais na Caixa Economica Federal e atividades de ensino na Escola Técnica Estadual de São Paulo e na faculdade Sumaré.

Entre seus campos de interesse estão o aprendizado automático de conhecimento, processamento de linguagem natural e máquinas adaptativas.

Uma proposta do uso de adaptatividade para busca de padrões musicais (5 de janeiro 2009)

P. R. N. Pedruzzi, R. A. Redder Jr. e J. J. Neto

***Resumo* — Este artigo apresenta uma proposta de utilização de adaptatividade para um sistema de busca musica. O trabalho[1] consiste na construção de um sistema capaz de identificar uma música, dentre um repertório previamente estabelecido, a partir de uma amostra de áudio contendo a execução de um pequeno trecho de uma de suas melodias, cantado ou assobiado.**

O principal foco do trabalho está no reconhecimento de melodias similares que podem apresentar transposição de tonalidade, dilatação de durações, erros de notas etc.

Primeiramente são apresentados alguns detalhes sobre o projeto desenvolvido, em seguida define-se formalmente a noção de proximidade/semelhança entre melodias e, por fim, propõe-se um algoritmo para reconhecimento de melodias próximas, utilizando adaptatividade.

***Palavras chave*— Reconhecimento de padrões (*pattern matching*), reconhecimento de cadeias (*string matching*), melodia (*melody*), autômato (*automaton*), adaptatividade (*adaptivity*), mínimos quadrados (*least squares*).**

I. Introdução

Este artigo apresenta os avanços obtidos no trabalho de um projeto final de graduação [1]. Tal trabalho consiste no desenvolvimento de um método de comparação de conteúdos musicais baseado em autômatos adaptativos, servindo de base para um protótipo de um sistema de identificação de melodias. A busca é realizada sobre um repertório previamente estabelecido, a partir de uma amostra de áudio contendo a execução de um pequeno trecho de sua melodia assobiado.

O principal objetivo deste artigo é apresentar o desenvolvimento deste autômato adaptativo para reconhecimento e busca de melodias, assim como alguns resultados preliminares.

Comumente as pessoas se deparam com o problema de tentar identificar uma música a partir do conhecimento de apenas um trecho da mesma, este trecho, no entanto, muitas vezes consiste apenas de uma melodia, ou um ritmo da música original. Esta situação é recorrente em diferentes contextos, como ao tentar identificar uma música ouvida em um filme, um programa de TV, uma rádio, etc.

Este problema em geral é difícil, devido à grande quantidade de músicas existentes e pela falta de ferramentas que possam auxiliar esta busca. As formas mais comuns de organização de repositórios musicais recaem sobre estilos musicais e nomes, o que não é o suficiente para endereçar o problema apresentado. Além disso, seria extremamente inviável para um indivíduo varrer um extenso repositório musical procurando pela música desejada. A forma mais comum atualmente de se resolver este problema, é com a busca de um especialista, como um vendedor especializado, por exemplo. Assim, percebe-se que as dificuldades apresentadas geram uma grande demanda por técnicas que possam auxiliar esse tipo de busca.

O principal foco do trabalho desenvolvido é a apresentação de uma forma de comparação de melodias baseados em autômatos adaptativos. Para uma avaliação efetiva de tal método de comparação, o protótipo de um sistema mais amplo que se apóia sobre tal técnica foi desenvolvido, a fim de prover um cenário de teste mais concreto.

II. Proximidade de Melodias

Quando uma pessoa canta uma melodia ou a toca em um instrumento, somos eventualmente capazes de identificar a que música aquela melodia pertence. Nosso cérebro é capaz de reconhecer estas semelhanças mesmo na presença de variações ou imprecisões na melodia que ouvimos.

Um exemplo típico de tais variações é a transposição tonal, em que a melodia é reproduzida com uma variação (logarítmica) fixa na altura de todas as notas, para mais ou para menos. Outro exemplo de variação é a dilatação ou contração das durações das notas que compõe tal melodia.

Em outros casos, há a ocorrência de erros na reprodução, tais como uma nota errada (com altura diferente), ou mesmo a omissão ou adição de notas à melodia original. Estes erros são, em geral, provenientes da incapacidade ou imprecisão do próprio executor, que reproduz a melodia que o mesmo tem em mente.

Tais fatos evidenciam a existência de um conceito subjetivo de proximidade entre melodias, e é exatamente a subjetividade deste conceito que dificulta a implementação da comparação de trechos melódicos por sistemas computacionais.

Assim, no escopo deste trabalho, alguns dos erros possíveis em uma reprodução por um ser humano foram identificados e categorizados, de modo a servir de base para o desenvolvimento da técnica proposta.

Desta forma, os erros abordados são:

1. Transposição tonal
2. Variação temporal

Este artigo é baseado em um projeto final de graduação desenvolvido pelos alunos Pedro R. N. Pedruzzi e Ricardo A. Redder Jr. sob a orientação do Prof. Dr. João José Neto, no Laboratório de Linguagens e Técnicas Adaptativas (LTA EP/USP).

3. Erros simples
   a. Troca de uma nota
   b. Adição de uma nota
   c. Omissão de uma nota

Tais categorias englobam grande parte dos erros cometidos durante a reprodução de uma melodia, e todas elas contribuem (em maior ou menor grau) para o conceito de proximidade de dois trechos de áudio. Percebe-se também, que tais erros só podem ser estabelecidos quando em comparação com uma segunda melodia, que é considerada a correta.
Assim, define-se que a medida de similaridade entre duas melodias será função dos erros (dentre estas categorias) encontrados em uma melodia quando comparada à outra.
Para mensurar os erros em cada categoria diferentes técnicas se fazem necessárias, assim, três técnicas diferentes foram empregadas para endereçar cada problema.

A. TRANSPOSIÇÃO TONAL E VARIAÇÃO TEMPORAL

Em um ambiente sem a presença de erros, pode-se analisar a proximidade entre duas melodias com a mesma quantidade de notas, assumindo que existe uma relação matemática que mapeiam aproximadamente as notas de uma melodia nas notas da outra. Uma relação mapeia as alturas e outra as durações. Sendo p1 e p2, respectivamente as alturas de uma nota do trecho 1 e sua correspondente no trecho 2; e d1 e d2 as durações destas; a relação que aproxima o mapeamento de durações é do tipo:

$$p_1 = A.p_2$$

A constante A representa uma proporcionalidade entre as durações, porém permitindo dilatações e contrações.
Para mapear as alturas utiliza-se a seguinte relação:

$$\log d_1 = \log d_2 + B$$

A constante B representa a transposição tonal. A relação logarítmica é necessária pelo fato de que a percepção do ouvido humano para alturas de notas é exponencial.
A partir destas relações de aproximação, calculam-se os parâmetros A e B que melhor aproximam a distribuição, utilizando o método dos mínimos quadrados [2]. O valor dos parâmetros obtidos eventualmente pode ser utilizado para avaliar a proximidade entre as melodias. Porém, nesta modelagem, o relevante não são os parâmetros obtidos da redução, e sim, a soma quadrática dos erros ao utilizá-los:

$$S_p = \sum (A.p_2 - p_1)^2$$
$$S_d = \sum (\log d_2 + B - \log d_1)^2$$

Quanto menor forem estas somas, mais próximas são as melodias comparadas. A distância entre as melodias é definida por uma soma ponderada destas somas, com pesos parametrizáveis.
Maiores detalhes sobre o método de comparação desenvolvido podem ser encontrados em [1].

B. ERROS SIMPLES

Para lidar com erros simples foi empregada uma técnica baseada em autômatos adaptativos [3]. A idéia básica é a construção de um autômato adaptativo a partir de uma das melodias que seja capaz de reconhecer melodias similares a esta, contendo eventuais erros simples devidamente espaçados. Então se avalia a semelhança entre as melodias aplicando o autômato sobre a outra melodia.
O primeiro passo do processo estabelecido consiste em quantizar as alturas das notas das melodias, acomodando-as em uma escala discretizada (que varia de 0 a 127), que corresponde à codificação utilizada em arquivos MIDI. Desta maneira as melodias podem ser vistas como cadeias de símbolos. Note que este processo de quantização não é um simples arredondamento, e envolve a estimativa de um fator de transposição que deve ser aplicado para reduzir ao máximo o erro de quantização. Este método foi chamado de quantização absoluta e foi descrito em [1].
A configuração inicial do autômato concebido é um caminho de estados e transições capaz de reconhecer cadeias de entrada equivalentes àquela que produziu o autômato, a menos de transposição tonal, que é corrigida no início do reconhecimento com base na primeira nota.
Porém ao receber um símbolo neste caminho para o qual não existe transição, ocorre uma ação adaptativa em que uma estrutura de novos estados e transições é incorporada ao autômato a partir do estado corrente. A nova configuração é capaz de contornar as situações de erro previstas (troca, adição ou omissão de uma nota).
Para melhor compreensão do processo considere o seguinte exemplo de comparação entre duas cadeias. Seja a cadeia de base para comparação: 69, 71, 73, 74, 76, 77, 76. A partir desta cadeia, o autômato da Figura 1 é gerado.

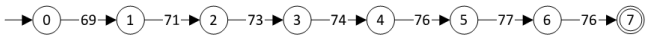


Figura 1 - Configuração inicial do autômato

Dada uma segunda cadeia: 69, 71, 74, 74, 76, 77, 76; que corresponderia à cadeia a ser comparada com a cadeia base. Tal cadeia serve como entrada para o autômato construído. Após consumir os dois primeiros valores (69 e 71) o terceiro valor (74) não pode disparar a transição, e sinaliza um erro. Assim uma ação adaptativa é disparada, gerando uma nova configuração para o autômato, que pode ser visto na Figura 2. Após esta ação, o reconhecimento continua, neste caso, o quarto valor seria consumido, disparando a transição do estado 8 para o estado 4, e em seguida o restante da cadeia seria consumido normalmente atingindo o estado.
Cada uma das três novas transições geradas (8 → 3, 8 → 4 e 8 → 5) indicam um tipo de erro: adição, troca ou omissão, respectivamente. Desta forma, o fato da cadeia seguir o caminho 8 → 4 indica que houve a troca de uma nota, ou seja, a troca da nota 73 (terceira nota) por uma nota 74.

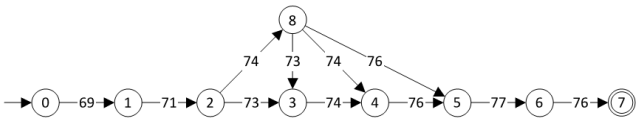


Figura 2 - Configuração após adaptação

Desta forma pode-se dizer que a distância entre as duas melodias em questão é de um erro simples, mais especificamente uma troca.

III. SISTEMA DE BUSCA

Para avaliar os métodos de comparação propostos o protótipo de um sistema de busca foi desenvolvido, de forma a englobar as partes mais relevantes de um sistema concreto.

O protótipo apresentado se limita a receber um sinal de áudio (um assobio) de um usuário, converter tal sinal para uma sequência de notas, e em seguida comparar tal sequência com um repositório de músicas previamente estabelecido. Através das comparações efetuadas pode-se estabelecer uma medida de distância entre cada música do repositório e o sinal de entrada, e assim gerar uma lista das músicas que mais se assemelham ao sinal de entrada.

O fluxo de informações do sistema pode ser visto na Figura 3.

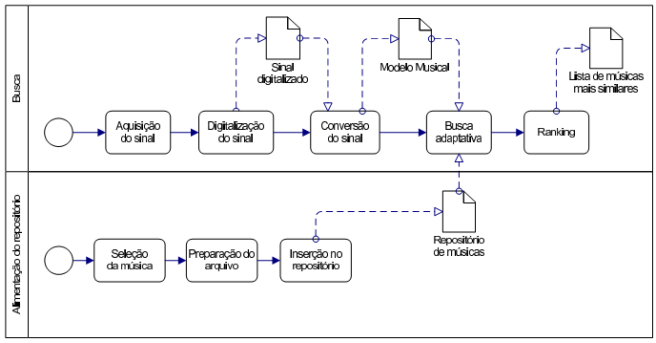


Figura 3 - Fluxo de informações do sistema

Neste sistema, inicialmente um sinal gerado pelo usuário (um assobio) é gravado, correspondendo a sua interpretação de uma música. Este sinal é digitalizado, e em seguida serve de entrada para um componente de conversão que converte o sinal digitalizado, contendo as frequências do sinal original, em um modelo musical que representa as notas assobiadas (início, duração e frequência de cada nota). O modelo musical gerado é quantizado de acordo com o método apresentado anteriormente. A partir da cadeia de notas quantizadas um processo de busca sobre as músicas do repositório é efetuado, utilizando-se o método de comparação adaptativa apresentado. Após esta comparação pode-se apresentar como resposta do sistema uma lista contendo as músicas do repositório mais similares à melodia de entrada, ordenadas pela similaridade.

A. BUSCA

Com base na definição de distância entre melodias apresentada na seção anterior é possível construir um mecanismo de busca simples de uma melodia sobre um repositório de músicas, comparando-se a melodia de entrada com todas as músicas do repositório. O processo segue da seguinte forma:

A partir do modelo musical gerado pelo componente de conversão, e após sua quantização, um autômato análogo ao da Figura 1 é gerado, esta é a cadeia que deve ser comparada com as músicas do repositório.

A comparação é feita aplicando-se ao autômato todas as melodias do repositório de maneira a utilizar uma janela deslizante; ou seja, para uma dada música, o autômato recebe inicialmente a primeira nota da cadeia representando a música sendo analisada, e consome a cadeia até que não seja mais possível continuar (caso encontre mais de um erro simples seguido), ou chegue ao final do autômato. Em seguida, o mesmo processo é repetido, porém tomando como base a nota seguinte, no caso, a segunda nota.

O caminho mais curto entre o estado inicial e o final é obtido quando a cadeia de entrada equivale exatamente à seqüência de notas da cadeia procurada. Porém, outras transições existem para flexibilizar a comparação, contornando alguns tipos de erros. Por este motivo, o autômato é capaz de reconhecer outras cadeias que não correspondem à cadeia exata da procurada.

Durante a execução deste processo, os tipos de erros identificados e suas localizações na cadeia são registrados. Desta forma, para uma dada música, o menor valor obtido (menor número de erros) é dado como o valor de erro para a música em questão.

Após a execução deste processo para todas as músicas do repositório é possível estabelecer para cada música o trecho mais próximo da melodia de entrada, além da quantidade de erros em relação a esta.

Com posse destas informações é possível reverter os erros gerados, e assim produzir uma melodia artificial, baseada na melodia de entrada, porém desfazendo-se os erros percebidos. Desta forma, se uma nota foi omitida, por exemplo, pode-se adicioná-la à melodia novamente. Isso possibilita a realização da comparação no domínio temporal apresentada anteriormente, obtendo-se assim mais uma medida de erro que pode auxiliar no estabelecimento da similaridade entre as melodias.

Ao fim da comparação da melodia de entrada com todas as músicas do repositório, têm-se algumas medidas que podem auxiliar no estabelecimento da similaridade entre a melodia de entrada, e cada música do repositório:

1. Número de erros entre a melodia e a música
   Proveniente da execução do autômato.
2. Aceitação ou não do autômato
   Proveniente da execução do autômato.
3. Valores numéricos de soma de distâncias
   Proveniente da comparação numérica.

B. ORDENAÇÃO

Os valores de cada medida apresentadas anteriormente influenciam na similaridade entre a melodia de entrada e uma música dada. Porém a importância que deve ser dada a cada dimensão não é clara, assim não é possível estabelecer de imediato uma fórmula que sumarize todos os dados obtidos.
Desta forma, a fim de avaliar o desempenho do método de comparação adaptativa, apenas os dados provenientes do autômato foram utilizados para ordenação.

IV. RESULTADOS

Inicialmente foi criado um repositório contendo 37 músicas, de diversos gêneros e todas em formato MIDI. Em seguida, 19 amostras de áudio (assobios) foram colhidas, todas de músicas existentes no repositório.
Todas as amostras foram submetidas ao sistema construído, colhendo-se a resposta dada. Conforme explicado na seção anterior, contabilizaram-se apenas os resultados provenientes do autômato, desta forma procurava-se ter uma idéia da eficiência de comparação do autômato proposto.
Por fim, as respostas foram analisadas, procurando-se dentro de cada resposta, a posição que a melodia procurada realmente encontrava-se. Ou seja, para um assobio correspondendo a uma música A, procurou-se na resposta do sistema qual a posição da música A na lista ordenada gerada. A compilação desta análise pode ser vista no gráfico da Figura 4.

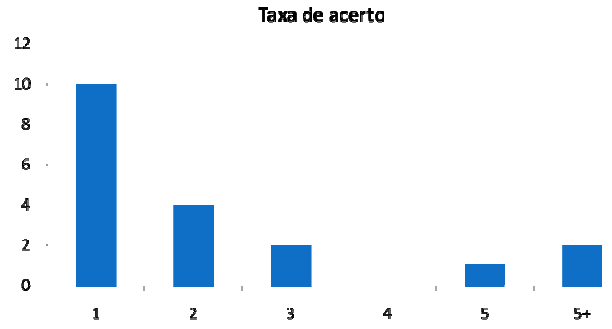


Figura 4 - Taxa de acerto para o conjunto de testes

O gráfico mostra que na maior parte dos casos o sistema identificou corretamente a música procurada como a mais similar à melodia de entrada.
Em outros casos, apesar da música desejada não corresponder a primeira colocação, ainda assim encontrava-se entre as três primeiras colocações.

V. TRABALHOS FUTUROS

Após a conclusão desta fase de trabalhos, alguns avanços e melhorias necessários se fazem evidentes:

1. Maior granularidade de erros
   Apenas três tipos de erros foram considerados para a criação do autômato proposto (omissão, adição e troca), porém percebeu-se que ao menos uma nova classe de erros se faz necessária, uma classe que indique uma pequena variação com relação à nota em questão, dando uma granularidade maior ao tipo de erro de troca.

2. Múltiplos erros
   Apenas erros simples foram considerados, que são os erros apresentados anteriormente, casos de erros seguidos não foram abordados, e levam ao fim da execução do autômato. Erros mais complexos poderiam ser considerados, por exemplo: TROCA, TROCA, OMISSÃO. A metodologia proposta não seria capaz de lidar com tais variações.

3. Definição de pesos para cada classe de erro
   Conforme citado anteriormente, diversos fatores contribuem para a variação da distância entre dois trechos de áudio, porém, não é claro a forma como dada um destes fatores contribui, assim, mais estudos, e avaliações se fazem necessárias para a determinação dos pesos mais adequados para cada dimensão.

4. Generalização do método de comparação
   O método de comparação proposto independe do fato da cadeia ser proveniente de uma sequência de notas, o que abre a possibilidade de generalização deste método para lidar com cadeias de símbolos de qualquer natureza. Uma generalização de todos os conceitos envolvidos se faria necessária, como alfabeto de entrada, classes de erros, etc. Tal método de comparação seria adequado para buscas de padrões em cadeias.

VI. CONCLUSÕES

Ao final do desenvolvimento deste trabalho pode-se perceber que a técnica proposta é consistente e é capaz de lidar de maneira eficiente com distúrbios comuns em reproduções de áudio por seres humanos, como trocas de nota ou omissões.
Percebe-se também que a combinação de diversas técnicas, a fim de mensurar diferentes dimensões de variações nos trechos comparados, parece ser o caminho mais indicado.
Além disso, fica evidente que o método ainda é prematuro para uso em sistemas de larga escala, porém a abordagem diferenciada do problema, ou seja, o uso de uma técnica adaptativa sobre tal campo, abre interessantes possibilidades.

REFERÊNCIAS

[1] Pedruzzi, P. R. N., e Redder, R. A.. "Reconhecimento e Busca Adaptativos de Padrões Musicais". Thesis. Escola Politécnica da Universidade de São Paulo, 2008.
[2] Otto Bretscher (1995). "Linear Algebra With Applications", 3rd ed.. Upper Saddle River NJ: Prentice Hall.
[3] J. J. Neto, "Adaptive Automata for Context-Sensitive Languages", SIGPLAN NOTICES, Vol. 29, n. 9, September, 1994, pp. 115-124.

Implementação de uma Solução Adaptativa para o Problema do Emparelhamento de Cadeias

(05 Janeiro 2009)

S. M. Melo, I. A. M. Rodrigues, A. A. de Castro Jr.

***Resumo*— O problema do emparelhamento de cadeias consiste em encontrar ocorrências de uma cadeia de caracteres (conhecida como padrão) dentro de outra cadeia de tamanho maior ou no corpo de um texto. Existe uma grande variedade de soluções propostas para este problema, uma delas baseada em autômatos finitos. Rodrigues *et al*[1] descreveu uma solução que utiliza os autômatos adaptativos para criar um conjunto de submáquinas que constroem o autômato finito que reconhece o padrão. Entretanto, a solução proposta não foi implementada completamente. Neste trabalho, pretende-se complementar os resultados da proposta adaptativa apresentada por Rodrigues, através da implementação completa do modelo proposto, da realização de um estudo sobre a complexidade de tempo e espaço do modelo apresentado e da comparação com outras técnicas existentes. Vale ressaltar que este trabalho apresenta os resultados parciais de um projeto de conclusão de curso com término previsto para junho de 2009. O trabalho está sendo realizado sob a orientação do Prof. Amaury Antônio de Castro Junior, do Curso de Sistemas de Informação do campus de Coxim (CPCX) da Universidade Federal de Mato Grosso do Sul (UFMS).**

Palavras-chave*— *autômatos adaptativos, emparelhamento de cadeias, Adaptools, complexidade de algoritmos.

I. INTRODUÇÃO

O problema do emparelhamento de cadeias consiste em encontrar ocorrências de uma cadeia de caracteres dentro de outra cadeia de tamanho maior ou no corpo de um texto. Este problema é também conhecido como *string-maching* ou *pattern-matching* [2,3].

O crescimento do uso de alinhamento de seqüências na busca de soluções para o problema de seqüenciamento genético, dentre outros tantos tipos de aplicações ligadas ou não à biologia molecular, torna o problema do emparelhamento de cadeias alvo de muitos trabalhos de pesquisa em Ciência da Computação.

Há, na literatura, uma grande variedade de algoritmos para o problema do emparelhamento de cadeias, entre eles destacam-se o de Força Bruta, o de KMP (Knuth, Morris e Pratt) e o de Boyer-Moore, considerados importantes e interessantes de serem estudados [3]. Uma outra solução bastante eficiente é baseada na utilização de autômatos finitos [2,5]. Esta última constrói um autômato finito que é responsável pela busca, na cadeia de texto, de todas as ocorrências de um determinado padrão. Para cada padrão, existe um autômato de emparelhamento de cadeias correspondente. Esse autômato é obtido a partir do padrão em uma etapa de pré-processamento, antes de ser utilizado para a busca do respectivo padrão na cadeia de texto. Portanto, para cada padrão a ser buscado no texto é necessária a construção de um novo autômato.

Rodrigues *et al* [1,6] apresentaram uma alternativa utilizando os autômatos adaptativos para o mesmo problema. Os autômatos adaptativos são dispositivos que possuem algumas transições especiais que quando acionadas, executam funções adaptativas que modificam a topologia do próprio autômato. Estas funções adaptativas acrescentam ou eliminam transições e estados, permitindo que a estrutura e o comportamento do autômato sejam dinamicamente modificados.

Através dos autômatos adaptativos é possível automatizar a realização do pré-processamento, possibilitando a adequação dos parâmetros de entrada e a eliminação de comparações desnecessárias, provendo uma solução alternativa baseada no modelo adaptativo. Entretanto, faz-se necessário um estudo sobre a complexidade de tempo e espaço desta proposta, bem como a sua implementação e a comparação com outras técnicas existentes.

II. AUTÔMATO DE EMPARELHAMENTO DE CADEIA

A utilização de autômatos de emparelhamento de cadeias exige a construção de um autômato para cada padrão *P* existente a partir de uma etapa de pré-processamento. Após sua construção o autômato é usado para pesquisar a ocorrência do padrão no corpo do texto.

As etapas para busca de um padrão no texto são representadas pela Figura 1. A busca é realizada em duas etapas: a primeira é a construção do autômato (pré-processamento); a segunda é o uso deste autômato para realizar a busca e encontrar todas as ocorrências do padrão no texto.

S. M. Melo e I. A. M. Rodrigues são alunas do curso de Sistemas de Informação da Universidade Federal de Mato Grosso do Sul (UFMS) no Campus de Coxim (CPCX), Av. Márcio de Lima Nantes, S/N – Vila da Barra – CEP: 79400-000 - Coxim/MS, Brasil (endereço eletrônico: silmorita@gmail.com; iranirv@gmail.com).

A. A. Castro Jr. é docente do Departamento de Ciências Exatas (DEX) do Campus de Coxim (CPCX) da Universidade Federal de Mato Grosso do Sul (UFMS), Av. Márcio de Lima Nantes, S/N – Vila da Barra – CEP: 79400-000 - Coxim/MS, Brasil (endereço eletrônico: amaury.ufms@gmail.com)

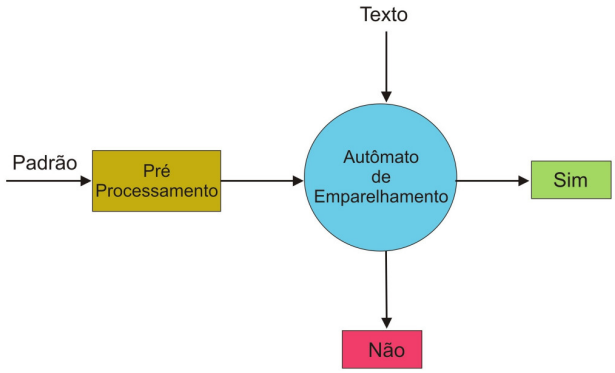


Fig. 1. Seqüência de etapas para criação e utilização de autômatos para o problema de emparelhamento de cadeias.

Inicialmente, é necessário definir uma função auxiliar σ, denominada função sufixo que mapeia de Σ* em {0,1,..*m*}, onde $\sigma(x)$ é o tamanho do maior prefixo de *P* que é o sufixo de *x*.

Por exemplo, se $p = ab$, $\sigma(e) = 0$, $\sigma(ccaca) = 1$, $\sigma(ccab) = 2$, o autômato de emparelhamento de cadeias que corresponde a um padrão $P[1..m]$ é definido da seguinte forma:

- Estado inicial q_0 é o estado 0,
- *P* é uma cadeia de padrão fixa,
- $Q = \{0,1,...,m\}$ é um conjunto finito de estados,
- *m* é o único estado aceitável,
- Σ é um alfabeto de entrada finito.
- A função de transição δ é descrita pela seguinte equação, para qualquer estado *q* e caractere *a*:

$$\delta(q, a) = \sigma(P_q a)$$

- O novo estado $\delta(q, a)$, corresponde ao prefixo de P com maior comprimento que é também sufixo de $P_q a$.

Para ilustrar a construção do autômato vamos considerar o padrão $P = ababaca$.

O autômato derivado do padrão *P* é mostrado abaixo.

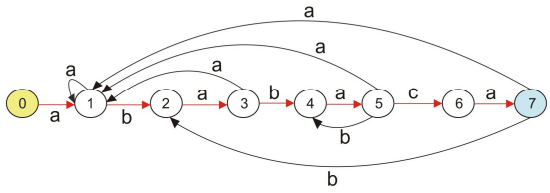


Fig. 2. Diagrama de transição de estados correspondente ao autômato de emparelhamento para o padrão $P = ababaca$.

O estado 0 é o estado inicial. O estado 7 é o único estado de aceitação. Uma aresta orientada do estado *i* para um estado *j* rotulado com a representação $\delta(i,a) = j$. As arestas direcionadas para direita, que formam a "espinha dorsal" do autômato, correspondem a seqüência de caracteres do padrão. As arestas que estão direcionadas a esquerda correspondem ao reconhecimento de prefixos durante o casamento dos caracteres do padrão e do texto. No diagrama, foram omitidas as arestas que retornam para o estado inicial.

A utilização do autômato no texto $T = abababacaba$ é mostrada abaixo. Neste exemplo, apenas uma ocorrência do padrão foi encontrada.

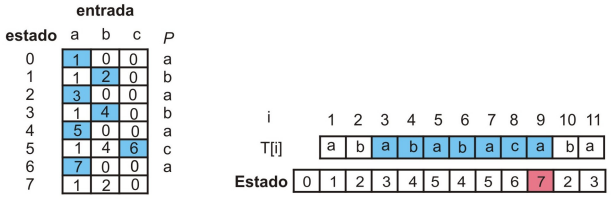


Fig. 3. Função de transição (esquerda) e operação do autômato sobre o texto T (direita).

III. AUTÔMATOS ADAPTATIVOS

Autômato adaptativo é uma instância de dispositivo adaptativo [7] que tem como camada subjacente um autômato de pilha estruturado (APE) [8].

De acordo com Neto [4], um autômato adaptativo *M* é definido por uma máquina de estados inicial E_0 a qual será submetida uma cadeia *W* de comprimento *n*. E_n é a máquina de estados final, depois de reconhecer a cadeia *W* inteira. E_i é uma máquina de estados intermediária que será submetida a uma cadeia W_i, que é uma sub-cadeia de *W*, sem os *i* primeiros elementos. Definindo *W* como sendo constituída pelos elementos $a_0a_1a_2...a_{n-1}$, pode-se dizer que a trajetória de reconhecimento do autômato adaptativo pela cadeia *W* é:

$$(E_0, a_0) \text{ -> } (E_1, a_1) \text{ -> } (E_2,a_2) \text{ ->...->} (E_{n-1},a_{n-1}).$$

No modo de operação do autômato adaptativo, o conceito de passo deve ser estendido para que se possa incorporar as ocasionais mudanças na topologia do dispositivo subjacente. Com isso, pode-se distinguir duas situações possíveis na operação dos autômatos adaptativos:

- Passo subjacente: corresponde à execução de um passo na operação do dispositivo subjacente, neste caso, o APE. Nesta situação, não ocorre nenhuma alteração no conjunto de regras que define o APE.
- Passo adaptativo: corresponde à execução de um passo de operação associado a alguma ação adaptativa não nula. Nesta situação, as componentes que definem o conjunto de regras do dispositivo subjacente são modificadas pelos efeitos da ação adaptativa correspondente e, conseqüentemente, a sua topologia deve refletir essas alterações.

Portanto, as transições que compõem o autômato adaptativo são representadas por regras de produção com a seguinte forma:

$$(\gamma g, s, \sigma\alpha), \mathcal{A} :\rightarrow (\gamma g', s', \sigma'\alpha), \mathcal{B},$$

onde:

- γ corresponde ao meta-símbolo que representa o conteúdo da pilha não considerado pelo autômato (não influencia na aplicação da produção);

- g e g' representam o estado armazenado no topo da pilha antes e depois da aplicação da transição, respectivamente;
- s e s' representam o estado corrente da transição e o próximo estado, respectivamente;
- σ é o símbolo do alfabeto que será consumido da cadeia e σ' é o símbolo do alfabeto que será empilhado na cadeia (ambos podem ser iguais a cadeia vazia, representada pelo símbolo e);
- α corresponde ao meta-símbolo que representa o restante da cadeia não considerada pelo autômato (não influencia na aplicação da produção);
- $\mathcal{A}$ e $\mathcal{B}$ são chamadas a funções adaptativas a serem executadas antes e depois da mudança de estados determinada pela produção (quando é executado um passo subjacente, $\mathcal{A} = \mathcal{B} = \emptyset$). As funções adaptativas podem ter *n* parâmetros e assumem a seguinte forma $\mathcal{A}(a_1, a_2, ...,a_n)$, em que a_i são argumentos que assumirão valores a serem usados no corpo da função. As funções adaptativas são declaradas conforme descrito em [8].

IV. MODELO ADAPTATIVO PARA O EMPARELHAMENTO DE CADEIAS

A utilização de autômato finito para solucionar o problema do emparelhamento de cadeias impõe a necessidade de um pré-processamento do padrão para a construção de um autômato de emparelhamento que reconheça aquele padrão. Dessa forma, para cada padrão a ser procurado no texto, a tecnologia adaptativa se coloca como uma alternativa para automatizar este pré-processamento. O modelo proposto por Rodrigues *et al* [1] utiliza os autômatos adaptativos na fase de construção do autômato usado na busca. Este modelo possui uma biblioteca de funções adaptativas que são usadas durante todo o processo de busca do padrão no texto. Neste modelo, para cada símbolo lido no padrão um novo estado é criado e novas transições são inseridas para todos os símbolos do alfabeto.

Na construção de um autômato é necessário saber o tamanho do padrão que será reconhecido, isto é feito através de uma submáquina inicialmente com apenas um estado e uma função adaptativa. Esta submáquina percorre o padrão e se automodifica, construindo o autômato auxiliar que representa o tamanho do padrão.

Outro autômato que deve ser construído é um autômato que reconhece exatamente uma vez a cadeia de entrada, que será usado para a busca no texto. Esse autômato possibilita que todas as ocorrências do padrão sejam encontradas consultando apenas uma vez cada símbolo do texto.

A inserção de novas transições e estados de destino no autômato deve considerar a possibilidade de sobreposição de símbolos de ocorrências sucessivas do padrão no texto. Por exemplo, se o padrão é *aba* e o texto é *ababa*, então o último caracter *a* da primeira ocorrência do padrão no texto, corresponde ao primeiro da segunda ocorrência. De acordo com as sobreposições possíveis é necessário determinar se a função cria um novo estado ou cria uma transição para um estado já existente. Para solucionar este problema, é necessário encontrar o maior prefixo de *P* que é um sufixo da subcadeia que já é reconhecida pelo autômato, concatenado com o símbolo para o qual será criado a transição. Este processo é realizado através de uma função adaptativa que cria um autômato adaptativo invertido com os símbolos do padrão, percorrendo-o de trás para frente.

Após percorrer todo o padrão e criar o autômato, o símbolo ├ (final do padrão) é encontrado. Com isso, uma função adaptativa é executada para remover as transições associadas com a função que criava os estados e acrescentava as transições no autômato de emparelhamento construído. Além disso, essa função insere uma ação adaptativa que será usada, posteriormente, para contagem do número de ocorrências do padrão no texto.

Agora com o autômato construído pode-se iniciar a busca no texto, que é iniciada no primeiro símbolo do texto e prossegue consumindo cada símbolo até o final. Cada símbolo lido faz com que o autômato mude de estado e caso alcance o estado final significa que o padrão foi encontrado no texto.

V. ADAPTOOLS

O AdapTools é um software livre, escrito em JAVA, que foi originalmente desenvolvido por Pistori [9] com o objetivo de executar autômatos adaptativos, bem como suas especializações: autômatos de pilha estruturados e autômatos de estados finitos. O estudo e a aplicação dessa ferramenta no desenvolvimento de soluções adaptativas para problemas reais podem dar suporte para o desenvolvimento de outras aplicações que façam uso de Tecnologia Adaptativa. A maior vantagem dos dispositivos baseados na tecnologia adaptativa é a sua facilidade de uso, sua relativa simplicidade e o fato de sua operação poder ser descrita de forma incremental, e seu comportamento, programado para se alterar dinamicamente em resposta aos estímulos de entrada recebidos.

O núcleo do Adaptools é composto por uma máquina virtual que executa uma versão levemente modificada de um autômato adaptativo. Essas pequenas modificações no formalismo original simplificam e uniformizam o formato de especificação de transições internas, transições externas e das ações adaptativas elementares. Com isso, todos os elementos do dispositivo podem ser apresentados através de uma única tabela [9,10].

VI. CONSIDERAÇÕES FINAIS

Este trabalho está sendo desenvolvido como projeto final de graduação do Curso de Sistemas de Informação do campus de Coxim (CPCX) da Universidade Federal de Mato Grosso do Sul (UFMS), sob orientação do Prof. Amaury Antônio de Castro Junior, com término previsto para junho de 2009.

Um dos objetivos deste trabalho em andamento é a implementação completa do modelo adaptativo proposto por Rodrigues *et al* [1], utilizando a ferramenta Adaptools. Com isso, será possível simular e analisar a solução proposta com todo o suporte gráfico da ferramenta.

Vale ressaltar que na solução adaptativa inicialmente proposta, quanto maior o tamanho do alfabeto, mais funções

adaptativas são necessárias. Dessa forma, pode-se verificar a possibilidade de criação de uma solução adaptativa para evitar essa manipulação das funções adaptativas. Além disso, pretende-se propor uma nova versão das funções adaptativas que possam reaproveitar os estados e transições de um autômato para a busca de novos padrões.

Um outro resultado previsto deste trabalho é um estudo sobre a complexidade de tempo e espaço do modelo adaptativo para o problema de emparelhamento de cadeias, bem como uma comparação com outras técnicas existentes.

REFERÊNCIAS

[1] E. S. C. Rodrigues, F. A. Rodrigues, R. L. A. Rocha, "Autômatos Adaptativos para Emparelhamento de Cadeias," in II Workshop de Tecnologia Adaptativa (WTA2008), São Paulo/SP, LTA/PCS/EPUSP, 2008.

[2] Cormen, Thomaz H.; Leiserson, Charles E.; Stein, Clifford e Rivest, Ronald L. – Algoritmos: Teoria e prática. 1.ed. Rio de Janeiro, Campus, 2002.

[3] Szwarcfiter, Jayme Luiz e Markenzon, Lilian – Estrutura de dados e seus algoritmos. 2.ed. JC, 2004.

[4] J. J. Neto, "Contribuições à metodologia de construção de compiladores", Tese de Livre Docência, USP, São Paulo,1993.

[5] Hopcroft, John E.; Motwani, Rajeev e Ullman, Jeffrey D. – Introdução à Teoria de Autômatos, Linguagens e Computação. 2.ed. Rio de Janeiro, Campus, 2002.

[6] J. J. Neto - Um Levantamento da Evolução da Adaptatividade e da Tecnologia Adaptativa - IEEE Latin America Transactions , vol. 5, no. 7, nov. 2007.

[7] 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, 2002, pp. 234-250.

[8] J. J. Neto, Adaptive automata for context-dependent languages, ACM SIGPLAN Notices, Vol 29, n. 9, pp 15 – 24, 1994.

[9] Pistori, H. e Neto, J. J. AdapTools: Aspectos de Implementação e Utilização Boletim Técnico PCS, Escola Politécnica, São Paulo, 2003.

[10] JESUS, L.; SANTOS, D. G.; CASTRO JR., A. A.; PISTORI, H. AdapTools 2.0: Aspectos de Implementação e Utilização. Revista IEEE América Latina. Vol. 5, Num. 7, ISSN: 1548-0992, Novembro 2007. (p. 527-532)

Silvana Morita Melo é aluna de graduação do quarto ano do curso de bacharelado em Sistemas de informação da Universidade Federal de Mato Grosso do Sul (UFMS) Campus de Coxim (CPCX), onde desenvolveu atividades na área de desenvolvimento web e manutenção de páginas (2007), foi aluna voluntária de iniciação científica no projeto: "Estudo e avaliação das técnicas e das ferramentas para o projeto de data warehouse" (2007-2008), participou também do projeto de extensão intitulado: Todo dia é dia de parque (2008), ambos na mesma instituição.

Irani Aparecida Moreira Rodrigues é aluna de graduação do quarto ano do curso de bacharelado em Sistemas de informação da Universidade Federal de Mato Grosso do Sul (UFMS) Campus de Coxim (CPCX).

Amaury Antônio de Castro Junior é graduado em Ciência da Computação pela Universidade Federal de Mato Grosso do Sul (1997) e mestre em Ciência da Computação pela mesma universidade (2003). Atualmente, é aluno de doutorado da Escola Politécnica da USP, sob a orientação do Prof. João José Neto. É membro do Laboratório de Linguagens e Técnicas Adaptativas e do Grupo de Pesquisa em Engenharia e Computação da UCDB e professor assistente da Universidade Federal de Mato Grosso do Sul (UFMS), Campus de Coxim (CPCX). Tem experiência na área de Ciência da Computação, com ênfase em Teoria da Computação, atuando nos seguintes temas: Tecnologias Adaptativas, Autômatos Adaptativos, Projeto de Linguagens de Programação e Modelos de Computação.

Uma proposta de aplicação da Tecnologia Adaptativa na Teoria Algorítmica do Aprendizado

R. I. Silva Filho, R. L. A. Rocha

[1]

***Abstract*— In this preliminary work, we propose an architecture of machine learning to use the adaptive technology and the model of identification in limit applied in inference processes.**

I. INTRODUÇÃO

Este trabalho tem como objetivo apresentar o atual estágio de desenvolvimento da linha de pesquisa do autor, que é propor um modelo de aprendizado algorítmico que utilize a tecnologia adaptativa. De maneira mais específica, iremos estudar a viabilidade de uma descrição do modelo de identificação no limite utilizando o modelo dos autômatos adaptativos. O modelo de identificação no limite é um tipo particular de modelo de computação não convencional proposto por Mark Gold e que hoje faz parte da área de estudo da Teoria Algorítmica do aprendizado (*Algorithmic Learning Theory*).

O trabalho está organizado da seguinte forma. Na segunda sessão, temos uma apresentação das motivações em que se baseia este trabalho. Tais motivações têm um caráter multidisciplinar com origem em áreas como a Linguística, a Filosofia da Mente e a Inteligência Artificial.

A terceira sessão introduz a Teoria Algoritmica do Aprendizado. Trata-se de um sumário que contém o essencial para esclarecer o contexto teórico específico. Aqui serão apresentados os principais termos e definições relacionados com a inferência indutiva.

A quarta sessão trata da identificação no limite, o critério de aprendizagem assintótica para a identificação de regras desconhecidas. Uma verificação de sua relação com a Tecnologia Adaptativa é vista na quinta sessão, onde também é proposto um esboço para uma arquitetura que envolva as duas áreas de estudo. As consequências e possíveis desdobramentos futuros deste trabalho preliminar serão discutidos na conclusão.

A. *Nomeclatura*

Os conceitos básicos da teoria dos autômatos e dos autômatos adaptativos, bem como os aspectos gerais relacionados com a teoria das linguagens formais são considerados como conhecidos. Como referência para esses temas, indicamos [7] , [10]. Todo autômato é definido em função de um conjunto Σ finito de símbolos denominado alfabeto. Toda linguagem é definida na forma $L_i \subseteq \Sigma^*$, onde o símbolo "*" representa o fecho transitivo e reflexivo (também chamado de estrela de Kleene). A expressão $\{R_l\}_{l=1}^{L}$ representa o conjunto das regras genéricas $(R_1, R_2, ...R_L)$ com $L \in \mathbb{N}$. O símbolo $\#$ representa uma pausa e não é um elemento do conjunto Σ. O conjunto $\mathbb{N}$ é o conjunto dos números naturais

II. MOTIVAÇÃO

Nossa motivação parte de uma hipótese: o ser humano é uma máquina ótima para a busca de soluções[11]. Se um indivíduo é confrontado com um determinado problema, sua reação segue um padrão estabelecido por sua intuição, experiência e conhecimento formal. Partindo do pressuposto de que todo indivíduo possui uma representação interna do mundo que o cerca, e de que tal representação guarda fatos e permite ao indivíduo realizar predições sobre os comportamentos futuros desse ambiente, assumimos que tal representação pode sofrer mudanças e estar sujeita a adaptações em função dos estímulos externos. Essas mudanças buscam proporcionar ao indivíduo uma "visão" mais completa dos comportamentos que regem o ambiente onde ele se insere. Assim, a representação do mundo disponível ao sujeito nunca (e este é um ponto crucial) será completa. Mais ainda: a representação será dinâmica e sujeita a um processo de construção ao longo da vida do indivíduo. Desta forma, o indivíduo procura "cobrir" a realidade que o cerca usando os recursos de representação interna que ele possui.

Inicialmente, vamos abstrair questões relacionadas à formação e a natureza da memória. Nossa abordagem consiste em ter uma representação do "conhecimento" como o objeto sujeito a ser alterado pelos comportamentos adaptativo e indutivo que veremos descritos nas sessões seguintes. É conveniente esclarecer que o propósito aqui não é criar um modelo de arquitetura cognitiva. Nosso objetivo é desenvolver uma arquitetura para o aprendizado de máquina que possua atributos semelhantes ao aprendizado humano. Porém, diferente do ponto de vista conexionista[2], não adotaremos uma abordagem estrutural e sim funcional[12], ou seja, não incorporaremos os atributos de interesse da aprendizagem humana propondo um modelo que imite as estruturas que compõem o cérebro. Nossa busca baseia-se em *modelos computacionais.* Nosso interesse é por representações comportamentais que nos disponibilizem as

[1] R. I. Santos Filho é pesquisador e doutorando no Laboratório de Linguagens e Técnicas Adaptativas.

R. L. A. Rocha é pesquisador e professor Doutor no Laboratório de Linguagens e Técnicas Adaptativas.

características que queremos ver incorporadas em nossa proposta de aprendizado de máquina, mesmo que estruturalmente tais modelos computacionais não lembrem qualquer elemento neurofisiológico relacionado com a cognição.

Assim, assumimos que já temos em mãos um objeto que se constitui um modelo daquilo que iremos acatar como o "conhecimento". No nosso caso, esse modelo serão as linguagens formais. Da mesma forma que não é nossa finalidade explicar a estrutura neurofisiológica envolvida no aprendizado, não nos preocuparemos em abordar como o conhecimento tem seu inicio partir do surgimento do indivíduo. Apenas assumimos que a nossa arquitetura já conta com um "conhecimento inato" e prosseguiremos analisando como a estrutura que representa esse conhecimento sofre as dinâmicas descritas ao longo deste trabalho.

Por enquanto, o importante é destacar que, por influência dessas dinâmicas, o indivíduo experimenta, ao longo de sua vida, a ação de inferir maneiras mais corretas de predição e construção do conjunto de regras $\{R_l\}_{l=1}^{L}$ que governa o ambiente onde ele está inserido. Assim, dependendo do momento ao qual o indivíduo foi exposto a um determinado problema, sua reação pode ser diferente da reação que seria tomada em relação a qualquer outro momento, tanto no passado quanto no futuro. Tais considerações estão ilustradas na figura abaixo. Neste diagrama conceitual, observamos uma linha diagonal em que o conhecimento $\mathfrak{M}$ do indivíduo sofre alterações em função dos estímulos $(e_1, e_2, ...e_n)$ que ocorrem em diferentes momentos no tempo. $\mathfrak{M}$ guarda a representação $\mathfrak{R}$ das regras do ambiente, onde $\mathfrak{R} \subseteq \{R_l\}_{l=1}^{L}$. Se pudéssemos "congelar" no tempo cada instância de $\mathfrak{M}$ antes desta mudar, veríamos as diferentes maneiras de como ele reagiria, em diferentes momentos da sua história. Assim, as linhas verticais indicam o comportamento de uma mesma instância de $\mathfrak{M}$ sujeita a estímulos diferentes, caso $\mathfrak{M}$ fosse estático, enquanto que as linhas horizontais indicam os comportamentos das diferentes instâncias de $\mathfrak{M}$, sujeitas ao mesmo estímulo.

III. INFERÊNCIA INDUTIVA

Se uma criança cresce observando um número cada vez maior de sentenças da sua língua nativa, eventualmente desenvolve uma gramática que converge para um gerador de sentenças válidas dessa linguagem [14]. Do ponto de vista do diagrama ilustrado na Fig. 1, observaríamos que as

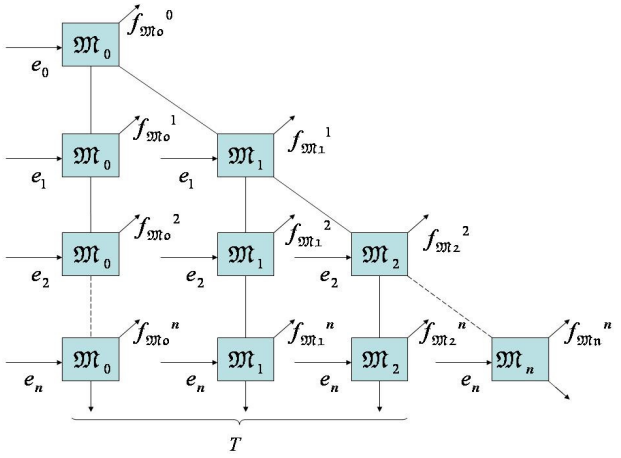


Figura 1: Comportamento do aprendizagem de um indivíduo. As flechas diagonais representam a resposta do indivíduo frente a um estímulo e_i.

reações e o discurso da criança se aprimoram à medida que ela é submetida a novas construções sintáticas e a novos exemplos de vocabulário, de modo a inferir indutivamente as regras pertinentes à linguagem nativa.

O termo inferência indutiva denota a geração e a escolha de hipóteses sobre as regras gerais que governam um determinado domínio, a partir de um fluxo de exemplos disponíveis ao longo do tempo[1]. A Inferência Indutiva é um *framework* de aprendizado, em que "aprendizado" é entendido como a capacidade de identificar um determinado conjunto dentro de uma classe de escolhas possíveis [1].

A inferência indutiva centraliza sua atenção na figura do aprendiz. Podemos utilizar a criança mencionada anteriormente como exemplo de um aprendiz. No processo de aprendizado, o aprendiz recebe um fluxo de dados sobre uma linguagem desconhecida para ele. De posse desses dados, ele, de tempos em tempos, gera e modifica suas hipóteses acerca do funcionamento dessa linguagem.

Um problema de inferência indutiva é visto como a quádrupla $I = \langle \mathfrak{R}, \mathfrak{H}, \mathfrak{T}, \mathfrak{I} \rangle$ [13].

$\mathfrak{R}$ é representação das regras do ambiente. Para nós, toda linguagem formal L será uma representação válida.

$\mathfrak{H} = (\phi_1, \phi_2, \phi_3, \ldots)$ é o espaço de hipóteses, consistindo de um conjunto de representações tal que, para cada regra, existe pelo menos uma hipótese que a represente nesse espaço. O nosso espaço de hipóteses será constituído por sistemas programáveis fixados, ou seja, uma enumeração de todas Máquinas de Turing. Porém, também é possível utilizar programas LISP, Gramáticas, etc. Dessa forma, cada ϕ_i denota uma função recursiva parcial ou total.

O conjunto $\mathfrak{T} = \{\eta_a \in L : a \in \mathbb{N}\}$ é o fluxo de dados ao qual o aprendiz tem acesso e representa os exemplos ou textos de uma linguagem L. Os exemplos são uma apresentação admissível da própria linguagem, desconhecida pelo aprendiz.

A Teoria Algoritmica do Aprendizado abstrai a figura do aprendiz na forma de um dispositivo recursivo $\mathcal{A}$ que recebe

como entrada $(\eta_a \cup \{\#\})^*$, onde o símbolo $\#$ representa uma pausa e, de tempos em tempos, apresenta como uma saída um elemento do conjunto $\mathfrak{H}$. Um aprendiz $\mathcal{A}$ pode ser uma função parcial ou total. Não é necessário que $\mathcal{A}$ seja capaz de aprender toda e qualquer linguagem, mas ele deve ser capaz de lidar com uma coleção S de classes de linguagens.

Uma classe de linguagens pode ser aprendida se e somente se existe um aprendiz tal que, para um fluxo de textos apresentados, $\mathcal{A}$ gera as hipóteses corretas sobre o comportamento da linguagem pertencente à classe. Grande parte dos esforços realizados no âmbito do estudo da Teoria Algorítmica do Aprendizado está na análise das classes S. Existem três tipos de classes envolvidas:

• **Classe recursiva**. A classe é descrita como uma lista $\mathbb{S}_{L_r} = (L_0, L_1, \ldots)$ de linguagens onde $\{\langle e, x\rangle : x \in L_e\}$ é recursiva.

• **Classe recursivamente enumerável**. A classe é descrita como uma lista $\mathbb{S}_{L_{re}} = (L_0, L_1, \ldots)$ de linguagens onde $\{\langle e, x\rangle : x \in L_e\}$ é recursivamente enumerável.

• **Classe geral**. A classe é descrita como uma lista $\mathbb{S}_{L_g} = (L_0, L_1, \ldots)$ de linguagens.

Por fim, $\mathfrak{I}$ é um critério de aprendizagem e refere-se à estratégia de produções das hipóteses pelo aprendiz em relação aos exemplos apresentados. O critério adotado neste trabalho é aquele criado por Gold[6], denominado de Identificação no Limite.

IV. IDENTIFICAÇÃO O LIMITE

Critérios de Aprendizado. Um critério de aprendizagem se refere ao modo como a sequência de hipóteses de saída será construída em relação aos exemplos recebidos até então. A lista abaixo menciona os principais, pois a literatura apresenta outras variações[5].

• **Aprendizado Explanatório**: O aprendiz $\mathcal{A}$ gera uma sequência $\Phi \subset \mathfrak{H}$ finita de hipótese que converge para um índice i, que se constitui na resposta correta. O tipo de convergência nesse critério é chamado de convergência sintática[6].

• **Aprendizado Comportamentalmente Correto**, ou Aprendizado no Limite: Este critério exige que a semântica (e não a sintaxe) das hipóteses convirjam corretamente no limite. O processo de inferência pode continuar modificando as suas hipóteses. $\mathcal{A}$ gera uma sequência Φ infinita de hipóteses que, após um índice i, são todas descrições corretas de L. A diferença entre os dois critérios é que, no primeiro, as hipóteses param de mudar e a linguagem L é computada pela última hipótese, enquanto que, no segundo critério, as hipóteses não param de mudar; é necessário apenas que elas reconheçam L pelos seus exemplos[4].

Podemos verificar, portanto, que a garantia de um critério de convergência é extremamente importante qualquer que seja o critério de aprendizado envolvido. Assim, é necessário que o aprendiz convirja sintáticamente ou semanticamente em função dos exemplos apresentados. Textos que pertencessem a linguagens fora da classe a ser aprendida fariam o aprendiz convergir para hipóteses erradas. As razões para isso residem no fato de existir um número incontável de linguagens diante de um número contável de índices.

V. IDENTIFICAÇÃO O LIMITE E A TECNOLOGIA ADAPTATIVA

Vamos tratar agora da questão do funcionamento do aprendiz. Apesar do aprendiz ser visto até agora como uma caixa-preta, existe um método universal para implementá-lo denominado de identificação por enumeração[5]. Ele se baseia na geração de uma lista da enumeração computável de todas as descrições $d_1, d_2, d_3, \ldots$ de todas as linguagens L_i de uma classe $\mathbb{S}$, de modo que cada linguagem da classe tem pelo menos uma ou mais descrições dentro da enumeração.

Dada uma coleção T de textos para essa linguagem, o método irá vasculhar a lista de descrições com o intuito de encontrar a primeira descrição d_i compatível com os exemplos dados (a verificação dessa compatibilidade também deve ser computável), transformando a descrição d_i em um elemento de Φ. O critério de convergência para esse método se baseia em duas condições. Primeiro, uma hipótese correta é sempre compatível com os exemplos dados. A segunda condição é a de que uma hipótese incorreta é incompatível com uma coleção suficientemente grande de exemplos para todos os textos dados[1]. A Fig. 2 mostra uma ilustração que representa a dinâmica desse método.

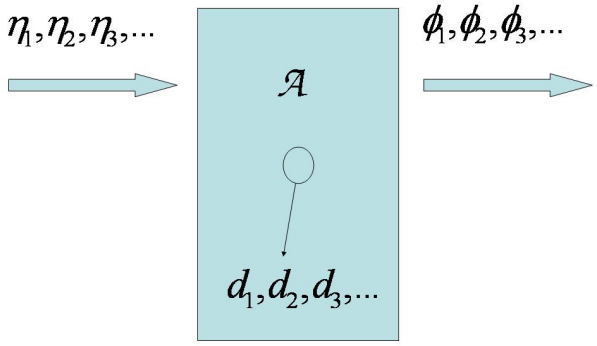


Figura 2: Método de identificação por enumeração. Uma nova hipótese escolhida é mantida enquanto não entrar em conflito com os exemplos.

Apesar de ser genérica e poderosa o bastante, a identificação por enumeração pode se tornar impraticável dependendo do tamanho da classe de linguagens envolvida. Em função desse fato, vamos buscar uma proposta alternativa para a identificação por enumeração. Façamos a seguinte conjectura: suponhamos a existência de uma estrutura U. Para um dado conjunto Σ, a estrutura U possui a capacidade de automodificar-se, de forma que, para cada

modificação efetuada, ela gera um novo reconhecedor R_i para uma linguagem $L_i \subset \Sigma^*$ diferente, pertencente a uma determinada classe $\mathbb{S}$.

Vamos supor também que esta estrutura U possui internamente um componente associado, um módulo o qual chamaremos de ζ. Entre suas atribuições, este módulo possui a capacidade de "ler" os elementos $\eta_a \in T$ e induzir as modificações em U.

Neste ponto, poderíamos fazer a seguinte pergunta: a estrutura U é um dispositivo adaptativo? A resposta envolve a questão da leitura dos exemplos pelo módulo ζ. Se afirmarmos que o módulo induz as alterações na estrutura U em função do texto lido, então, pela própria definição, a resposta é afirmativa, de acordo com [8] a estrutura U é um dispositivo adaptativo.

Vamos fazer agora uma pausa para discutir um pouco mais acerca das hipóteses geradas por um aprendiz genérico $\mathcal{A}$. Como foi visto antes, independentemente do critério de aprendizagem, da classe de linguagens ou do critério de aprendizagem e convergência, temos as hipóteses geradas definidas como uma lista de sistemas programáveis fixados. Assim, utilizando a equivalência entre Autômatos Adaptativos e as Máquinas de Turing mencionadas em [9], podemos definir, sem perda de generalidade, o conjunto de todas as hipóteses geradas por um aprendiz como sendo a lista, potencialmente infinita, denominada $\Phi_A = (a_1, a_2, a_3, \ldots)$, composta por uma enumeração de Autômatos Adaptativos, ou seja: sem perda de generalidade, podemos utilizar os autômatos adaptativos como uma representação do nosso conjunto de hipóteses. Usaremos esse resultado adiante.

De volta à descrição sobre a estrutura U, foi dito que, entre duas automodificações consecutivas, a estrutura U é um reconhecedor para uma linguagem específica. O conjunto de automodificações poderia ser arbitrário, mas convencionou-se que toda modificação de U resultaria em reconhecedores para linguagens diferentes, porém dentro de uma mesma classe. Se o elemento do texto não for reconhecido pela atual configuração da estrutura , o módulo ζ requisita uma modificação de U que "se adapta" de forma que a nova configuração identifica o exemplo apresentado como pertencente à linguagem. Quando o processo de adaptação se completa, o módulo ζ apresenta como saída o reconhecedor R adaptado para reconhecer todos os exemplos anteriormente apresentados, mais o último que foi responsável pela sua transformação. A Fig. 3 apresenta um diagrama de blocos para a estrutura U.

Já sabemos que U é um dispositivo adaptativo *por definição*. Poderíamos então perguntar: é possível reproduzir o comportamento de U utilizando puramente o modelo dos Autômatos Adaptativos? Caso não seja possível, quais novos elementos teríamos que incluir para obter uma solução que se comporte como a estrutura U ?

Os autômatos adaptativos são um modelo de computação [9]. A partir do momento em que apresentamos a opção da adaptatividade para a identificação por enumeração, é fundamental possuirmos um modelo formal, tanto para

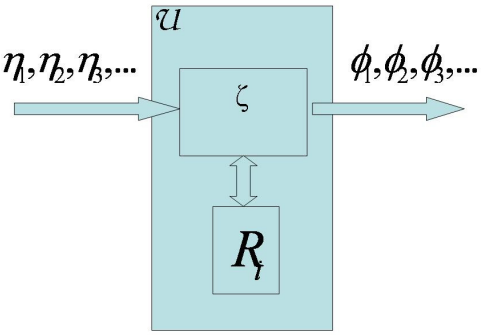


Figura 3: Diagrama de blocos da Estrutura U proposta

análise quanto para implementação.

Para respondermos se o modelo dos autômatos adaptativos pode ou não ser utilizado, vamos imaginar o seguinte cenário: um programador possui um arquivo em um determinado formato. Esse arquivo possui a seguinte série de *strings*:

1000000001
1100000011
1110000111
1111001111
1111111111

O programador é orientado pelo seu superior a desenvolver um reconhecedor para a linguagem representada pelas *strings*, utilizando para isso os Autômatos Adaptativos. O seu progresso será monitorado através de um repositório onde o programador guarda o código produzido. Caso a versão guardada lá funcione para todas as *strings* contidas no arquivo dado, então um novo conjunto de exemplos será fornecido em um novo arquivo (iremos admitir que o número de arquivos é finito). Para facilitar o trabalho do desenvolvedor, são dadas duas informações: todas as *strings* contidas nos arquivos são exemplos válidos da linguagem dada e a linguagem em questão é $L = 1^n 0^* 1^n$ com $n \geq 0$. Com essas informações, o programador parte para o desenvolvimento do reconhecedor. Obviamente, de posse dessas informações, ele irá conseguir projetar o Autômato Adaptativo para tratar dessa situação.

Porém, o que aconteceria se ele não recebesse a descrição da linguagem, de maneira a ter somente os exemplos dados no arquivo? Existiria uma maneira do programador descobrir a linguagem somente com base nos exemplos e assim construir o seu reconhecedor baseado em Autômatos Adaptativos ?

O programador inicialmente analisaria, de alguma forma, os exemplos fornecidos, já essa é a única informação que ele possui. Em algum momento, ele faria uma suposição: *"provavelmente*, a linguagem representada pelas string é: $L = ww^{-1}$."

Ele percebe, ao implementar o protótipo do reconhecedor, que essa suposição foi muito boa, pois todas as *strings* do arquivo foram processadas com sucesso. Assim, ele armazena seu reconhecedor no repositório e aguarda alguma ação por parte do seu superior. Quando recebe a indicação de que o repositório foi atualizado, o superior libera a segunda massa de exemplos. Ele não avalia o código gerado, pois essa não é a sua função. Cabe ao programador essa tarefa.

O segundo arquivo chega com os novos dados:

0000000000
00000000
0000
00
0

O programador fornece as *strings* ao seu reconhecedor, porém, durante o processsamento dos exemplos, a última *string* composta pelo símbolo "0" não é aceita. Caso ele não mude de suposição, ou seja, não implemente outro Autômato Adaptativo com um conjunto de funções adaptativas diferentes, seu programa não reconhecerá por completo todo o conjunto de novos exemplos dados.

Assim, com esse exemplo, ilustramos o seguinte fato: o modelo dos Autômatos Adaptativos por si só não pode alterar a linguagem para a qual ele está codificado para reconhecer. As funções adaptativas podem alterar o autômato subjacente de modo que este reconheça as cadeias das quais ele não possuía configurações de aceitação anteriormente, porém essa adaptação se dará apenas para as palavras de uma mesma linguagem fixada e codificada nas funções adaptativas. Mudar a linguagem aceita pelo Autômato Adaptativo implica, no mínimo, alterar o conjunto de funções adaptativas presente no autômato.

Assim, o programador do nosso exemplo teria que iniciar uma nova tentativa, mudando as funções adaptativas do seu autômato. A cada nova suposição, seu superior veria uma nova atualização no repositório, cada uma com sua data de criação. Ao final, quando o programador convergisse para linguagem representada pelas *strings*, seu superior teria uma enumeração de todas as versões de reconhecedores criadas.

VI. CONCLUSÃO

Com esse cenário, podemos responder a pergunta feita anteriormente: de fato, o modelo dos Autômatos Adaptativos não é o suficiente para modelarmos toda a estrutura U. Para ser mais específico, não podemos reduzir toda a estrutura U ao modelo dos Autômato Adaptativo.

Embora o módulo ζ da arquitetura não pertença ao modelo dos Autômatos Adaptativos, as instâncias R_i que são as funções reconhecedoras podem, sim, ser Autômatos Adaptativos. Dessa forma, as ações adaptativas em cada R_i também estariam sujeitas a um processo adaptativo, pois o módulo ζ teria a capacidade de alterar as funções adaptativas em cada R_i com o objetivo de gerar novas hipóteses.

Assim, temos um conceito novo com que trabalhar: a Adaptatividade de Segunda Ordem. Nela, temos dispositivos adaptativos que também estão sujeitos a sofrer automodificações. A adaptatividade de segunda ordem pode auxiliar na formação de linguagens onde a única informação disponível são os exemplos apresentados a um aprendiz. Na adaptatividade de segunda ordem, cada automodificação gera um Autômato Adaptativo para uma nova linguagem, com o intuito de adaptar-se aos exemplos apresentados e convergir para a linguagem representada por esses mesmos exemplos.

Com esse trabalho, determinamos em que nível de adaptatividade devemos trabalhar. As etapas seguintes irão concentrar-se em detalhar da arquitetura aqui sugerida e obter resultados sobre sua validação. Esse rumo implicará uma séria de questões a serem respondidas, referentes às classes de linguagens que a arquitetura pode tratar, suas limitaçõess e características.

VII. REFERÊNCIAS

[1] Angluin, D. e Smith, C. H. 1983. *Inductive Inference: Theory and ethods*. ACM Comput. Surv. 15, 1983.

[2] Arbib, M. *The Handbook of Brain Theory and Neural Networks*. MIT Press, 1997.

[3] Case,J. e Jain, S. e Manguelle, N. *Refinements of inductive inference by Popperian and reliable machines*. CJNM94, Kybernetika 30, pp. 12—52, 1994.

[4] Case, J., Lynes, C. Machine inductive inference and language identification. *Proceedings of the nineth International Colloquium on Automata, Languages and Programming*, Lecture Notes in Computer Science 140:107--115, 1982.

[5] Florencio, C.C. *Consistent identification in the limit of any of the classes k-valued is NP-hard*. In: Proceedings of the Conference on Logical Aspects of Computational Linguistics LACL, volume 2099 of LNCS/LNAI, Springer-Verlag, pp 125--138,2001.

[6] Gold. E. M. *Language identification in the limit*. Information and Control,10, pp 447--474, 1967.

[7] Lewis, H. and Papadimitriou,C. *Elements of the Theory of Computation*. Prentice-Hall,1997.

[8] Neto, J. J. *Adaptive Rule-Driven Devices - General Formulation and Case Study*. Lecture Notes in Computer Science. Implementation and Application of Automata 6th International Conference, CIAA 2001, Vol.2494, Pretoria, South Africa,Springer-Verlag, pp. 234--250, 2001.

[9] Rocha, R. L. A. e Neto, J. J. *Autômato adaptativo, limites e complexidade em comparação com máquina de Turing*. In: Proceedings of the second Congress of Logic Applied to Technology - LAPTEC 2000. São Paulo: Faculdade SENAC de Ciências Exatas e Tecnologia, pp. 33--48, 2001.

[10] Rozenberg G e Salomaa, A. editors. *Handbook of Formal Languages*. Berlin: Springer, 1997.

[11] Searle, J. R. *The mystery of consciousness*. New York: New York Review of Books, 1997.

[12] Stuart J. Russell e Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 1995.

[13] Zilles, S.On the Synthesis of Strategies Identifying Recursive Functions In: Proceedings of the 14th Annual Conference on Computational Learning theory and 5th European Conference on Computational Learning theory. Lecture Notes In Computer Science, vol. 2111. Springer-Verlag, London, pp 160-176, 2001.

[14] Yang,C. D.Universal Grammar, statistics or both?, Trends in Cognitive *Sciences*. Volume 8, Issue 10,pp 451-456, 2004.

Descoberta de padrões em bases de dados utilizando Técnicas Adaptativas

A. H. Tchemra, R. Camargo

***Resumo* – Este artigo tem como objetivo apresentar como os métodos da Tecnologia Adaptativa podem ser aplicados, como formas alternativas de solução, em ferramentas de apoio a decisão. O método proposto trata a Mineração de Dados Adaptativa, que consiste de um processo de descoberta de padrões e técnicas de mineração de dados agregadas ao conceito de adaptatividade. O estudo da aplicabilidade dessa proposta é desenvolvido a partir de uma base de dados organizada de forma multidimensional. Nesta aplicação, a tomada de decisão utiliza os padrões descobertos, que representam os critérios de um problema de decisão, como entrada numa tabela de decisão adaptativa baseada em métodos multicritério e técnicas adaptativas.**

***Palavras-chave*: tecnologia adaptativa, mineração de dados adaptativa, tabela de decisão adaptativa, tomada de decisão, múltiplos critérios, mineração de dados.**

I. INTRODUÇÃO

A área de Tecnologia Adaptativa pesquisa métodos adaptativos que permitem aos dispositivos adaptativos alterar suas estruturas internas e seus comportamentos de forma autônoma, de acordo com as necessidades requeridas durante o processo para obtenção da solução de determinados problemas [1].

O objetivo deste artigo é apresentar um estudo de um modelo de Mineração de Dados Adaptativa, como ferramenta para descoberta de padrões para auxiliar no processo de tomada de decisão.

Esse estudo trata a Mineração de Dados Adaptativa como um processo de identificação de tendências ou padrões em um conjunto de dados para orientar decisões a serem tomadas, por exemplo, dentro de atividades de planejamento estratégico.

Conforme [2], quando os padrões são definidos através de ferramentas de Mineração de Dados, é possível que as informações existentes e as inesperadas possam ser melhor investigadas.

De acordo com [2], Mineração de Dados investiga padrões ocultos existentes em grandes bases de dados. Essa ferramenta combina a identificação de padrões no conjunto de dados com tecnologias avançadas de aprendizagem computacional, envolvendo técnicas como inteligência artificial e estatística, descobrindo relações não explícitas. Os dados são, em geral, obtidos a partir de sistemas transacionais existentes, filtrados e armazenados em um depósito único de dados. Atualmente, observa-se que o avanço tecnológico e a redução de custo de armazenamento tornaram a investigação de dados corporativos uma realidade.

Conforme [1], as características da Tecnologia Adaptativa conferem aos seus métodos a classificação de sistemas inteligentes, permitindo desta forma um tratamento alternativo para a mineração de dados com a implementação de técnicas adaptativas, objetivando um estudo de suas características em relação aos métodos existentes.

Este artigo apresenta na seção II os conceitos principais sobre Mineração de Dados, seguida pela seção III que descreve o modelo de Mineração de Dados Adaptativa proposto. A seção IV mostra a forma como os resultados obtidos na Mineração de Dados Adaptativa podem ser utilizados em um processo de tomada de decisão, com o auxílio de uma tabela de decisão adaptativa. Finalmente, na seção V são apresentadas as conclusões e considerações sobre o modelo adaptativo proposto.

II. CONCEITO DE MINERAÇÃO DE DADOS

Os estudos sobre Mineração de Dados ou Data Mining tiveram origem em análise estatística na década de 60, que evoluíram, posteriormente nos anos 80, para novas técnicas de inteligência artificial, tais como lógica fuzzy, redes neurais, árvores de decisão [4].

Mineração de dados, segundo [5], consiste em um conjunto de técnicas utilizadas na exploração de conjuntos de dados, normalmente mantidos em tabelas, formando um banco de dados. A mineração de dados tem como objetivo o descobrimento de relacionamentos complexos envolvendo conceitos, tais como: padrões, regras, fatos em dados armazenados.

Conforme [2], o processo de descoberta de conhecimento e mineração de dados (KDD*, Knowledge Discovery and Data Mining*), pode ser tratado em quatro etapas:

1. Seleção de dados: etapa para determinar o agrupamento de dados e atributos de interesse;
2. Limpeza dos dados: consiste na remoção de ruídos, na transformação de alguns campos e na criação de campos combinados;
3. Mineração dos dados: aplicação de algoritmos específicos para extrair padrões de interesse;
4. Avaliação: etapa em que os padrões descobertos são disponibilizados para os usuários em forma inteligível, facilitando a visualização.

Deve-se ressaltar que o conceito de padrão, segundo

[2], é uma unidade de informação ou atributo de um registro que se repete, ou então é uma seqüência de informações/atributos presentes em uma estrutura que se repete.

Segundo [3], exitem cinco tipos de técnicas usadas para a Mineração de Dados:

1. Associações: que identificam afinidades entre um conjunto de dados em um grupo de registros; por exemplo: 72% de todos os registros que contêm itens A, B e C, também contêm itens D e E; dessa maneira, regras associativas procuram estabelecer ligações entre um elemento e outro;
2. Padrões Seqüenciais: que identificam seqüências de registros que ocorrem em decorrência de outros. Por exemplo: na ocorrência de um evento A, 32% dos clientes com determinadas características realizarão o evento B, em um determinado espaço de tempo;
3. Classificação: que divide as classes predefinidas, permitindo que registros de uma classe permaneçam próximos. Exemplificando: poderia haver classes de registros quanto à freqüência do comparecimento de clientes em uma agência bancária: infreqüentes (nunca freqüentam a agência), freqüentes (comparecem de modo freqüente) e ocasionais (ocasionalmente freqüentam a agência);
4. Agrupamento: a partir da base de dados, descobre classes ocultas, enquanto que a classificação já inicia com classes predefinidas;
5. Previsão: que tem como objetivo o cálculo de previsão do valor futuro de uma variável, como por exemplo, prever uma determinada projeção de vendas, considerando registros devidamente classificados.

III. PROPOSTA DE UM MODELO DE MINERAÇÃO DE DADOS ADAPTATIVA

As pesquisas sobre técnicas adaptativas e mineração de dados possibilitam o estudo de Mineração de Dados Adaptativa, que trata o processo de descoberta de padrões ocultos existentes em um determinado ambiente.

A proposta do modelo de Mineração de Dados Adaptativa é graficamente apresentada na figura 1.

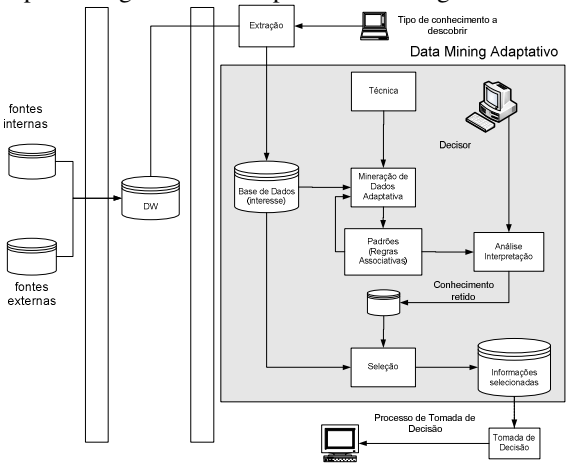


Figura 1 – Modelo de Mineração de Dados Adaptativa

O modelo proposto é composto pelas seguintes etapas principais:

1. definição do conhecimento a ser descoberto, que é fornecido por um decisor[2];
2. a partir de uma base de dados organizada e da definição do padrão a ser descoberto, é feita a extração dos registros que satisfazem a definição solicitada;
3. é aplicada a mineração de dados para a descoberta de padrões, por exemplo, baseada em regras associativas, usando técnicas adaptativas; os resultados desta etapa resultam nos atributos associados, que formam o padrão descoberto;
4. a análise dos padrões obtidos, caso sejam validados pelo decisor, representam as regras associativas, que interpretadas são retidas como conhecimento adquirido;
5. o conhecimento retido, formado pelo conjunto de regras associativas, é usado para gerar uma base de dados com as informações relacionadas com a busca do conhecimento desejado;
6. a base de dados com as informações selecionadas pode ser utilizada, por exemplo, em um processo de tomada de decisão.

O modelo de Mineração de Dados Adaptativa (DMA) proposto é definido formalmente com o emprego da sêxtupla:

DMA = (BD, TC, TDM, PD, BC, FA), cujos elementos são descritos a seguir:

- BD é a base de dados composta pelos registros de interesse para a mineração de dados;
- TC é o tipo de conhecimento a ser descoberto;
- TDM é a técnica de mineração de dados a ser utilizada;
- PD é o conjunto de padrões descoberto;
- BC é a base de conhecimento resultante;
- FA representa o conjunto de funções adaptativas que forma a camada adaptativa do modelo.

A etapa de mineração de dados adaptativa consiste, portanto, na localização de padrões PD_i através de técnicas específicas TDM, como a de regras associativas, que através de funções adaptativas FA_j permitem gerar novas ocorrências de padrões. A aplicação das funções adaptativas quando instanciadas, executam as ações adaptativas de inserção, remoção e consulta no conjunto de regras, modificando de forma autôma o próprio conjunto de regras.

O conhecimento em BC está codificado só no conjunto de regras. Uma transação adaptativa composta pelo conjunto de ações adaptativas leva a uma próxima situação, com uma configuração consistente, e incorpora uma nova informação no conjunto de regras.

[2]Neste estudo, decisor dever ser entendido como um especialista em determinado assunto, podendo ser um gestor, gerente, diretor ou até mesmo um grupo de profissionais.

IV. INTEGRAÇÃO DA MINERAÇÃO DE DADOS ADAPTATIVA COM A TABELA DE DECISÃO ADAPTATIVA

Como exemplo de aplicação dos resultados obtidos no modelo proposto de Mineração de Dados Adaptativa, descreve-se nesta seção, de forma resumida, uma maneira de se integrar duas ferramentas adaptativas.

A figura 2 mostra como os resultados da Mineração de Dados Adaptativa podem servir a um processo de decisão com o auxílio de uma Tabela de Decisão Adaptativa para um determinado problema de tomada de decisão.

A Mineração de Dados Adaptativa apresenta ao final do seu processo um conjunto de regras associativas, que representam o conhecimento adquirido a ser utilizado em um processo de tomada de decisão.

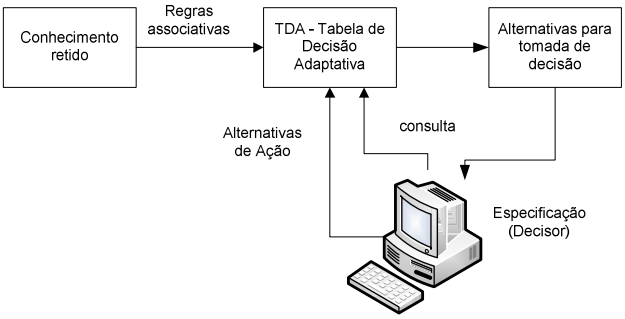


Figura 2 – Integração: Mineração de Dados Adaptativa e Tabela de Decisão Adaptativa

A tabela de decisão adaptativa, cujo formalismo se encontra em [6], pode apoiar nesse processo de tomada de decisão. A tabela de decisão pode ser gerada a partir das regras associativas, representando a entrada das regras iniciais R_k da tabela. Cada regra descreve o conhecimento descoberto na mineração de dados e, portanto, é composta pelo padrão encontrado. Esse padrão é composto por atributos, que representam os critérios C_i da tabela de decisão adaptativa.

Para completar os dados da tabela de decisão, o decisor deve inserir as alternativas de ação A_j para a tomada de decisão, associando-as às regras iniciais da tabela.

A tabela de decisão adaptativa, através de um método multicritério específico e da definição de funções adaptativas FAD_l relacionadas com as regras da tabela, está pronta para ser usada no processo de tomada de decisão.

Diante de um problema de decisão, consultas à tabela de decisão podem ser realizadas pelo decisor. A consulta tem como entrada uma regra RC, com atributos específicos, a ser pesquisada na tabela com o objetivo de buscar a solução ou alternativa de ação para a regra.

Uma ação adaptativa de consulta ao conjunto de regras da tabela é executada e caso os parâmetros da regra de entrada sejam coincidentes com alguma regra da tabela, então a alternativa de solução é apresentada.

Caso a regra não seja encontrada, uma ação adaptativa de inclusão de regra é executada, determinando qual é a alternativa de ação possível para a consulta. Uma nova configuração da tabela de decisão é apresentada, após uma ação adaptativa de exclusão de regras que elimina regras conflitantes ou redundantes.

A nova configuração da tabela de decisão adaptativa resultante tem incorporado novas regras, podendo desta forma ser utilizada para novas consultas.

V. CONCLUSÕES

Neste artigo são apresentados os mecanismos de Mineração de Dados Adaptativa, originado em informações existentes em um determinado banco de dados, que contém registros de transações obtidos em fontes internas ou externas em um ambiente.

Esse dispositivo em estudo propõe um modelo genérico, podendo tratar diversas técnicas de mineração associadas às técnicas adaptativas, sendo sua aplicabilidade ampla, identificando padrões que geram conhecimento.

Para efeito de elaboração da proposta, esse estudo aborda o conceito Mineração de Dados e Tecnologia Adaptativa, que integrada a Tabela de Decisão Adaptativa, pode apoiar no processo de tomada de decisão.

Deve-se observar que essa proposta está em fase de estudo e de especificação.

O objetivo desta proposta é trazer contribuições, tanto nos aspectos teóricos e práticos de aplicação da Tecnologia Adaptativa quanto nos processos de Mineração de Dados na descoberta de padrões e tomada de decisão.

REFERÊNCIAS BIBLIOGRÁFICAS

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

[2] RAMAKRISHNAN, R. *Sistema de Gerenciamento de Banco de Dados.* 3ª Ed. McGraw-Hill, São Paulo, 2008.

[3] Watson, R. *Data Management: Banco de Dados e Organizações*. 3ª Ed. LTC, Rio de Janeiro 2004.

[4] Kimbal, R., *The Data Warehouse Toolkit*. 2 ª Ed.Wiley Computer Publishing, USA, 2002.

[5] Moxon, B (1998). *Defining Data Mining-DBMS, Data Warehouse Supplement*, Aug.1996. HTTP//:dbmsmag.com/9608d53.html (30.Out.2008).

[6] TCHEMRA, A. H. *Aplicação da Tecnologia Adaptativa em Sistemas de Tomada de Decisão*. I WTA – Workshop sobre Tecnologia Adaptativa, Jan 2007 - www.pcs.usp.br/~lta.

Angela Hum Tchemra é bacharel e licenciada em Matemática pela Universidade Presbiteriana Mackenzie e mestre em Administração de Empresas pela mesma universidade. Atualmente é professora de cursos de graduação na Universidade Presbiteriana Mackenzie, Faculdade de Engenharia São Paulo e Fundação Armando Álvares Penteado. É Doutoranda do Departamento de Engenharia de Computação e Sistemas Digitais da Escola Politécnica da Universidade de São Paulo.

Rubens de Camargo é bacharel em Administração de Empresas, com especialização em Análise de Sistemas pela Faculdade Associadas de São Paulo e mestre em Administração de Empresas pela Universidade Presbiteriana Mackenzie. Atualmente é professor de cursos de graduação na Universidade Presbiteriana Mackenzie. É Doutorando do Departamento de Engenharia de Computação e Sistemas Digitais da Escola Politécnica da Universidade de São Paulo.

SHORT PAPERS

Uma proposta de autômato adaptativo para reconhecimento de anáforas pronominais segundo algoritmo de Mitkov

Djalma Padovani

***Resumo*— A Extração de Informações é uma área do processamento da linguagem natural que apresenta diversos desafios e um deles é a identificação de termos anafóricos ou co-referentes. Existem diversas propostas para resolução do problema e uma delas, chamada algoritmo de Mitkov, poderia ser implementada de uma forma mais eficiente por meio de tecnologias adaptativas, trazendo uma contribuição importante para esta área, pois existem poucos trabalhos propondo o uso de tecnologias adaptativas para o processamento da linguagem natural. Este trabalho propõe um autômato adaptativo que implementa o algoritmo de reconhecimento de anáforas pronominais de Mitkov.**

Palavras Chave*— *Autômatos Adaptativos, Processamento de Linguagem Natural, Reconhecedores Sintáticos, Gramáticas Livre de Contexto

I. INTRODUÇÃO

A Extração de Informações é uma área do processamento da linguagem natural que apresenta diversos desafios e um deles é a identificação de termos anafóricos ou co-referentes. Existem diversas propostas para resolução do problema e uma delas, chamada algoritmo de Mitkov, poderia ser implementada de uma forma mais eficiente por meio de tecnologias adaptativas, trazendo uma contribuição importante para esta área, pois existem poucos trabalhos propondo o uso de tecnologias adaptativas para o processamento da linguagem natural. Este trabalho propõe um autômato adaptativo que implementa o algoritmo de reconhecimento de anáforas pronominais de Mitkov.

VI. O PROBLEMA DA ANÁFORA

Anáfora é um fenômeno lingüístico no qual uma informação anteriormente introduzida é referenciada posteriormente em outra frase [1]. Por exemplo na frase "O João ama a Maria, mas ela não o ama", o sintagma nominal "O João" da primeira frase é referenciado na segunda frase através do pronome "o" (mesmo número, gênero e pessoa). O mesmo acontece com o sintagma "a Maria" e o pronome pessoal "ela" (mesmo gênero e número).

Formalmente, a expressão lingüística que introduz a anáfora na frase é denominada expressão anafórica. A informação previamente introduzida, que deve ser ligada à expressão anafórica, é denominada antecedente e o processo pelo qual é identificado o antecedente de uma expressão anafórica é denominado resolução anafórica ou resolução de anáforas.

As expressões anafóricas podem ser de diferentes tipos, tais como [2]:

- Pronominais: são aquelas nas quais o termo anafórico é um pronome. Por exemplo, na frase "O menino leu o livro, mas ele não gostou", o pronome "ele" é uma anáfora pronominal .
- Definidas: são compostas por um substantivo antecedido por um artigo definido. Por exemplo, na frase "Comprei uma casa. A casa fica longe daqui", o artigo "a" e o substantivo "casa" constituem uma anáfora definida.
- Demonstrativas: são compostas por um substantivo antecedido de um pronome demonstrativo. Por exemplo na frase "Comprei uma casa. Esta casa será sempre minha", o pronome "esta" e o substantivo "casa" constituem uma anáfora demonstrativa.

Em [3] são apresentadas diferentes técnicas de resolução de anáforas pronominais: algoritmo de Hobbs, algoritmo de Dagan e Itai, algoritmo de Lapin e Leass, o algoritmo de Palomar e o algoritmo de Mitkov.

O algoritmo de Mitkov tem por objetivo resolver anáforas pronominais cujos antecedentes são sintagmas nominais. A abordagem usada por Mitkov evita análises semânticas e sintáticas complexas e utiliza como método fundamental de resolução uma lista de heurísticas denominadas indicadores de antecedentes.

A abordagem proposta por Mitkov realiza os seguintes passos [3]:

- Examina a sentença corrente e as duas sentenças precedentes (se existirem) à anáfora em busca de sintagmas nominais (SN).
- Dentre os SNs encontrados seleciona somente aqueles que concordam em gênero e número com a anáfora e os agrupa em um conjunto de candidatos a antecedentes potenciais.
- Os SNs deste conjunto são pontuados por indicadores de antecedentes e posteriormente é realizada a soma dos pontos.

O SN escolhido como antecedente da anáfora será aquele com maior soma resultante das pontuações destes indicadores.

Os indicadores de antecedentes podem ser:

- PSN – Primeiro Sintagma Nominal. O primeiro sintagma nominal de cada sentença recebe 1 ponto.
- VI – Verbos Indicativos. Os sintagmas nominais seguidos de um verbo pertencente a um conjunto pré-definido (de maior saliência) recebem 1 ponto.
- RL – Reiteração Lexical. Sintagmas nominais que se repetem 2 ou mais vezes na mesma sentença na qual aparece o pronome recebem 2 pontos e os que se repetem apenas 1 vez recebem 1 ponto.
- PSTS – Preferência por SN em Títulos de Seção. Se

o sintagma nominal aparecer no título da seção onde o pronome aparece, então ele recebe 1 ponto.

- PC – Padrões de Colocação. Sintagmas nominais que seguem os mesmos padrões dos pronomes anafóricos podem ser candidatos a antecedentes, recebendo 2 pontos. Por exemplo, <Verbo>+<SN>... <Verbo><Pronome>.
- RI – Referência Imediata. Sintagmas nominais do tipo <V1>+<SN>...<V2>+<SN>...<Vn>+<SN>, onde V1, V2 e Vn são verbos, recebem 2 pontos.
- IS - Instruções Seqüenciais. O sintagmas nominais SN1 que aparecem no padrão <Para>+ <V1>+SN1>...<V2>+<SN2>...<V3>+<SN3>, onde V1, V2 e Vn são verbos, recebem 2 pontos.
- TP - Termo Preferencial. Os sintagmas nominais classificados como representativos do gênero textual recebem 1 ponto.
- SNI - Sintagma Nominal Indefinido. Os sintagmas nominais compostos por artigos ou pronomes indefinidos são punidos e recebem 1 ponto negativo.
- SNP - Sintagma Nominal Preposicionado. Os sintagmas nominais compostos por preposições também são punidos e recebem 1 ponto negativo.
- DR - Distância Referencial. Sintagmas nominais podem ser promovidos de acordo com a distância entre eles e a anáfora. Sintagmas nominais presentes na mesma sentença da anáfora recebem 2 pontos; sintagmas nominais presentes na sentença anterior à da anáfora recebem 1 ponto e sintagmas nominais presentes em 2 sentenças anteriores à anáfora recebem 0 pontos.

De maneira geral, as técnicas de Extração de Informações usam heurísticas e expressões regulares para a seleção de elementos semânticos do texto tais como expressões anafóricas[4][5]. Autômatos adaptativos poderiam ser usados de uma forma mais eficiente pois dispõem de recursos para a implementação de regras semânticas durante o processo de reconhecimento sintático.

II. AUTÔMATOS ADAPTATIVOS

Um autômato adaptativo é uma máquina de estados, inicialmente fixa, à qual são impostas sucessivas alterações durante a sua operação, resultantes da aplicação de ações adaptativas associadas às regras de transições realizadas pelo autômato. Desta maneira, estados e transições podem ser eliminados ou incorporados ao autômato em decorrência de cada um dos passos executados durante a análise da entrada [6].

O autômato adaptativo tem sua estrutura baseada no autômato de pilha estruturado, acrescido de ações adaptativas associadas às regras de transição. As ações adaptativas correspondem às chamadas de funções adaptativas e estas, por sua vez, são compostas de ações adaptativas elementares, declarações de variáveis e de geradores, e eventuais chamadas de funções adaptativas anteriores e posteriores à execução das ações adaptativas elementares [7].

O reconhecimento de uma cadeia ω pelo autômato adaptativo é caracterizado pela seqüência de reconhecimentos parciais de sucessivas sub-cadeias αi da cadeia ω pelas correspondentes máquinas de estados Ei ($0 \leq i \leq n$). Para reconhecer a cadeia ω o autômato terá percorrido, uma trajetória de reconhecimento correspondente à seqüência (E0,α0), (E1,α1), ... , (En,αn) , com $\omega = \alpha 0\ \alpha 1 \ldots\ \alpha n$, onde (Ei,αi) representa o consumo da sub-cadeia αi pela máquina de estados Ei [6].

III. MODELO PROPOSTO

O autômato adaptativo pode ser usado para implementar o algoritmo de Litkov de uma maneira eficiente pois, à exceção das chamadas e dos retornos das funções adaptativas, ele se comporta como um autômato finito.

O projeto de tal autômato deve levar em conta as características próprias do algoritmo de Mitkov. Em primeiro lugar, o algoritmo identifica o sintagma candidato por meio de pontuações obtidas com a aplicação de heurísticas, além de estados de aceitação. O autômato precisa analisar a cadeia, identificar os sintagmas nominais e pontuá-los de acordo com as regras do algoritmo, repetindo o processo até encontrar um pronome ou a cadeia terminar. Quando o autômato encontrar um pronome, ele precisa recuperar o sintagma nominal de maior pontuação e identificá-lo como candidato a antecedente.

Outra característica importante do projeto é a necessidade de ordenação dos sintagmas de acordo com a pontuação obtida na classificação. O autômato deve incorporar mecanismos de ordenação que facilitem a identificação do sintagma candidato ao final da análise da entrada, evitando, com isso, perda de desempenho com novas buscas. Uma lista encadeada ponderada pelos pontos acumulados, por exemplo, pode ser pensada como uma maneira mais rápida de identificar o sintagma candidato, sem que haja necessidade de retornar o processamento.

Por outro lado, é necessário preparar o léxico para que possa ser usado pelo autômato, identificando os *tokens* que compõem o texto e associando-os aos componentes morfológicos usados pelo algoritmo de Litkov: sintagmas nominais, sintagmas nominais indeterminados, sintagmas nominais preposicionados e sintagmas verbais. Se o *token* analisado é um substantivo, é necessário também identificar se ele está presente no título, se é um termo preferencial, a sentença em que aparece, e seu gênero (masculino ou feminino) e número (singular ou plural). Já se o *token* analisado é um verbo, é necessário identificar se ele faz parte do conjunto de verbos indicativos. Por fim, é necessário criar uma gramática que possa ser usada pelo autômato para reconhecer as sentenças da linguagem analisada. As seções seguintes descrevem os elementos usados no projeto do autômato.

A. Gramática e Léxico

A Fig. 1 apresenta uma gramática simplificada proposta para identificar os elementos usados pelo algoritmo de Litkov. (Uma gramática completa da Língua Portuguesa do Brasil pode ser encontrada em [8]).

A gramática é composta do conjunto de não-terminais

(S, SN, SV, Sind, Sprep, ind, prep, pro, N , V, D), onde:

S – Raiz
SN – Sintagma Nominal
SV – Sintagma Verbal
Sind – Sintagma indefinido
Sprep – Sintagma preposicionado
ind – elemento indefinido do SN
prep – elemento preposicional do SN
pro – elemento pronominal
N – núcleo do sintagma nominal
V – núcleo do sintagma verbal
D – demais elementos da Língua Portuguesa

S ➔ SN SV
SN ➔ Sind | Sprep | N | D
SV ➔ V SN | V
Sind ➔ ind N
Sprep ➔ prep N
ind ➔ ∀ α ϵ (artigos indefinidos)
prep ➔ ∀ β ϵ (preposições|contrações)
pro ➔ ∀ γ ϵ (pronomes)
N ➔ ∀ π ϵ (substantivos)
V➔ ∀ ξ ϵ (verbos)
D ➔ ∀ δ ℰ (artigos indefinidos | preposições |contrações | pronomes| substantivos| verbos)

Fig. 1. Gramática proposta.

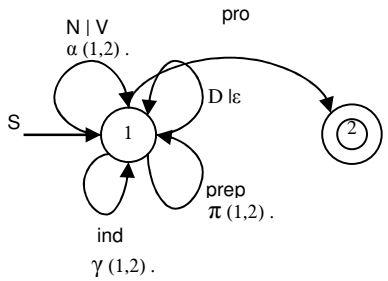
e dos terminais α ϵ (artigos indefinidos), β ϵ (preposições | contrações), π ϵ (pronomes), ξ ϵ (substantivos) , φ ϵ (verbos) e δ ℰ (artigos indefinidos | preposições | contrações | pronomes | substantivos | verbos). O espaçamento simples é usado como separador de *tokens* e o ponto é usado como separador de sentenças.

No caso de substantivos, o léxico é acrescido com a indicação do gênero, número, termo preferencial e presença no título da seção. Por exemplo, o substantivo "flúor" recebe a classificação <N\masc\sing\tp\psts>. No caso de verbos, o léxico é acrescido da identificação de verbo indicativo. Por exemplo, o verbo "analisar" recebe a classificação <V\VerboC>.

B. Autômato Adaptativo

A Fig. 2 apresenta o autômato adaptativo que implementa o algoritmo de Mitkov. São considerados os indicadores de antecedentes PSN, VI, RL, PSTS, TP, SNI, SNP e DR.

O autômato proposto lê cada átomo de entrada S até encontrar o primeiro espaçamento, montando um *token*. Em seguida, ele procura pelo não-terminal do *token* na lista de classificação léxica e, caso o encontre, realiza a transição para o estado correspondente. Caso não encontre o não terminal, ele realiza uma transição em vazio e continua a leitura da cadeia.



Produções Iniciais:

{(1, D |ε) :⟶ 1 }
{(1, N|V) , α (1,2) :⟶ 1}
{(1, ind), γ (1,2) :⟶ 1}
{(1, prep), π (1,2) :⟶ 1}
{(1, pro) :⟶ 2 }

Funções Adaptativas:

α (j,k): { o* :
- [(j, N|V), α (j,k) :⟶ j]
- [(j, ind), γ (j,k) :⟶ j]
- [(j, prep), π (j,k) :⟶ j]
+ [(o, '.') :⟶ j]
+ [(j, N) :⟶ o, □ (o,k)]
+ [(j, V) :⟶ o, □ (o,k)]}

□ (j,k): { t*u*:
+ [(j, ind) :⟶ t]
+ [(j, V|N|D| ε) :⟶ j]
+ [(j, pro) :⟶ k]
+ [(t, D| ε) :⟶ t]
+ [(t, N) :⟶ j]
+ [(j, prep) :⟶ u]
+ [(u, D| ε) :⟶ u]
+ [(u, N) :⟶ j]}

γ (j,l): { p*r* :
- [(j, N|V), α (j,l) :⟶ j]
- [(j, ind), γ (j,l) :⟶ j]
- [(j, prep), π (j,l) :⟶ j]
+ [(j, ind) :⟶ p]
+ [(p, D| ε) :⟶ p]
+ [(p, N) :⟶ r, δ (j,r,l,p)]}

δ (j, r,l,p): {v*
+
[(r,'.'): ⟶ j]
+ [(r, pro) :⟶ l]
+ [(r, ind) :⟶ p]
+ [(r, V|N|D| ε) :⟶ r]
+ [(r,prep) :⟶ v]
+ [(v, D| ε) :⟶ v]
+ [(v, N) :⟶ r]
+ [(j, V) :⟶ r]
+ [(j,prep): ⟶ v]}

π (j,l): { q*s* :
- [(j, N|V), α (j,l) :⟶ j]
- [(j, ind), γ (j,l) :⟶ j]
- [(j, prep), π (j,l) :⟶ j]
+ [(j, prep) :⟶ q]
+ [(q, D| ε) :⟶ q]
+ [(q, N) :⟶ s, ξ (j, s,l,q)]}

ξ (j, s,l,q): {x*
+ [(s, '.') :⟶ j]
+ [(s, pro) :⟶ l]
+ [(s, prep) :⟶ q]
+ [(s, V|N|D| ε) :⟶ s]
+ [(s, ind) :⟶ x]
+ [(x, D| ε) :⟶ x]
+ [(x, N) :⟶ s]
+ [(j, V) :⟶ s]
+ [(j,ind): ⟶ x]}

Fig. 2. Autômato adaptativo para o algoritmo de Mitkov

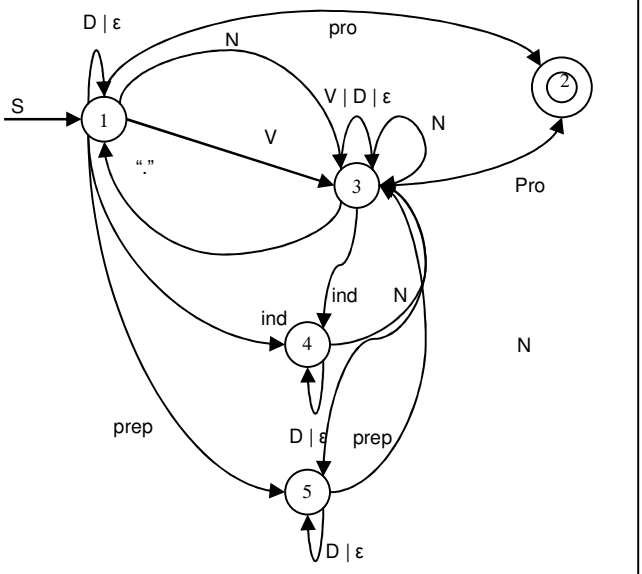
O processo se repete até que o autômato encontre um pronome ou finalize a leitura da cadeia. O autômato é construído dinamicamente, de acordo com o não-terminal identificado no estado 1.

A função adaptativa α (j,k) é chamada pelo autômato no estado 1 antes de processar um *token* N ou V, criando o estado 3 e as produções que levam o autômato do estado 1 ao novo estado e deste ao estado 1. Após a criação do

novo estado, o autômato chama a função (j,k), criando o estado 4 e o estado 5, e as produções que interligam o estado 3 aos estados 2, 3, 4 e 5.

A função adaptativa γ (j,l) é chamada pelo autômato no estado 1, antes de processar o *token* ind, criando o estado 4 e o estado 3, e as produções que levam o autômato do estado 1 ao estado 4. Em seguida, o autômato chama a função δ (j, r,l,p), criando o estado 5 e as produções que interligam o estado 3 aos estados 1, 2, 3, 4 e 5.

A função adaptativa π (j,l) é chamada pelo autômato no estado 1, antes de processar o *token* prep, criando o estado 5 e o estado 3, e as produções que levam o autômato do estado 1 ao estado 5. Em seguida, o autômato chama a função ξ (j,s,l,q), criando o estado 4 e as produções que interligam o estado 3 aos estados 1, 2, 3, 4 e 5.



Produções Finais:

(1, D |ε) :⟶ 1
(1, V) :⟶ 3
(1, N) :⟶ 3
(1, ind) :⟶ 4
(1, prep) :⟶ 5
(1, pro) :⟶ 2
(3, pro) :⟶ 2
(3, V| D |N | ε) :⟶ 3
(3, ".") :⟶ 1
(3, ind) :⟶ 4
(4, N) :⟶ 3
(4, D | ε) :⟶ 4
(5, N) :⟶ 3
(5, D | ε) :⟶ 5

Fig. 3. Situação final do autômato.

Se o autômato identifica um pronome nos estados 1 ou 3, ele realiza uma transição para o estado de aceitação 2. Nota-se que o autômato estabiliza-se após a criação dos estados 3, 4 e 5 e das correspondentes transições, sendo capaz, a partir daí, de processar qualquer elemento da gramática proposta. A Fig. 3 apresenta a situação final do autômato.

A pontuação dos candidatos a termo antecedente é feita sempre que o autômato encontra um substantivo. Uma lista encadeada ordenada pela pontuação é atualizada e, quando o autômato encontra um pronome, o substantivo mais bem pontuado da lista de mesmo gênero e número do pronome é apresentado como termo candidato. Um exemplo conceitual de implementação é apresentado a seguir.

O algoritmo Identifica_Anáfora lê a cadeia de entrada e calcula os pontos do *token* por meio da rotina Calcula_Pontos. Se o *token* lido é um substantivo, Identifica_Anáfora chama a rotina Insere_Lista e atualiza uma lista ordenada ponderada pelos pontos do *token*. Identifica_Anáfora continua a leitura da cadeia e se em determinado momento o *token* lido é um pronome, Busca_Candidato_Lista é chamada, verificando os *tokens* que possuem o mesmo gênero e número do pronome e calculando a distância referencial entre estes elementos e o pronome. Busca_Candidato_Lista retorna o *token* candidato à instância de Identifica_Anáfora finalizando o processamento.

Identifica_Anáfora

1. [Lê e identifica *token* da cadeia de entrada]
   Termo = Lê_*Token* (cadeia)

   [Identifica termo do tipo D]
   Se Termo(tp) = D
   então Leia_Proximo

   [Identifica nova sentença]
   Se Termo = "."
   então sentença := sentença +1

   Termo(sa) := sentença

   [Identifica SNI e SNP]
   Se Termo(tp) = (ind | prep)
   então Redutor := True

   [Identifica SN]
   Se Termo(tp) = N
   então
   Chama Calcula_Pontos
   Insere_Lista (Termo(vl), Termo(gn), Termo(nm), Termo(sa), Termo(p))
   Se Redutor= True
   então Redutor :=False

   [Identifica Pronomes]
   Se Termo(tp) = pro
   então
   Candidato = Busca_Candidato_Lista ()
   Retorna Candidato

Calcula_Pontos:

1. [Verifica se SN é PSN]
   Se Lista = ε e Termo(tp) = N
   então Termo(p) := Termo(p) + 1

   [Verifica se SN é PSTS]
   Se Termo(tp) = N e N = PSTS
   então Termo(p) := Termo(p) +1

[Verifica se SN é TP]
Se Termo(tp) = N e N = TP
então Termo (p) := Termo (p) +1

[Verifica se SN é SNI]
Se Termo(tp) = N e Redutor = True
então Termo(p) := Termo(p) – 1
Redutor = False

[Verifica se SN é SNP]
Se Termo(tp) = N e Redutor = True
então Termo (p) := Termo (p) – 1
Redutor = False

2. [Verifica se LookAhead é VI]
Se LookAhead (SN) = V e V = VI
então Termo(p) := Termo(p) +1

Insere_Lista:

1. [Se a lista está vazia]
Lista(id) := primeiroID
Lista(vl) := Termo(vl)
Lista(gn) := Termo(gn)
Lista(nm) := Termo(nm)
Lista(sa) := Termo(sa)
Lista(p) := Termo(p)
Lista(lk) := null

2. [Se pontuação do termo for maior que primeiro registro da lista]
Se Termo(p) > Lista(p)
então
ListaI(id) := novo id
ListaI(vl) := Termo(vl)
ListaI(gn) := Termo(gn)
ListaI(nm) := Termo(nm)
ListaI(sa) := Termo(sa)
ListaI(p) := Termo(p)
ListaI(lk) := id do primeiro registro da lista

3. [Se pontuação do termo for menor que primeiro registro da lista]
Repita enquanto Termo(p) <= Lista(p)

Guarda(lk) := Lista(lk)
Lista(lk) := novo id
Termo(lk) := Guarda(lk)

ListaI(id) := novo id
ListaI(vl) := Termo(vl)
ListaI(gn) := Termo(gn)
ListaI(nm) := Termo(nm)
ListaI(sa) := Termo(sa)
ListaI(p) := Termo(p)
ListaI(lk) := Termo(lk)

Busca_Candidato_Lista:

1. [Varre lista em busca do candidato]
Repita enquanto Lista(p) > Candidato(p)]
[Verifica Gênero e Número]
Se Lista(gn) = Termo(gn) e Lista(nm) = Termo(nm)
então
[Verifica DR]
Se Lista(st) = Termo(st)
então Lista (p) := Lista (p) + 2
Senão
Se Lista(st) = Termo(st) – 1
então Lista (p) := Lista (p) + 1

[Identifica Candidato]
Candidato(vl) := Lista(vl)
Candidato(p) := Lista(p)
Retorna Candidato

Sendo:
Cadeia – cadeia analisada
Sentença – número da sentença analisada
Gênero – gênero do termo analisado
Número – número do termo analisado
SN – sintagma nominal
PSN – primeiro sintagma nominal
PSTS – preferência por sintagma em título de seção
TP – termo preferencial
SNP – sintagma preposicionado
SNI – sintagma indefinido
VI – verbo indicativo
LookAhead – função de leitura do termo seguinte
Lista – conjunto de atributos do elemento no qual a lista está posicionada
ListaI – conjunto de atributos do termo que será inserido na lista
Termo – conjunto de atributos que caracterizam o termo analisado
Candidato – conjunto de atributos que caracterizam o termo candidato a antecedente
id – identificador único
tp – tipo do *token*
vl – valor do *token*
gn – gênero
nm – número
sa – sentença
p – pontos
lk – ponteiro para o próximo elemento

VII. EXPERIMENTO

Foi realizado um experimento no qual a implementação conceitual descrita na seção anterior foi codificada. Foi utilizada a linguagem de programação Python[9], pois ela dispõe de recursos de processamento de expressões regulares (*Regular Expression* – RE), processamento de linguagem natural (*Natural Language Tool Kit* – NLTK) e autômatos finitos (*Determinist Finite Automon* – DFA) necessários para a realização do trabalho.

O programa foi testado com o texto 'O flúor fortifica o esmalte, uma espécie de capa protetora dos dentes. Com a difusão de seu uso, outro problema surgiu: a fluorose, o excesso de flúor no organismo. Afinal, a substância não

se encontra apenas na água e cremes dentais: ela também está presente em diversos alimentos' [3]. O objetivo foi identificar o termo ao qual o pronome 'ela' faz referência (substância).

Como a frase apresenta diversos substantivos (flúor, esmalte, espécie, capa, dentes, difusão, uso, problema, fluorose, substância e água), foi possível testar todas as heurísticas do algoritmo implementadas pelo autômato.

Os elementos da frase foram etiquetados manualmente de acordo com suas classificações morfológicas e com identificações usadas pelo algoritmo de Mitkov: verbos indicativos (VI), termos preferenciais (TP) e preferências por SN em títulos de seção (PSTS). Foram identificados substantivos, verbos, preposições, pronomes, contrações e artigos indefinidos. Os demais elementos foram classificados em um único identificador ('D'). Também foram identificados o gênero e o número dos substantivos e do pronome. A Tabela I apresenta os elementos da frase e suas respectivas etiquetas.

TABELA I
ELEMENTOS E ETIQUETAS

Elemento	Etiqueta	Elemento	Etiqueta
O	<D>	Excesso	<NN\\masc\\sing\ \ntp\\npsts>
flúor	<NN\\masc\\sing\ \tp\\psts>	De	<D>
fortifica	<V>	Flúor	<NN\\masc\\sing\ \tp\\psts>
O	<D>	No	<D>
esmalte	<NN\\masc\\sing\ \ntp\\npsts>	Organismo	<NN\\masc\\sing\ \ntp\\npsts>
uma	<ind>	Afinal	<D>
espécie	<NN\\fem\\sing\\ ntp\\npsts>	A	<D>
de	<D>	Substância	<NN\\fem\\sing\\ ntp\\npsts>
capa	<NN\\fem\\sing\\ ntp\\npsts>	Não	<D>
protetora	<D>	se	<D>
dos	<D>	encontra	<V>
dentes	<NN\\masc\\plur\ \ntp\\npsts>	apenas	<D>
Com	<D>	na	<prep>
a	<D>	água	<NN\\fem\\sing\\ ntp\\npsts>
difusão	<NN\\fem\\sing\\ ntp\\npsts>	e	<D>
de	<D>	cremes	<NN\\masc\\plur\\ ntp\\npsts>
seu	<D>	dentais	<D>
uso	<NN\\masc\\sing\ \ntp\\npsts>	ela	<pron\\fem\\sing>
outro	<D>	também	<D>
problema	<NN\\masc\\sing\ \ntp\\npsts>	está	<V>
surgiu	<V>	presente	<D>
a	<D>	em	<D>
fluorose	<NN\\fem\\sing\\ ntp\\npsts>	diversos	<D>
o	<D>	alimentos	<NN\\masc\\plur\\ ntp\\npsts>

O programa é composto de um analisador léxico, que identifica os tokens da frase, um gerador de autômatos, que cria os novos estados e transições, e de um classificador, que classifica os *tokens* de acordo com os critérios de pontuação descritos no algoritmo de Mitkov. A Fig.4 apresenta a situação inicial do autômato, o alfabeto utilizado e a função de transição. Nota-se que as transições não permitidas são identificadas na função de transição com a palavra "None'. Por exemplo, não existe estado para o autômato ir se estiver no estado 1 e ler um token do tipo 'NN' (substantivo).

```
This DFA has 2 states
States: [1, 2]
Alphabet: ['D', 'NN', 'V', 'VI', 'ind', 'prep', 'pron', '.']
Starting state: 1
Accepting states: [2]
Transition function:
        1       2
D       1       None
NN      None    None
V       None    None
VI      None    None
ind     None    None
prep    None    None
pron    2       None
.       None    None
Current state: 1
Currently accepting: False
```

Fig. 4. Situação inicial do autômato.

À medida que o programa lê os *tokens* da cadeia, ele usa o gerador de autômatos adaptativamente para criar novos estados e transições, que são realizadas em seguida. A Fig. 5 ilustra este comportamento passo a passo.

```
estado atual 1 token ['O', 'D']
estado atual 3 token ['fluor', 'NN', 'masc', 'sing', 'tp', 'psts']
estado atual 3 token ['fortifica', 'V']
estado atual 3 token ['o', 'D']
estado atual 3 token ['esmalte', 'NN', 'masc', 'sing', 'ntp', 'npsts']
estado atual 4 token ['uma', 'ind']
estado atual 3 token ['especie', 'NN', 'fem', 'sing', 'ntp', 'npsts']
estado atual 3 token ['de', 'D']
estado atual 3 token ['capa', 'NN', 'fem', 'sing', 'ntp', 'npsts']
estado atual 3 token ['protetora', 'D']
estado atual 3 token ['dos', 'D']
estado atual 3 token ['dentes', 'NN', 'masc', 'plur', 'ntp', 'npsts', '.']
estado atual 3 token ['Com', 'D']
estado atual 3 token ['a', 'D']
estado atual 3 token ['difusao', 'NN', 'fem', 'sing', 'ntp', 'npsts']
estado atual 3 token ['de', 'D']
estado atual 3 token ['seu', 'D']
estado atual 3 token ['uso', 'NN', 'masc', 'sing', 'ntp', 'npsts']
estado atual 3 token ['outro', 'D']
estado atual 3 token ['problema', 'NN', 'masc', 'sing', 'ntp', 'npsts']
estado atual 3 token ['surgiu', 'V']
estado atual 3 token ['a', 'D']
estado atual 3 token ['fluorose', 'NN', 'fem', 'sing', 'ntp', 'npsts']
estado atual 3 token ['o', 'D']
estado atual 3 token ['excesso', 'NN', 'masc', 'sing', 'ntp', 'npsts']
estado atual 3 token ['de', 'D']
estado atual 3 token ['fluor', 'NN', 'masc', 'sing', 'tp', 'psts']
estado atual 3 token ['no', 'D']
estado atual 3 token ['organismo', 'NN', 'masc', 'sing', 'ntp', 'npsts', '.']
estado atual 3 token ['Afinal', 'D']
estado atual 3 token ['a', 'D']
estado atual 3 token ['substancia', 'NN', 'fem', 'sing', 'ntp', 'npsts']
estado atual 3 token ['nao', 'D']
estado atual 3 token ['se', 'D']
estado atual 3 token ['encontra', 'V']
estado atual 3 token ['apenas', 'D']
estado atual 5 token ['na', 'prep']
estado atual 3 token ['agua', 'NN', 'fem', 'sing', 'ntp', 'npsts']
estado atual 3 token ['e', 'D']
estado atual 3 token ['cremes', 'NN', 'masc', 'plur', 'ntp', 'npsts']
estado atual 3 token ['dentais', 'D']
estado atual 2 token ['ela', 'pron', 'fem', 'sing']
```

Fig. 5. Processamento da frase de teste.

O autômato encontra-se inicialmente no estado 1. Ao

ler o *token* 'O', identificado como 'D', ele permanece no estado 1. Em seguida, ao ler o *token* 'flúor', o autômato cria os estados 3, 4 e 5 e realiza uma transição do estado 1 para o estado 3, no qual ele permanece até ler o *token* 'uma', quando realiza uma transição do estado 3 para o estado 4. Em seguida, ao ler o *token* 'espécie', o autômato volta ao estado 3, no qual permanece até ler a contração 'na', quando realiza uma transição para o estado 5. Em seguida, ao ler o *token* 'água', ele volta para o estado 3 e permanece nele até ler o *token* 'ela', quando realiza uma última transição para o estado de aceitação 2. A Fig.6 apresenta a situação final do autômato, o alfabeto utilizado e a nova função de transição após o processamento da frase. Nota-se que, neste momento, o autômato possui 5 estados e encontra-se no estado de aceitação 2.

```
This DFA has 5 states
States: [1, 2, 3, 4, 5]
Alphabet: ['D', 'NN', 'V', 'VI', 'ind', 'prep', 'pron', '.']
Starting state: 1
Accepting states: [2]
Transition function:
        1       2       3       4       5
D       1       None    3       4       5
NN      3       None    3       3       3
V       3       None    3       None    None
VI      3       None    3       None    None
ind     4       None    4       None    None
prep    5       None    5       None    None
pron    2       None    2       None    None
.       None    None    1       None    None
Current state: 2
Currently accepting: True
```

Fig. 6. Situação final do autômato.

A Fig.7 apresenta o resultado final gerado pelo programa: uma lista de substantivos e suas respectivas pontuações, e uma lista de termos candidatos a antecedente.

```
Lista de Sustantivos e Pontuações
[['fluor', 'NN', 'masc', 'sing', 'tp', 'psts'], [1, 2], 3]
[['cremes', 'NN', 'masc', 'plur', 'ntp', 'npsts'], [2, 1], 1]
[['substancia', 'NN', 'fem', 'sing', 'ntp', 'npsts'], [2, 1], 3]
[['organismo', 'NN', 'masc', 'sing', 'ntp', 'npsts', '.'], [2, 1], 1]
[['excesso', 'NN', 'masc', 'sing', 'ntp', 'npsts'], [1, 1], 1]
[['fluorose', 'NN', 'fem', 'sing', 'ntp', 'npsts'], [1, 1], 2]
[['problema', 'NN', 'masc', 'sing', 'ntp', 'npsts'], [1, 1], 1]
[['uso', 'NN', 'masc', 'sing', 'ntp', 'npsts'], [1, 1], 1]
[['difusao', 'NN', 'fem', 'sing', 'ntp', 'npsts'], [1, 1], 2]
[['dentes', 'NN', 'masc', 'plur', 'ntp', 'npsts', '.'], [1, 1], 1]
[['capa', 'NN', 'fem', 'sing', 'ntp', 'npsts'], [0, 1], 2]
[['esmalte', 'NN', 'masc', 'sing', 'ntp', 'npsts'], [0, 1], 1]
[['agua', 'NN', 'fem', 'sing', 'ntp', 'npsts'], [2, 1], 2]
[['especie', 'NN', 'fem', 'sing', 'ntp', 'npsts'], [0, 1], 1]

Candidatos
[['substancia', 'NN', 'fem', 'sing', 'ntp', 'npsts'], [2, 1], 3]
```

Fig. 7. Resultado final do processamento.

A lista de substantivos está ordenada pela pontuação obtida durante o processamento. Ao encontrar o primeiro pronome, o programa atualiza a pontuação de acordo com os critérios de distância referencial (DR) e reiteração lexical (RL), percorrendo a lista até o momento em que os pontos acrescidos já não influenciem mais no resultado final.

A lista de termos candidatos a antecedente apresenta corretamente o termo 'substância', o mais bem pontuado e de mesmo gênero e número da anáfora analisada, o pronome 'ela'.

VIII. CONCLUSÕES

Este artigo apresentou uma proposta de autômato adaptativo para reconhecimento de anáforas pronominais segundo o algoritmo de Mitkov. O modelo proposto necessita de uma preparação léxica e recomenda um substantivo candidato ao final do processamento. Procurou-se favorecer o desempenho na concepção do mecanismo de funcionamento, evitando-se passos redundantes durante o processamento. A conclusão final foi de que a tecnologia adaptativa pode ser mais eficiente do que as técnicas de Extração de Informações usualmente utilizadas [7][8], pois dispõe de recursos para a implementação de regras semânticas durante o processo de reconhecimento sintático ao invés da técnica de uso de expressões regulares ao final do processamento. Para trabalhos futuros, propõe-se que sejam realizados testes quantitativos, visando avaliar estatisticamente a eficiência do autômato e que seja avaliada a possibilidade de utilizá-lo também para resolução de outros tipos de anáforas que não as pronominais.

REFERÊNCIAS

[1] S.A. Freitas. "Interpretação Automatizada de Textos: Processamento de Anáforas", Tese de Doutorado, orientado por C.S.Menezes e J.G.P.Lopes, Centro Tecnológico da Universidade Federal do Espírito Santo, 2005.

[2] C. Gasperin , R. Goulart , R.Vieira. Uma Ferramenta para Resolução Automática de Co-referência, 2003. Programa Interdisciplinar de Pós-Graduação em Computação Aplicada, Unisinos, São Leopoldo, Brasil. Disponível em: http://www.rodrigo.goulart.nom.br/publicacoes/gasperin2003.pdf.

[3] A.R Chaves. "A Resolução de Anáforas Pronominais da Língua Portuguesa com Base no Algoritmo de Litkov", Dissertação de Mestrado, orientada por L.H.M. Rino, Centro de Ciências Exatas e de Tecnologia da Universidade Federal de São Carlos, 2007.

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

[5] E. Riloff and J. Lorenzen. Extraction-based text categorization: Generating Domain Specific role relationships automatically. In Natural Language Information Retrieval, Dordrecht, The Netherlands: Kluwer Academic Publishers, 1999. p. 167-196.

[6] J.J. Neto. "Contribuições à Metodologia de Construção de Compiladores", Tese de Livre Docência, Escola Politécnica da Universidade de São Paulo, 1993.

[7] C.Y.O.Taniwaki e J.J.Neto. "Autômatos Adaptativos no Tratamento Sintático de Linguagem Natural", 2001. Disponível em: <http://www.pcs.usp.br/~lta/artigos/taniwaki_bt2001.pdf>.

[8] C. Luft,. Moderna gramática brasileira, 2000. São Paulo: Globo.

[9] Python. Python Programming Language – Official Website. Disponível em: <http://www.python.org>.

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

Análise, Sob o Ponto de Vista de Adaptatividade, de Sistemas Híbridos em Inteligência Computacional: Algoritmos Genéticos & Sistemas Nebulosos (06 Novembro 2008)

M. A. P. Burdelis, M. T. C. Andrade

***Resumo*— Este trabalho apresenta uma análise de sistemas híbridos envolvendo Computação Evolutiva (CE) e Computação Nebulosa (ou Fuzzy) (CF), sob o ponto de vista de adaptatividade. Analisa também a utilização de Tabelas de Decisão Adaptativas para simular Algoritmos Genéticos. Além disso, uma visão do panorama atual de pesquisas por relações e transformações entre modelos descritos por três diferentes técnicas de Inteligência Computacional (IC) - Computação Nebulosa (ou "Fuzzy"), Computação Neural e Computação Evolutiva - é apresentada.**

***Palavras Chave*— Dispositivos Adaptativos; Algoritmos Genéticos; Sistemas Nebulosos; Sistemas Híbridos; Adaptatividade; Inteligência Computacional; Inteligência Artificial.**

I. INTRODUÇÃO

O Conceito de Adaptatividade, segundo [17], corresponde a: "a capacidade que tem um sistema de, sem a interferência de qualquer agente externo, tomar a decisão de modificar seu próprio comportamento, em resposta ao seu histórico de operação e aos dados de entrada". Ainda segundo [17], entende-se como "Tecnologia Adaptativa" (TA) a aplicação da adaptatividade com fins práticos. Este conceito, consideravelmente simples, possui uma vasta gama de aplicações, como por exemplo: o reconhecimento de padrões [6], síntese de voz [22], e até mesmo arte por computador [3], entre outros. Segundo [16], a Inteligência Artificial (IA) é um campo que pode ser fortemente beneficiado pela Tecnologia Adaptativa (TA), pois dispositivos adaptativos possuem mecanismos para adquirir, representar e manipular conhecimento.

A Inteligência Artificial (IA) corresponde a um ramo consideravelmente recente da ciência, e pode ser vista, segundo [7] como a ciência cujo objetivo é compreender e construir "entidades inteligentes" (conforme consenso entre diversos livros modernos). Para atingir este objetivo, existe uma vasta gama de técnicas e abordagens. Basicamente, dentro do ramo da ciência denominado "Inteligência Artificial", duas famílias de técnicas podem ser encontradas: a "Inteligência Artificial Clássica", e a "Inteligência Computacional".

A IA Clássica faz uso de quaisquer meios necessários na tentativa de projetar entidades inteligentes, sem se limitar aos métodos naturais inerentes aos seres vivos [9]. Sendo assim, pode fazer uso de técnicas como: métodos heurísticos, lógica simbólica, conhecimento encapsulado, etc. (técnicas não baseadas originalmente na inteligência dos seres biologicamente constituídos).

Recentemente, a Inteligência Computacional (IC) surgiu como um campo derivado do campo original de IA. Este ramo busca explorar o potencial de criação de soluções pela simulação de características dos seres inteligentes biologicamente constituídos. Segundo [8], a IC abrange os campos de Computação Neural (CN), Computação Evolutiva (CE), e Computação Nebulosa (ou Fuzzy) (CF) – as quais são áreas amplamente aceitas como componentes da IC, também segundo [9]. A IC também abrange outras técnicas, entre elas, segundo [8]: "Computação baseada em DNA", e "Computação Quântica".

A CF surgiu baseada na Teoria de Conjuntos Nebulosos lançada em 1965, com a publicação de [21]. Esta teoria tem por objetivo viabilizar a modelagem de sistemas vagos e mal-definidos, dos quais não se possui muitas informações precisas. Este tipo de modelagem é obtido tentando-se imitar o raciocínio humano, aproximado e sem precisão numérica.

A CN se baseia em conclusões do estudo dos tecidos celulares do cérebro de seres vivos, em particular nas estruturas de neurônios interconectados. Pela construção de modelos matemáticos, estruturalmente inspirados na estrutura de tecidos cerebrais, obtêm-se as Redes Neurais Artificiais (RNAs). Estas RNAs são entidades de processamento paralelo e distribuído, com capacidade de adaptação, e que podem ser utilizadas para diversas aplicações, como reconhecimento de padrões (por ex: reconhecimento de imagens); classificação de dados; bancos de dados; entre outros [12].

Finalmente, a CE se apóia na teoria da evolução das

O presente trabalho foi realizado com apoio do CNPq, Conselho Nacional de Desenvolvimento Científico e Tecnológico - Brasil.

M. A. P. Burdelis é mestrando da Escola Politécnica da Universidade de São Paulo, em Engenharia de Eletricidade, com area de concentração em Sistemas Digitais, e linha de pesquisa em Inteligência Artificial. E-mail: mauricio.burdelis@poli.usp.br

M. T. C. Andrade é professor livre-docente do Laboratório de Engenharia de Conhecimento (KNOMA), vinculado ao Departamento de Engenharia de Computação e Sistemas Digitais da Escola Politécnica da Universidade de São Paulo. E-mail: marco.andrade@poli.usp.br

espécies. Na própria natureza, um mecanismo de seleção de indivíduos faz com que sobrevivam apenas os seres vivos mais bem-adaptados ao meio ambiente (ou seja, "apenas os mais fortes sobrevivem"). Entre os indivíduos sobreviventes, ocorrem cruzamentos e mutações, que podem gerar novos indivíduos ainda mais aptos (ou menos aptos). Sobre a nova população, novas seleções acontecem, e pela repetição deste processo, obtém-se a evolução dos indivíduos. Na computação evolutiva, um problema a ser resolvido corresponde ao meio ambiente, e a população de indivíduos é, na realidade, uma população de soluções (candidatas à solução final do problema em questão). Algoritmos Genéticos (AGs) correspondem a uma abordagem específica de CE, proposta originalmente em [14].

Este artigo apresenta alguns sistemas híbridos envolvendo CE e CF, os quais vêm sendo construídos e estudados para a pesquisa por relações e transformações entre estas duas técnicas de IC, como parte de uma linha de pesquisa maior, proposta em [1] – a qual busca relações de equivalência entre modelos construídos utilizando-se cada uma das três técnicas: CF, CE e CN. Estes sistemas híbridos são analisados sob o ponto de vista de Adaptatividade.

II. TECNOLOGIA ADAPTATIVA

A Tecnologia Adaptativa (TA) corresponde a um conceito que envolve o estudo de técnicas, modelos e ferramentas, que possibilitam que um sistema computacional tenha a capacidade de, segundo [17]:

(1) modificar seu próprio comportamento;
(2) realizar esta modificação de forma autônoma;
(3) realizar esta modificação em resposta a:
a. seu histórico de operação; e
b. aos seus dados de entrada (ou seja: sua situação corrente e momentânea do ambiente).

Uma propriedade interessante de sistemas adaptativos, mencionada em [17], é que:

(4) duas instâncias diferentes do mesmo sistema adaptativo podem evoluir para comportamentos finais completamente diferentes, dependendo da diversidade de eventos a que forem submetidos em suas operações.

O conceito de adaptatividade se estende para qualquer dispositivo dirigido por regras. Em [16], encontra-se uma formalização geral desta extensão, descrevendo o conceito de "Dispositivos Adaptativos Dirigidos por Regras" (DADR).

Esta formalização afirma que dispositivos dirigidos por regras podem ser transformados em dispositivos adaptativos, adicionando-se ao sistema uma "camada adaptativa", que contém os elementos necessários para se associar a cada regra um conjunto de ações (a ser executado cada vez que a regra for acionada), responsáveis pela modificação do próprio conjunto de regras. Assim sendo, DADRs são compostos por duas partes: uma "camada adaptativa", responsável pelo comportamento adaptativo, e um "dispositivo subjacente", tipicamente não-adaptativo. É importante observar que, de acordo com esta formalização:

(5) um DADR possui como resultado de sua adaptação uma modificação em seu conjunto de regras.

Segundo [16], um dispositivo dirigido por regras é qualquer máquina formal cujo comportamento depende exclusivamente de um conjunto finito de regras (o qual mapeia cada configuração possível da máquina em sua próxima configuração correspondente). Um autômato, por exemplo, é um dispositivo que incorpora este recurso de forma mais natural, porém outros dispositivos também podem fazê-lo, uma vez que a formalização referida é bastante genérica, e não depende da natureza do formalismo não-adaptativo escolhido. Outros exemplos de dispositivos dirigidos por regras, (transformados em DADRs nas citações a seguir), são: Redes de Markov [3]; Árvores de Decisão [18], e Tabelas de Decisão [16] e [4].

A. *Tabelas de Decisão*

Tabelas de decisão são exemplos de dispositivos dirigidos por regras, muito interessantes para a aplicação do conceito de adaptatividade [16].

Estas tabelas codificam regras e ações. As colunas da tabela (exceto pelas duas primeiras) correspondem às regras, e as linhas da tabela (exceto pela primeira) são divididas em dois grupos: um correspondente a condições de regras, e outro correspondente a ações.

Para cada regra as células correspondentes às condições relevantes são marcadas, com um símbolo correspondente a "verdadeiro" (V), ou "falso" (F), indicando que cada condição correspondente deve ser verdadeira (ou falsa) para que a regra seja satisfeita. Células correspondentes a condições irrelevantes para a regra em questão são marcadas com o símbolo "– ". Células marcadas em linhas correspondentes a ações indicam se as respectivas ações devem ser executadas quando a regra é satisfeita (símbolo "V" - "verdadeiro"), ou não (símbolo "F" - "falso").

		regras				
		0	1	2	…	m
Linhas de Condições	C1	T	-	F	…	T
	C2	F	F	T	…	T
	…	…	…	…	…	…
	Cn	-	F	-	…	T
Linhas de Ações	A1	T	F	F	…	T
	A2	F	T	F	…	T
	…	…	…	…	…	…
	Ak	F	F	F	…	T

Fig. 1. Tabela de Decisão com "m" regras, "n" condições, e "k" ações

Segundo [16], a operação destas tabelas se realiza da seguinte maneira:

1. A situação do sistema é verificada, conforme as combinações de condições informadas em cada regra codificada na tabela.
2. Se nenhuma regra satisfaz a situação atual; então nenhuma ação é executada.
3. Se apenas uma regra satisfaz as condições atuais, então temos uma escolha determinística, e a regra em questão é selecionada e aplicada.

4. Se mais de uma regra satisfaz a situação atual, então temos uma situação não-determinística – conseqüentemente, todas estas regras são elegíveis de aplicação em paralelo.
5. A regra selecionada é então aplicada, executando-se o conjunto de todas as ações indicadas pelo valor booleano "verdadeiro" nas células da regra correspondentes a linhas de ações.
6. Uma vez que a regra em questão tenha sido aplicada, a tabela de decisão está preparada para ser utilizada novamente.

B. Tabelas de Decisão Adaptativas

Em [16], propõe-se a obtenção de Tabelas de Decisão Adaptativas, pela inclusão de linhas correspondentes a "ações adaptativas", ao final da tabela. Estas ações adaptativas são divididas em dois tipos:

- ações a serem executadas antes da aplicação da regra;
- ações a serem executadas após a aplicação das regras.

		regras				
		0	**1**	**2**	**...**	**m**
Condições	**C1**	T	-	F	...	T
	C2	F	F	T	...	T
	...	...	...	...	...	...
	Cn	-	F	-	...	T
Ações	**A1**	T	F	F	...	T
	A2	F	T	F	...	T
	...	...	...	...	...	...
	Ak	F	F	F	...	T
Ações Adaptativas ("antes")	**BA1**	T	F	F	...	T
	BA2	F	T	T	...	T
	...	...	...	...	...	...
	BAp	F	F	T	...	F
Ações Adaptativas ("depois")	**AA1**	T	F	F	...	T
	AA2	F	T	F	...	T
	...	...	...	...	...	...
	AAq	T	F	T	...	T

Fig. 2. Formato de Uma Tabela de Decisão Adaptativa, com "m" regras, "n" condições, "k" ações, "p" ações adaptativas (a serem executadas antes da aplicação de regras), e "q" ações adaptativas (a serem executadas após a aplicação de regras)

Em [16], e [4], afirma-se que estas ações adaptativas podem ser interpretadas como chamadas de funções adaptativas. Nestes casos, também são indicados na própria tabela:

· os parâmetros a serem passados para estas funções; e
· determinadas ações a serem executadas por estas funções (como a exclusão ou a inclusão de regras).

Em [4], esta interpretação foi utilizada para a construção de uma Tabela de Decisão Adaptativa, a qual substituiu um Algoritmo Genético (AG) utilizado para estudo de biodiversidade. Neste AG, os indivíduos da população correspondiam a regras, as quais foram mapeadas como colunas de uma tabela de decisão adaptativa. Segundo [4], esta substituição não acarretou em perda de desempenho do sistema, quanto ao número de iterações necessárias para se obter um indivíduo com uma aptidão mínima. No trabalho mencionado, as funções adaptativas correspondiam a operadores de mutação e de cruzamento de AGs. Maiores detalhes sobre este trabalho serão analisados no final deste documento, na seção "V".

III. TRANSFORMAÇÕES ENTRE DE DIFERENTES TÉCNICAS DE IC (MODELOS NEBULOSOS; MODELOS EVOLUTIVOS E MODELOS NEURAIS)

Dentre as diferentes técnicas que podem ser classificadas como "Inteligência Computacional", destacam-se três:

- Computação Nebulosa (ou "Fuzzy");
- Computação Neural;
- Computação Evolutiva.

Em [1], vislumbrou-se a possibilidade de estas três técnicas possuírem relações entre si, que permitissem (e regulassem) a transformação de modelos (de problemas reais), descritos em uma destas técnicas, em modelos (dos mesmos problemas) descritos em outra destas técnicas. Esta idéia deu origem a uma extensa pesquisa realizada no laboratório KNOMA da Escola Politécnica da USP.

Caso estas relações existam e sejam encontradas, duas implicações principais poderiam ocorrer.

Em primeiro lugar, a modelagem de problemas pela utilização da Inteligência Computacional passaria a ter maior flexibilidade, e conseqüentemente maior poder para tratar diferentes problemas. Em outras palavras: uma solução (para um determinado problema) poderá usufruir as vantagens da utilização de modelos descritos nas três diferentes técnicas, caso estas transformações existam e sejam empregadas convenientemente. Por exemplo: um modelo descrito pela técnica de Computação Nebulosa, que não apresentasse resultados satisfatórios para determinados casos particulares de um problema, poderia ser convertido para seu modelo equivalente em Computação Neural, o qual poderia apresentar resultados satisfatórios para estes casos particulares em questão.

Em segundo lugar, resultados afirmativos desta pesquisa podem abrir caminhos para uma nova pesquisa, para a descoberta da existência de uma possível técnica fundamental da IC, da qual as técnicas CF, CN e CE poderiam ser casos particulares. Caso este último passo venha a comprovar a existência de tal técnica, essa descoberta poderia mudar a maneira atual de se trabalhar com IC.

Para se verificar a equivalência entre estas três diferentes técnicas de Inteligência Computacional, busca-se empregar um conceito semelhante ao envolvido no "Diagrama Gajski-Kuhn", do trabalho [11], fazendo-se uma analogia entre seus eixos e as três técnicas de IC. Originalmente, este diagrama é uma ferramenta utilizada para a representação de sistemas digitais, e possui três eixos:

- Funcional: o que um determinado circuito realiza (sua função);
- Estrutural: apresenta quais são os componentes lógicos pelos quais o circuito pode ser descrito;
- Geométrica: a representação física (estrutura espacial) do circuito.

O centro deste diagrama representa o nível mais baixo de abstração, o qual aumenta conforme se afasta do centro (chegando à representação de um sistema completo). Os

pontos indicados no diagrama (denominados "pontos notáveis") representam pontos definidos dos sistemas digitais, e cada circunferência do diagrama representa um nível de abstração. Uma característica muito importante deste diagrama é a existência de operações que levam de um ponto notável a outro, não necessariamente contido no mesmo eixo do primeiro.

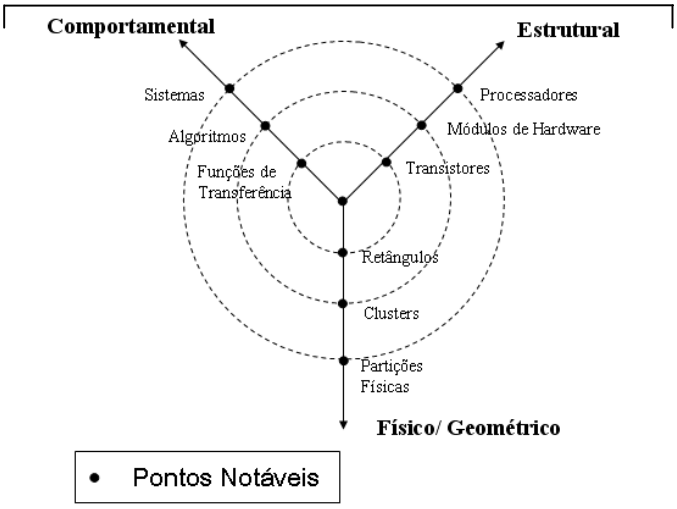


Fig. 3. O diagrama Gajski-Kuhn

Abstraindo-se o Diagrama de Gadjski-Kuhn para a Inteligência Computacional, pretende-se associar cada eixo a uma diferente técnica:

- Eixo Comportamental: associado à CF;
- Eixo Estrutural: associado a sistemas de CN;
- Eixo Geométrico: associado a técnicas de CE.

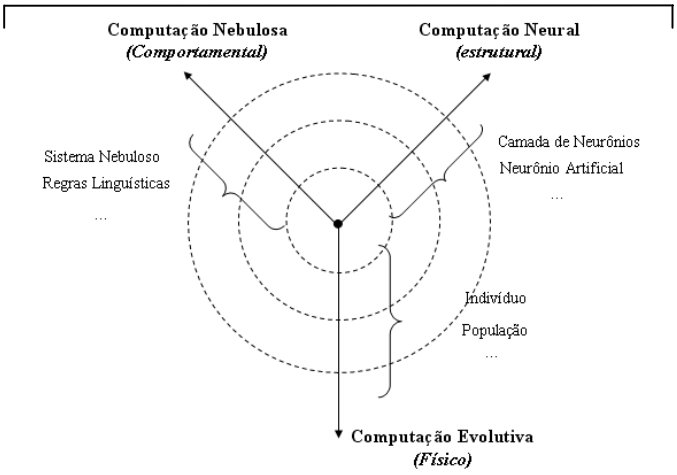


Fig. 4. O diagrama adaptado para a busca por relações

Desta forma, para se identificar relações entre a Computação Nebulosa e a Computação Neural, é necessário identificar pontos equivalentes aos pontos notáveis utilizados no Diagrama de Gadjski-Kuhn, e também identificar as operações que levam de um ponto a outro nos eixos.

Por se tratar de uma linha de pesquisa muito abrangente, envolvendo um trabalho muito vasto, esta foi dividida em diversos trabalhos menores e mais específicos, compreendendo dissertações de mestrado. Além disso, uma plataforma computacional foi desenvolvida no laboratório KNOMA da Escola Politécnica da USP, para experimentos computacionais de conversões de modelos. Até o momento, duas dissertações de mestrado foram geradas nesta linha de pesquisa: [5] buscando relações entre CF e CN, e [10] buscando conversões entre CF e CE. Outro trabalho está em andamento, também buscando relações entre CE e CF.

Em [5], foram pesquisadas relações/transformações de modelos descritos utilizando-se Computação Nebulosa, com/em modelos descritos utilizando-se a Computação Neural, e foram obtidas regras diretas de transformação para casos específicos de modelos Neurais.

IV. TRANSFORMAÇÕES ENTRE MODELOS NEBULOSOS E MODELOS NEURAIS - SISTEMAS HÍBRIDOS ENVOLVENDO CF E CE

C. Evolução de Bases de Conhecimento de Sistemas Nebulosos por Algoritmos Genéticos

Em [10], investigou-se a implementação de sistemas híbridos para se explorar transformações de elementos de sistemas nebulosos (ou "fuzzy") em elementos de algoritmos genéticos. No trabalho mencionado, não se trabalhou com a conversão direta de modelos Nebulosos em modelos evolutivos. Em outras palavras, um sistema nebuloso e um algoritmo genético foram construídos, com finalidades diferentes, e utilizados em conjunto, em um sistema híbrido.

Esta abordagem corresponde a um passo inicial para estudar relações e transformações entre técnicas de IC, que aparentemente possuem naturezas muito distintas.

Para efeito de experimentos, este trabalho buscou resolver o problema da navegação de um robô auto-guiado em um circuito, através de simulações computacionais (software desenvolvido em linguagem de programação orientada a objetos "C#", em ambiente Windows XP).

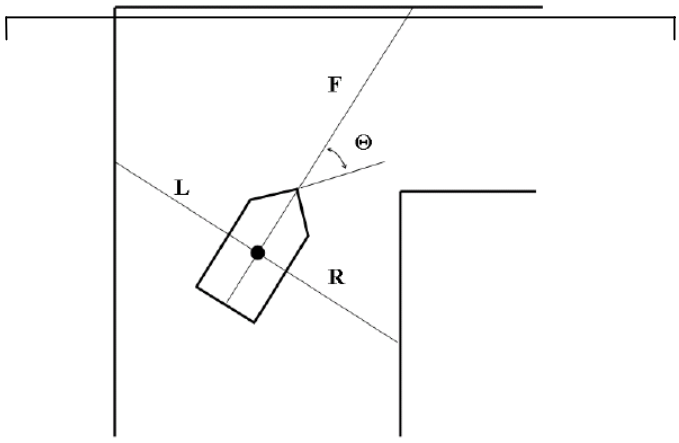


Fig. 5. Ilustração conceitual do robô auto-guiado implementado em plataforma computacional para simulações de um sistema de inferência nebuloso - retirada de [10]

O robô, representado na tela por um ponto, sempre se move para frente com velocidade constante, e um Controlador Lógico Nebuloso (CLN) é utilizado para guiá-

lo. Este sistema possui como entradas, medidas de distância do robô em relação às paredes do circuito navegado (obtidas a partir de dois sensores laterais e um sensor frontal). A saída do CLN é o valor de um ângulo, que representa a alteração na direção de movimento deste robô. O objetivo é não permitir que o robô sofra colisões com as paredes do circuito. Um objetivo secundário é evitar a realização de curvas bruscas.

Neste trabalho possíveis bases de conhecimento (envolvendo base de dados e base de regras) do sistema nebuloso utilizado para a navegação foram convertidas em indivíduos de uma população de um algoritmo genético. Com isso, o sistema nebuloso utilizado para a navegação do "robô" pôde ter suas regras evoluídas para atingir um alto desempenho de navegação (isto é: evitar colisões nas paredes da pista, e também evitar curvas bruscas).

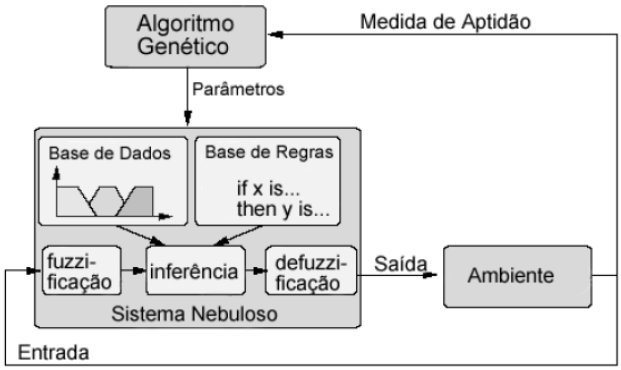


Fig. 6. Ilustração conceitual do sistema híbrido implementado em [10] - retirada de [10]

D. *Tabelas de Decisão Adaptativas*

Para esta análise, é importante notar que a base de conhecimento do CLN é modelada como um indivíduo de um AG, e alterada pelo sistema, conforme as condições de seu ambiente (neste caso, o circuito a ser percorrido pelo robô auto-guiado). A base de conhecimento envolve:

- a base de dados do CLN: correspondente às partições nebulosas das variáveis de entrada e de saída; e
- a base de regras do CLN: a qual determina seu comportamento a partir dos dados de entrada e da base de dados.

Desta forma, a base de regras do sistema (inclusa na base de conhecimento) sofre alterações realizadas pelo próprio sistema. Analisando-se este sistema híbrido quanto às capacidades e propriedades de Sistemas Adaptativos citadas em [17] e enumeradas na seção "II" deste documento, temos que todas – (1), (2), (3), (4) e (5) são atendidas pelo sistema desenvolvido em [10]. Quanto a (4), convém comentar que o sistema pode obter regras distintas, a partir de desenhos de circuitos distintos para navegação. Uma ressalva à formalização realizada em [16] é que este sistema não apenas altera sua base de regras, mas também sua base de dados. Porém, este fato não deixa de acarretar em uma alteração no seu comportamento.

Convém também comentar que, em [4], foi proposta a implementação de um AG por meio de Tabelas de Decisão Adaptativas. Neste trabalho mencionado, cada indivíduo do AG em questão (modelado por um vetor de números reais) corresponde a uma regra de um sistema. Esta propriedade permitiu que o AG fosse modelado como uma Tabela de Decisão Adaptativa, modelando os operadores de mutação e de cruzamento do algoritmo como funções adaptativas da tabela.

E. *Ajuste de Parâmetros de Algoritmos Genéticos por Sistemas Nebulosos*

Nesta frente de trabalho, estuda-se a implementação de sistemas híbridos para se explorar transformações de elementos de algoritmos genéticos em elementos de sistemas nebulosos. Desta forma, medidas de desempenho de algoritmos genéticos, como tamanho, ou diversidade da população, são modeladas como variáveis nebulosas de entrada de um sistema nebuloso; e parâmetros de controle de algoritmos genéticos, como taxa de mutação e taxa de cruzamento, são modelados como variáveis nebulosas de saída de um sistema nebuloso.

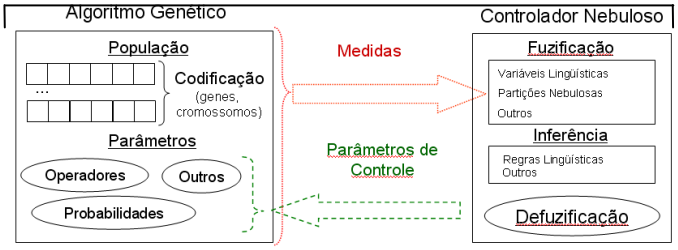


Fig. 7. Ilustração conceitual do sistema híbrido buscado por esta frente de trabalho

Diversos trabalhos são encontrados na literatura, utilizando CF para regular o comportamento de modelos de CE (predominantemente Algoritmos Genéticos). Entre eles, podemos citar: [15]; [2]; [20]; e [13].

F. *Análise deste Tipo de Sistema Híbrido sob o Ponto de Vista de Adaptatividade*

Este tipo de sistema híbrido é denominado, em [13], como "Algoritmos Genéticos Adaptativos" (do original: "Adaptive Genetic Algorithms"), e apresenta a capacidade de ajustar parâmetros do Algoritmo Genético dinamicamente, para realizar a busca por soluções de maneira mais rápida e eficiente.

Contudo, ao se analisar os trabalhos [15], [2], [20], e [13]; percebe-se que estes alteram valores numéricos dos modelos evolutivos em questão (como por exemplo: taxa de mutação; taxa de cruzamento etc). Desta forma, as regras destes sistemas não são alteradas.

Analisando-se este sistema híbrido quanto às capacidades e propriedades de Sistemas Adaptativos citadas em [17] e enumeradas na seção "II" deste documento, temos que: (1), (2), (3), e (4) são atendidas; porém (5) não é atendida.

Uma observação importante cabe ao trabalho [13]: este trabalho foca no equilíbrio entre dois tipos de comportamento de um Algoritmo Genético, definidos neste trabalho como: "exploração" (do original "exploration"); e "aproveitamento" (do original "exploitation"). O Algoritmo Genético implementado possuía dois diferentes operadores

de cruzamento: um com propriedades de "exploração", e outro com propriedades de "aproveitamento". A cada iteração, apenas um destes operadores era utilizado. A probabilidade de execução de cada um destes operadores era regulada por um único parâmetro numérico, que por sua vez era regulado pelo sistema nebuloso. Assim sendo, este trabalho afirma que o sistema nebuloso era capaz de alterar o comportamento do AG, influenciando se este irá focar em ações de "exploração" ou ações de "aproveitamento" – e esta alteração no comportamento não era realizada por alterações no conjunto de regras, mas sim pela alteração de parâmetros numéricos. Por este motivo, apesar de este trabalho ser denominado como um "Algoritmo Genético Adaptativo", este não se encaixa na definição de Sistemas Adaptativos abordada em [17] e [16].

V. ANÁLISE DA IMPLEMENTAÇÃO DE ALGORITMOS GENÉTICOS POR MEIO DE TABELAS DE DECISÃO ADAPTATIVAS

Em [4], uma técnica para implementar um AG através de Tabelas de Decisão Adaptativas é proposto. Esta seção deste documento busca analisar esta técnica e propor melhorias.

Algumas considerações importantes sobre o sistema em questão:

- o AG possuía, como indivíduos, regras de um sistema para análise de biodiversidade;
- estas regras foram todas codificadas como vetores de números reais;
- as funções adaptativas chamada pela tabela correspondiam aos operadores de cruzamento e de mutação do AG, que por sua vez alteravam as regras (ou seja, os indivíduos);
- os indivíduos codificados como regras, nunca eram utilizados como regras da tabela em si;
- a cada iteração do algoritmo, apenas uma ação de cruzamento e uma ação de mutação eram realizadas;
- o operador de cruzamento utilizado (e implementado por uma função adaptativa) foi o "cruzamento multipontos", com dois pontos (uma definição deste operador pode ser encontrada em [19]);
- a população de indivíduos aumenta sempre em um indivíduo por iteração, podendo atingir portanto o crescimento de até quatrocentos indivíduos (número máximo de iterações permitido) acima do tamanho inicial da população.
- o sistema implementado não apresentou alterações de desempenho em relação ao AG inicial, quanto ao número de iterações necessário.

Com base nestas considerações, pode-se observar:

- A princípio, qualquer AG pode ser modelado como uma Tabela de Decisão Adaptativa, e não apenas aqueles cujos indivíduos correspondam a regras, uma vez que em AGs os indivíduos são codificados como vetores de números (normalmente reais ou binários).
- Outros operadores de cruzamento, como "cruzamento de um ponto", ou "cruzamento uniforme" também podem ser implementados como funções adaptativas de uma Tabela de Decisão Adaptativa.
- São muito raros os algoritmos genéticos em que ocorre apenas uma ação de mutação e uma de cruzamento por iteração. Algumas alterações nas regras da tabela (regras estas não correspondentes a indivíduos) e também nas ações (não-adaptativas) codificadas poderiam ser realizadas, de forma a realizar um maior número de cruzamentos e mutações por iteração, implementando também taxas de mutação e de cruzamento.

VI. CONCLUSÕES

Algoritmos Genéticos, bem como sistemas híbridos (envolvendo AGs e sistemas nebulosos) podem possuir mudanças de comportamento, indicando adaptações a novas situações. Além disso, Dispositivos Adaptativos Dirigidos por Regras também podem exibir o comportamento de Algoritmos Genéticos. A utilização destas poderosas ferramentas e modelagens pode revelar diversos comportamentos em comum. A pesquisa por relações e transformações entre diferentes técnicas de IC, pode instigar a utilização destas técnicas em situações inexploradas, levando à quebra de sólidos paradigmas, porém ainda está no seu início.

Como trabalhos futuros, podemos enumerar:

- a utilização de Tecnologia Adaptativa como ferramenta para auxiliar a busca por relações e transformações entre as técnicas de IC;
- novas representações de AGs por Dispositivos Adaptativos Dirigidos por Regras, incorporando taxas de mutação e de cruzamento (uma representação e sua respectiva implementação já está em desenvolvimento, para a solução de problemas clássicos de Computação Evolutiva);
- a busca por sinergia entre Dispositivos Adaptativos, Algoritmos Genéticos e Sistemas Nebulosos.

REFERÊNCIAS

[1] ANDRADE, M. T. C. de. *Uma contribuição à pesquisa em inteligência computacional.* Tese (Livre Docência) - Escola Politécnica da Universidade de São Paulo, São Paulo, 2002.

[2] ARNONE, S.; DELL'ORTO, M.; TETTAMANZI, A. Toward a fuzzy government of genetic populations. IEEE Conference on Tools with Artificial Intelligence, v. 6, p.585-591, 1994.

[3] 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.

[4] BRAVO, C. et al. Towards an Adaptive Implementation of Genetic Algorithms INBI 2007, XXXIII CLEI – Conferencia Latinoamericana de Informática, San José, Costa Rica, 2007.

[5] CAVERSAN, F. L. Exploração de relações entre técnicas simbólicas e conexionistas da inteligência computacional. Dissertação (Mestrado) - Escola Politecnica da Universidade de São Paulo, 2006.

[6] 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 2002, , pp. 409-413. Toluca, Mexico, 2002.

[7] JORDAN M.I.; Russell S. Computational Intelligence, prefácio de The Mit Encyclopedia Of The Cognitive Sciences. The MIT Press, Massachussets, 1999.
[8] B.G.W. Craenen, A.E. Eiben; Computational Intelligence; Encyclopedia of Life Support Sciences, EOLSS; EOLSS Co. Ltd.
[9] D. B. Fogel, T. Fukuda, and L. Guan, "Scanning the special issue/technology on computational intelligence," in Proceedings of theIEEE, vol. 87, no. 9. Los Alamitos: IEEE Press, September 1999, pp.1415–1422.
[10] FIALHO, Álvaro R. S. Exploração de Relações Entre as Técnicas Nebulosas e Evolutivas da Inteligência Computacional. Dissertação (Mestrado) - Escola Politécnica da Universidade de São Paulo, São Paulo, 2007.
[11] GAJSKI, D.; KUHN, R. H. New VLSI tools. IEEE Computer, v. 16, p. 11-14, 1983.
[12] S. Haykin, Redes neurais: princípios e prática. Porto Alegre: Bookman,2001.
[13] HERRERA, F.; LOZANO, M. Adaptation of genetic algorithms parameters based on fuzzy logic controllers. In F. Herrera and J.L. Verdegay, editors, Genetic Algorithms and Soft Computing, pages 95 – 125. Physica Verlag, 1996.
[14] HOLLAND, J. Adaptation in natural and artificial systems. Ann Arbor: The University of Michigan Press, 1975.
[15] LEE, C. C.; TAKAGI, H. Dynamic control of genetic algorithms using fuzzy logic techniques. S. Forrest, Ed., Proc. of the Fifth Int. Conf. on Genetic Algorithms (Morgan Kaufmmann, San Mateo, 1993) 76-83.
[16] NETO, J.J. Adaptive Rule-Driven Devices – General Formulation and Case Study Lecture Notes in Computer Science, Springer-Verlag, Vol. 2494, pp. 234-250. Watson, B.W. and Wood, D. (Eds.): Implementation and Application of Automata 6th International Conference, CIAA 2001, Pretoria, South Africa, July 23-25, 2001.
[17] NETO, J. J. Um Levantamento da Evolução da Adaptatividade e da Tecnologia Adaptativa. Revista IEEE América Latina. Vol. 5, Num. 7, ISSN: 1548-0992, Novembro 2007. (p. 496-505).
[18] PISTORI, H.; NETO, J.J.; PEREIRA, M.C. Adaptive Non-Deterministic Decision Trees: General Formulation and Case Study. INFOCOMP Journal of Computer Science, Lavras, MG, 2006.
[19] REZENDE, S. O. Sistemas inteligentes: fundamentos e aplicações. Barueri, SP: Editora Manole, 2003.
[20] XU, H. Y.; VUKOVICH, G. Fuzzy evolutionary algorithms and automatic robot trajectory generation. Proc. of The First IEEE Conference on Evolutionary Computation (1994) 595-600.
[21] ZADEH, L. A. Fuzzy sets. Information Control, v. 8, p. 338-353, 1965.
[22] ZUFFO, F., PISTORI, H. Tecnologia Adaptativa e Síntese de Voz: Primeiros Experimentos Anais do V Workshop de Software Livre – WSL. Porto Alegre, junho de 2004.

Mauricio Alexandre Parente Burdelis possui graduação em Engenharia de Eletricidade com ênfase em Sistemas Digitais pela Escola Politécnica da Universidade de São Paulo (2001). Atualmente é mestrando Engenharia de Eletricidade, com area de concentração em Sistemas Digitais, e linha de pesquisa em Inteligência Artificial.

Tem interesse em Teoria Fuzzy, Redes Neurais, Computação Evolutiva, Sistemas Adaptativos e suas aplicações, bem como a utilização destas técnicas conjuntamente, em sistemas híbridos.

Marco Túlio Carvalho de Andrade possui graduação em Engenharia de Eletricidade pela Universidade de São Paulo (1982), Mestrado em Engenharia Elétrica pela Universidade de São Paulo (1990), Doutorado em Informática - Universidad Politécnica de Madrid (1995), Livre Docência na área de Inteligência Artificial pela Escola Politécnica da USP em 2.002. Atualmente é Professor Associado da Escola Politécnica da Universidade de São Paulo. Tem experiência, dentro do contexto de Engenharia Elétrica, na área de Inteligência Artificial, com ênfase em Inteligência Computacional (Fuzzy, RNA's e Computação Evolutiva), atuando principalmente nos seguintes temas: Teoria Fuzzy e aplicações; Modelagem Computacional de Sistemas Dinâmicos Fuzzy, Sistemas Complexos Fuzzy e Caos; Modelagem de Sistemas para Gestão de Conhecimento; Sistemas Inteligentes de Apoio à Tomada de Decisão; Modelagem e Desenvolvimento de Processos Para Gerenciamento de Projetos.

Uso de Máquina de Moore Adaptativa na Modelagem de Cursos

W. J. Dizeró e J. J. Neto

***Resumo* — Este trabalho apresenta uma proposta do uso de máquina de Moore adaptativa na elaboração de cursos para software educacionais. Uma máquina de Moore é um autômato finito com saída associadas aos seus estados. Assim, para cada estado pode-se associar o material didático a ser apresentado pela função de saída. Aplicando-se os conceitos de adaptatividade nesse transdutor, é possível elaborar cursos dinâmicos, que se auto-modifiquem com base nas experiências e nível de conhecimento individualizado dos alunos. Para complementar o artigo, um esquema de modelagem de curso baseado em máquina de Moore adaptativa é apresentado.**

***Palavras chaves* — autômato adaptativo com saída; ensino auxiliado por computador; máquina de Moore adaptativa; objetos de aprendizagem; tecnologias adaptativas.**

IX. INTRODUÇÃO

Nos últimos anos, inúmeras ferramentas computacionais foram propostas e desenvolvidas para ajudar no ensino nas mais diferentes áreas do conhecimento. É fato que o uso de tecnologias computacionais tem sido importante no auxilio do aprendizado para inúmeras modalidades de ensino, seja quando aplicada à educação infantil ou de nível superior, em programas regulares ou de educação continuada, no ensino presencial ou a distância. É verdade, também, que diversos grupos de pesquisa vêm explorando o uso do computador no ensino e atingindo um significativo grau de sucesso. Porém, as metodologias para o desenvolvimento de tais sistemas costumam empregar processos de desenvolvimento informais e muitas vezes inflexíveis. Processos informais podem levar a criação de sistemas com falhas de projetado, enquanto sistemas inflexíveis podem desmotivar seu uso. Ambas as situações podem ser cruciais no sucesso de um sistema de ensino auxiliado por computador.

Neste contexto, propõe-se um modelo baseado em máquina de Moore adaptativa, com o objetivo de definir formalmente cursos e, ainda, permitir que os cursos possam ser dinâmicos. Assim, enquanto o mecanismo formal do transdutor permite que se especifique o sistema através de uma notação matemática rigorosa, o potencial da tecnologia adaptativa [7,8,11] permite ao aluno uma experiência de aprendizagem individualizada, apresentando o material didático e seu roteiro de estudo adaptado ao nível de conhecimento e preferências.

W. J. Dizeró; (e-mail: wagner.dizero@poli.usp.br)
J. J. Neto; (e-mail: joao.jose@poli.usp.br)

Assim, a descrição deste artigo envolve, basicamente, a exposição dos conceitos acerca de máquina de Moore adaptativa, além da apresentação de um pequeno exemplo da especificação um curso baseado no modelo proposto. Na seção II é feita uma explanação sobre máquina de Moore, que será usada como o dispositivo subjacente guiado por regras (não adapativo). A seção III apresenta o modelo de um dispositivo guiado por regras adaptativo, chamado aqui de máquina de Moore adaptativa. A seção IV apresenta alguns conceitos e informações acerca de sistemas de ensino auxiliados por computador. Na seção V, encontra-se um esquema do funcionamento de um curso projetado com base no modelo proposto. Por fim, a seção V traz algumas considerações finais e propostas de trabalhos futuros para enriquecer este artigo. Ao final do texto, encontram-se as referências bibliográficas utilizadas.

X. MÁQUINA DE MOORE (NÃO ADAPATIVA)

Uma Máquina de Moore [3,4,5] é um autômato finito determinístico modificado, que possui saídas associadas aos estados. Tal máquina possui uma função que gera uma palavra de saída para cada estado da máquina, podendo essa ser uma palavra vazia. É representada por uma héptupla: $M = (\Sigma, Q, \delta, q_0, F, \Delta, \delta S)$, onde:

- Σ é um alfabeto de símbolos de entrada.
- Q é um conjunto de estados possíveis do autômato, o qual é finito.
- δ é a função programa ou de transição $\delta: Q\times\Sigma \rightarrow Q$
- q_0 é o estado inicial do autômato, tal que q_0 é elemento de Q
- F é um conjunto de estados finais tal que F está contido em Q.
- Δ é um alfabeto de símbolos de saída.
- δS é a função de saída $\delta S: Q \rightarrow \Delta^*$ a qual é uma função total que determina a geração de uma palavra de saída para cada estado.

O processamento de uma Máquina de Moore para uma dada entrada w consiste na sucessiva aplicação da função programa para cada símbolo de w (da esquerda para a direita), até ocorrer uma condição de parada. A palavra vazia como saída da função programa indica que nenhuma gravação é realizada e, portanto, a cabeça da fita de saída não se move. Se todos os estados geram saída vazia, então a Máquina de Moore se comporta como se fosse um autômato finito.

Outro tipo de transdutor é conhecido como máquina de Mealy, que também é um autômato finito modificado,

mas que possui as palavras de saída associadas com as transições entre os estados. Neste tipo de máquina, as palavras de saída dependem do estado atual e do valor das entradas. Essa máquina não será abordada aqui.

XI. MÁQUINA DE MOORE ADAPTATIVA

Uma Máquina de Moore Adaptativa (MMA) é um caso particular de aplicação do conceito de dispositivo guiado por regras adaptativo [7,11], no qual o mecanismo subjacente utilizado é o da Máquina de Moore (descrito na seção anterior). Portanto, uma Máquina de Moore Adaptativa é formalizada com uma dupla MMA = (MM0, CA), em que MM0 é uma Máquina de Moore em sua configuração original e e CA = (R,A) representa a camada adaptativa.

Os elementos da camada adaptativa, CA, são definidos:

R - é um conjunto de ações adaptativas, que contém também o valor nulo ε (representando ações que não causam alterações na camada subjacente).

$A : R \rightarrow R^2$ - é uma função que mapeia cada possível regra da camada subjacente em um par ordenado de ações adaptativas. As duas ações adaptativas deste par ordenado devem ser acionadas, respectivamente, antes e depois da execução da regra à qual elas estão associadas. Por isto, elas são também denominadas, respectivamente, ação anterior e ação posterior.

A execução das ações adaptativas induzem uma seqüência, MM_0, MM_1, MM_2, …, MM_n, de transformações da camada subjacente, na qual cada elemento MMi, $0 \leq i \leq n$ é definido de maneira análoga à MM_0. Os pares de elementos: Σ_i e δ_i; e, Δ e δS de MM_i, correspondem aos conjuntos de configurações e regras produzidos pela aplicação da uma ação adaptativa a_i $(i > 0)$, sobre MM_{i-1}.

Complementarmente a um autômato adaptativo, uma Máquina de Moore Adapativa pode adicionar ou remover as saídas associadas a cada estado, além de ter funções adaptativas para adicionar ou remover estados e transições.

XII. COMPUTADORES NA EDUCAÇÃO

A proposta dos métodos de ensino por computador, até poucos anos, vinha sempre sendo voltada para o ensino programado, que propicia um aprendizado por meio de um ambiente elaborado para atender a uma necessidade ou um objetivo bem específico.

Apesar de ter alcançado um notável sucesso, estes métodos de ensino auxiliado por computador ou CAI (Computer Assisted Instruction), como são conhecidos, nem sempre demonstram ser adequados para atender necessidades de aprendizagens mais ricas e complexas, pois limitam-se a comportar-se como meros livros eletrônicos. Tentar formatar o ensino computadorizado nos moldes do tradicional não é só improdutivo, é inviável.

A educação pode ser entendida como um processo de exploração, descoberta, observação e construção do conhecimento. Mas, é importante destacar que cada pessoa prefere aprender de maneira diferente. Algumas preferem através de recursos visuais, outras através de métodos verbais, algumas preferem explorar, outras deduzir. Ou seja, qualquer ferramenta pedagógica que não considere as diferenças individuais de cada aprendiz está fadada ao fracasso.

Desta forma, os computadores podem contribuir com esta nova demanda de flexibilidade e individualidade, tendo um papel de destaque como meio de ajuda neste novo processo educativo. Ao contrário de outras mídias utilizadas na educação, o computador, tem a característica de processar e manipular a informação que recebe. Ele pode, a partir de uma informação fornecida, transformá-la, traduzi-la, usá-la em cálculos, ordená-la, arrumá-la e mesmo fazer inferências. A flexibilidade pode ser exercida no tempo, no espaço e no conteúdo.

- Flexibilidade temporal significa que o aluno não está mais restrito ao horário da aula, podendo estipular seus próprios horários de acordo com suas conveniências.
- Já a flexibilidade no espaço significa que o aluno decide onde estudar. Se na sala de aula, no laboratório, na biblioteca ou em casa.
- Os conteúdos são flexibilizados, pois eles serão apresentados no ritmo de cada aprendiz, que decidirá a quantidade que será acessada de cada vez e com que profundidade cada tópico será abordado.

Todas estas alternativas significam que o ensino poderá ser adaptado a cada um, levando ao ensino personalizado de forma a atender às diferenças individuais. A idéia é investigar formas que permitam dotar o software educacional com a capacidade de adaptação a cada estudante.

Diferentes formalismos adaptativos poderiam ser utilizados para a modelagem de cursos, como, por exemplo, o uso de ISDL Adaptativo [2]. Contudo, optou-se pela utilização da Máquina de Moore Adaptativa devido a sua simplicidade de uso e alto poder de representação.

XIII. EXEMPLO DE MODELAGEM DE UM CURSO

Primeiramente, é apresentada a estrutura geral do sistema, na qual depois de sua abertura, é apresentado um menu para que o aluno possa escolher o curso que deseja participar. De maneira genérica, o sistema prevê o oferecimento de diferentes cursos. Assim, após a escolha do curso desejado, é realizada uma chamada à sub-máquina correspondente, passando de "A" e retornando em "B" após a realização do curso. Ao término do curso, retorna-se ao menu de escolha de cursos. As figuras 1 e 2, representam os autômatos desse modelo.

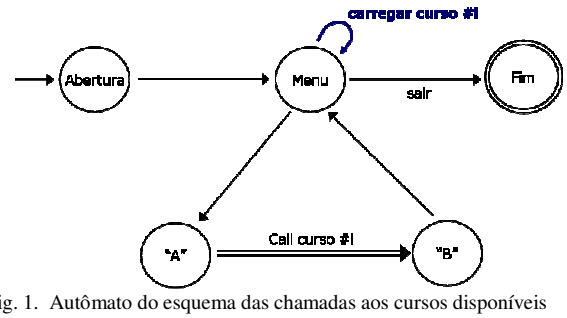


Fig. 1. Autômato do esquema das chamadas aos cursos disponíveis

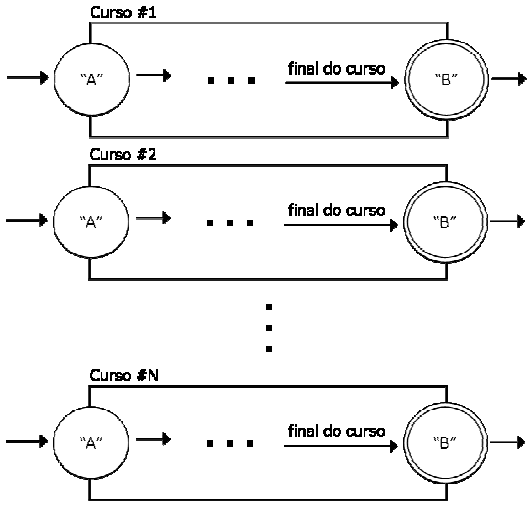


Fig. 2. Representação da chamada aos cursos.

A seguir, será, então, apresentado um exemplo demonstrando como será a representação de cada um dos cursos disponibilizados aos alunos. O exemplo será apresentado através de uma Máquina de Moore (não adaptiva). Posteriormente, serão apontadas as mudanças propostas para tornar o curso mais dinâmico, através do uso da adaptatividade.

A idéia central é elaborar um curso desmembrando-o seu material didático em pequenos blocos chamados de objetos de aprendizagem [10,12]. Assim, o curso pode ser montado com o mesmo material, de diferentes formas, ou ainda, ser modificado dinamicamente.

Cada curso é definido como um autômato [6] e referencia um conjunto de objetos de aprendizagem independentes, tal que o roteiro de estudos do curso permanece separado do material didático. Cada estado representa um tópico de estudo dentro do curso. As transições compõem o roteiro das aulas. A função de saída funciona como a ligação lógica dos estados com os objetos de aprendizagem.

Tomando como exemplo um curso para ensino de algoritmos, usando Máquina de Moore, tem-se:

Alfabeto de entrada:{próxima,exercício,anterior,resumos, saída}

Alfabeto de saída: { A, B, C, D, E, F, G, H, I }

A – Introdução a Algoritmos
B – Definição de Desvio Condicional
C – Exemplo de Desvio Condicional
D – Exercícios sobre Desvio Condicional
E – Definição de Laço de Repetição
F – Exemplo de Laço de Repetição
G – Exercícios sobre Laço de Repetição
H – Conclusões
I – Fim

A figura 3 ilustra uma possível configuração de Máquina de Moore, com suas regras de transição e de saída para a representação de um curso sobre algoritmos.

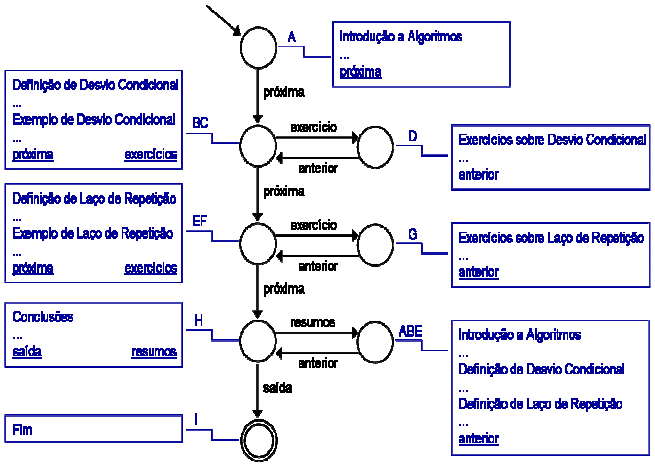


Fig. 3. Curso de Algoritmos representado com Máquina de Moore.

É possível notar que num mesmo estado podem existir mais de um objeto de aprendizagem associado, uma vez que a função de saída pode gerar palavras a partir do fecho de Kleene do alfabeto de símbolos de saída. É possível, também, notar o reuso de objetos de aprendizagem, que podem aparecer por mais de uma vez durante o curso. Da mesma maneira, é também verdade que esse material didático possa ser reaproveitado em outros cursos.

Para a modelagem de um curso estático, o funcionamento de uma Máquina de Moore é suficiente para se conseguir especificar formalmente todo o modelo. Contudo, um enriquecimento nesse tipo de máquina, através da camada adaptativa, permite maior poder de representação, podendo tornar o acompanhamento do curso mais flexível e dinâmico.

Desta forma, um sistema adaptativo tenta antecipar as necessidades e desejos dos alunos a partir de modelos que representam o seu perfil, nível de conhecimento e preferências.

Com base nos modelos de Sistemas Hipermídia Adaptativos [1,9], é proposta a aplicação de ações adaptativas em três níveis:

- Adaptatividade na navegação
- Adaptatividade na apresentação
- Adaptatividade no conteúdo

A adaptatividade na navegação tem por objetivo demonstrar um caminho ideal para que um aluno em particular alcance seus objetivos, evitando que o mesmo se disperse, diante de um amplo conjunto de opções e sinta-se desorientado diante da grande quantidade de material oferecido. A adaptação na navegação é obtida acrescentando-se novas transições no modelo do curso ou, ainda, removendo-se algumas das transições existentes. Uma vez que as transições definem o roteiro de aulas do curso, pode-se manter o mesmo conteúdo programático, porém oferecer seqüências alternativas para seguir do curso. De acordo com o perfil de cada aluno, pode-se ofertar um número maior de caminhos a serem seguidos

ou restringir essas possibilidades, tornando o curso mais linear e assim limitar o espaço de busca As regras adaptativas nesse nível incidem exclusivamente sobre a criação e remoção de transições.

A adaptatividade na apresentação diz respeito à forma como as informações serão exibidas para os alunos, dependendo do perfil de cada aluno. Ou seja, um mesmo conteúdo pode ser apresentado utilizando-se diferentes layouts ou mesmo através de diferentes mídias, conforme as preferências do usuário. Nota-se, também, que o material a ser apresentado ao aluno não está necessariamente pronto e pode ser gerado em tempo real. Essencialmente, a adaptatividade na apresentação implica em se modificar as palavras de saída associadas aos estados.

A adaptatividade no conteúdo refere-se a possibilidade de se substituir um objeto de aprendizagem por outro mais elaborado, ou mesmo mais simplificado, dependendo da situação. Com isso, o curso pode ser atualizado trocando-se as palavras de saída ou até mesmo criando-se novos estados, com tópicos complementares não contidos no curso original.

Uma questão importante a ser pensada diz respeito a onde e como aplicar regras adaptativas. A princípio, essas regras devem ser criadas pelo professor responsável pela elaboração do curso. Contudo, a critério do próprio professor, essas regras podem ser baseadas na própria inteligência coletiva do grupo de alunos participantes.

XIV. CONCLUSÕES

O uso de um modelo baseado em Máquina de Moore Adaptativa para projetar sistemas de ensino assistidos por computador apresenta um grande potencial para se criar ambientes de ensino flexíveis, que se ajustem ao perfil de cada estudante.

Uma das características importantes, que ficou evidenciada na proposta apresentada, é a possibilidade de se criar o material didático de forma independente do autômato. A independência do material didático em relação ao roteiro das aulas permite, por exemplo: reutilização do material instrucional em diversos cursos. Outra característica importante é a possibilidade de se criar, para um mesmo curso, diferentes roteiros de aulas, com enfoques diferenciados, pois um curso pode seu autômato modificado dinamicamente.

Como trabalho futuro, pode-se aplicar o conceito de máquina de Moore adaptativa para a elaboração de provas adaptativas. Monitorar o comportamento de diversos alunos e, através dos conceitos de Inteligência Coletiva, permitir que os materiais didáticos do curso e/ou roteiros de estudos do curso sejam modificados. Expandir o conceito usado para curso, para que o mesmo possa ser aplicado, por exemplo, numa grade curricular completa, e, assim, trabalhar com adaptatividade multi-nível. Criação de uma ferramenta de autoria, que ofereça uma interface gráfica amigável para que o professor possa montar o roteiro do curso, sem a necessidade de ser treinado.

REFERÊNCIAS

[1] BRUSILOVSKY, P. Methods and Techniques of Adaptive Hypermedia. User Modeling and User Adapted Interaction. Special issue on adaptive hypertext and hypermedia. Pittsburgh, 1996.

[2] CAMOLESI, Almir Rogério; JOSÉ NETO, João. Modelagem AMBER-Adp de um ambiente para Gerenciamento de Ensino a Distancia. In: SIMPÓSIO BRASILEIRO DE INFOMÁTICA NA EDUCAÇÃO, 2002, São Leopoldo. Anais do XIII Simpósio Brasileiro de Informática na Educação. São Leopoldo-RS, 2002. v. 1, p. 401-409.

[3] HARRISON, Michael A.: Introduction to Formal Language Theory, Ed. Addison-Wesley; 1a edição; Califórnia – USA (1978)

[4] HOPCROFT, J. E., Ullman, J. D.: Introduction to Automata Theory, Languages and Computation, Addison-Wesley (1979).

[5] LEWIS, Harry R. & Papadimitriou, Christos H.: Elementos de Teoria da Computação. Ed. Bookman, 2ª edição, (2000)

[6] MACHADO, Júlio P. et al.; Autômatos Finitos: um Formalismo para Cursos na Web; XIII Simpósio Brasileiro de Engenharia de Software 1999, SBES'99, Florianópolis, Brasil.

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

[8] NETO, J. J., Contribuições à Metodologia de Construção de Compiladores, Thesis (Livre Docência) – Escola Politécnica da Universidade de São Paulo, São Paulo, 1993.

[9] PALAZZO, L. A. M. (2002) "Sistemas de Hipermídia Adaptativa", In Anais do XXII Congresso da Sociedade Brasileira de Computação (SBC), Florianópolis.

[10] PIMENTA, Pedro e BAPTISTA, Ana Alice. Das plataformas de E-learning aos objetos de aprendizagem. In. DIAS, Ana Augusta Silva e GOMES, Maria João. Elearning para e-formadores. Minho, TecMinho, 2004, p. 97-109.

[11] PISTORI, H.. Tecnologia Adaptativa em Engenharia de Computacão: Estado da Arte e Aplicações. Tese (Doutorado) — Universidade de São Paulo.

[12] WILEY, D. A. Conecting learning objects to instructional theory: A definition, a methaphor and a taxonomy. The Instructional Use of Learning Objets. Wiley, D. (Ed.) 2001. Disponível na URL: http://www.reusabilility.org/read/chapters/wiley.doc. 2001. Acesso em 20/03/2005.

Wagner José Dizeró nasceu em Piracicaba-SP, Brasil, em 29 de Agosto de 1975. Se graduou na Faculdade de Tecnologia de Americana (1996) e defendeu mestrado em Ciência da Computação pela Universidade Federal de São Carlos (1999). Atualmente, é aluno de doutorado do departamento de Engenharia da Computação e Sistemas Digitais da Escola Politécnica da Universidade de São Paulo. Profissionalmente, é Professor no Centro Universitário de Lins a 9 anos, onde também é coordenador dos cursos de Bacharelado em Sistemas de Informação e Pós-Graduação em Gestão de Sistemas de Informação. Entre seus campos de interesse encontram-se: tecnologias adaptativas, engenharia de software, sistemas para Internet, informática na educação e educação a distância.

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.

XoTransito: Utilização de Algoritmos Adaptativos no Planejamento Automático de Rotas no Trânsito

F. S. Komori, R. O. Morelli, T. B. Torres e T. Matos

***Resumo*—Este artigo tem como objetivo apresentar o Xo-Transito, um sistema automático inteligente para planejamento de rotas viárias em áreas urbanas utilizando algoritmos adaptativos.**

Esse sistema foi proposto pelos autores como projeto de formatura do curso de Engenharia Elétrica com Ênfase em Computação. O projeto foi desenvolvido no Laboratório de Linguagens e Técnicas Adaptativas sob a orientação do Prof. Dr. João José Neto. Dados os pontos de origem e de destino, o sistema planeja rotas no mapa da cidade de São Paulo levando em consideração o trânsito no momento e outros fatores relevantes para o fluxo de veículos.

***Index Terms*—algoritmos adaptativos, estratificação, planejamento automático de rotas, tecnologia adaptativa.**

I. INTRODUÇÃO

O sistema XoTransito foi proposto e desenvolvido pelos autores como projeto de formatura do curso de Engenharia da Computação no ano de 2008. A proposta do trabalho foi criar um sistema que planejasse uma rota entre duas localizações da cidade de São Paulo levando em conta variáveis importantes, como o trânsito e a velocidade permitida nas vias. Esses eram considerados durante a execução do algoritmo de roteamento do sistema.

O algoritmo central utilizado no trabalho para a geração de rotas foi uma variação do clássico algoritmo A* [1]. Foram propostas alterações no algoritmo de roteamento original para torná-lo adaptativo e aproveitar as vantagens que a tecnologia adaptativa oferece para aperfeiçoar a geração de rotas de trânsito, permitindo tratar situações diversas, como congestionamentos e velocidade de vias.

Foi implementada também uma estratégia de estratificação do mapa utilizado no sistema, com a finalidade de criar um algoritmo hierárquico de roteamento, o que permite resolver o problema de forma estruturada, partindo do contexto geral e progredindo para o particular. Na estratificação, as vias da cidade são classificadas em três níveis hierárquicos. Essa técnica foi utilizada visando também economia de recursos computacionais, pois, ao classificar as vias do mapa em diversas camadas, apenas as vias de determinada camada (ou nível) são utilizadas pelo algoritmo a cada momento.

II. ALGORITMOS DE ROTEAMENTO CLÁSSICOS

No projeto, foram analisados diversos algoritmos de roteamento clássicos para serem utilizados como estrutura central do algoritmo adaptativo. Alguns algoritmos pesquisados foram: busca em largura (*breadth-first search*), busca em profundidade (*depth-first search*), busca bidirecional, busca de custo uniforme, *greedy best-first search* e A* (que também é um algoritmo do tipo *best-first search*). Tais algoritmos estão descritos em [1] e [2].

Dentre esses algoritmos clássicos, o A* recebe destaque, uma vez que é completo (sempre encontra uma solução caso exista) e ótimo (encontra a solução com o menor custo possível), desde que a heurística utilizada seja admissível (nunca superestima o custo real) e monotônica (sempre decresce com o aumento do custo). Além disso, é otimamente eficiente, ou seja, nenhum outro algoritmo ótimo garante visitar um menor número de nós do que A*. Entretanto, a aplicação desse algoritmo em sua forma original neste projeto não é viável, pois o desempenho é prejudicado para rotas entre pontos muito distantes (nos testes realizados, caminhos da ordem de 100 quarteirões foram calculados após visitarem-se mais de 8000 nós). Além disso, não é um requisito essencial que a busca encontre o caminho ótimo, já que o fator trânsito, variável e algumas vezes imprevisível, é também considerado.

O algoritmo A* se norteia basicamente por uma função $f(n) = g(n) + h(n)$, na qual $g(n)$ representa o custo do caminho da origem até o nó n (nó que está sendo avaliado) e $h(n)$ é uma função heurística que estima o custo do nó n até o destino. Dessa forma, o próximo nó a ser visitado é aquele contido na lista aberta (lista que contém os nós que ainda não foram visitados) que possua o menor valor de f. De modo simples, o A* funciona como uma busca de custo uniforme que utiliza uma heurística para determinar quais nós têm maior probabilidade de conduzir ao destino por um caminho de baixo custo, de forma a expandir menos nós durante a busca.

III. ALGORITMO DE ROTEAMENTO ADAPTATIVO

Foram implementadas três modificações no algoritmo A* e no grafo representativo do mapa buscando construir o algoritmo adaptativo. As modificações foram:

estratificação do mapa, consideração da situação do trânsito em tempo real e adaptatividade dos componentes do algoritmo A* (variação dos valores da função h).

Figura 5. Autômato de mudança de nível de ruas.

A. *Estratificação*

Considerando-se que o desempenho do algoritmo A* é baixo (da ordem de dezenas de segundos, utilizando-se os recursos computacionais atuais) para rotas longas (com mais de 50 quarteirões, aproximadamente, entre a origem e o destino), foi aplicada a estratificação do mapa de ruas de São Paulo.

Isso acarretou a criação de um mapa hierárquico, ou seja, um mapa separado em diversos níveis de detalhamento. Quanto maior o nível da observação, menor é o detalhamento do mapa (mapa com menor quantidade de ruas visíveis pelo algoritmo de roteamento do projeto).

No projeto, os níveis utilizados foram os seguintes: no nível mais alto, se encontram as vias rápidas (como as Marginais); no nível intermediário, se encontram as vias de velocidade intermediária (como a Avenida Nove de Julho); no nível mais baixo, se encontram as ruas de baixa velocidade. Desse modo, cada nível representa um detalhamento diferente do mapa, estruturando-o em camadas. A partir disso, o algoritmo de roteamento altera o nível do mapa dinamicamente, dependendo do progresso da análise de nós. Considerando essa abordagem, o algoritmo avalia todas as ruas somente no início e no fim do percurso. No meio do caminho, as ruas de menor velocidade são inicialmente descartadas com o objetivo de otimizar o desempenho. Ao final da execução desse passo, o caminho encontrado é avaliado e, caso possua muitos trechos congestionados, uma nova instância do A* é executada. Essa segunda execução somente é efetuada nos trechos mais críticos em termos de trânsito. A utilização desse algoritmo permite uma maior agilidade no processamento do algoritmo, uma vez que não verifica todos os nós possíveis no meio do caminho.

Considerando que o mapa é visto como um conjunto de regras (para cada par de nós, define se existe ou não uma aresta que os une) e que essas regras são alteradas durante a execução do algoritmo, pode-se afirmar que a estratificação permite a utilização da adaptatividade no algoritmo.

1) Exemplo de Estratificação: Considera-se o caminho entre o Tatuapé e o Butantã, uma rota longa que passa por diversas vias principais, como a Marginal Tietê, e avenidas, como a Salim Farah Maluf. Essa rota, caso fosse gerada com o algoritmo A* adaptativo com estratificação, consideraria no início todas as possibilidades (ruas, avenidas e vias principais), visando chegar a uma via de nível superior. A partir de então, consideraria apenas as vias principais e tentaria chegar ao destino utilizando somente o nível em que está ou um nível mais alto de estratificação. Ou seja, quando o algoritmo encontra uma via de nível superior ao que está atualmente, ele muda seu conjunto de regras para apenas considerar as ruas do nível superior que encontrou, diminuindo, assim, o número de nós e arestas do grafo a percorrer.

Resumindo, ele se utiliza do autômato da figura 1 para trocar entre níveis.

B. *Trânsito em Tempo Real*

Para se aproximar da realidade, o algoritmo pode considerar uma base de dados estatística (que armazena um conjunto de situações de trânsito para as ruas monitoradas em diferentes horários do dia), com o objetivo de utilizar, no grafo, um perfil de trânsito mais coerente com o momento real de percurso. Nesse sentido, o sistema considera que o motorista demora certo tempo para percorrer sua rota (que pode ser bem considerável em certos horários).

Uma abordagem como essa força o algoritmo a ser executado em pequenos trechos do percurso desejado, sendo que, a cada trecho, uma atualização no grafo de ruas é executada, com o objetivo de representar os pesos referentes ao trânsito relativos ao momento estimado em que o motorista estará naquele ponto. Adotando esse procedimento, o funcionamento do algoritmo se aproxima de uma situação ideal, pois passa a atender a grandes variações da situação do trânsito nas vias da cidade de São Paulo.

1) Exemplo de Trânsito em Tempo Real: Considera-se uma rota longa, de 25 km de extensão, ligando dois pontos distantes da cidade. É razoável considerar que, em São Paulo, essa rota demore de 1 a 2 horas para ser percorrida, e que, durante esse tempo, o trânsito varie diversas vezes no decorrer do trajeto. Caso o caminho seja traçado com o algoritmo A* adaptativo por temporização, a extensão da rota seria detectada (por exemplo, mais de 20 km), o que permite concluir que poderia haver modificação na configuração original do trânsito. Desse modo, o algoritmo calcularia a rota com o trânsito atual até certo ponto do trajeto (por exemplo, 15 km a partir do ponto inicial). Ao chegar nessa situação, o algoritmo poderia carregar dados históricos de trânsito para a próxima hora, atualizando o peso das arestas e traçando a rota entre os pontos novamente.

C. *Adaptatividade dos Componentes do A**

O algoritmo A* funciona com um grafo ponderado e

uma função heurística. Pode-se adicionar adaptavidade para esses dois elementos.

O grafo ponderado pode ser submetido a operações de inclusões e exclusões de arestas. Tais ações são realizadas, respectivamente, em casos de vias muito congestionadas (vias em que o fator de congestionamento é maior que 7, numa escala de 0 a 10) e no momento em que essas passam a ter um trânsito melhor (o fator de congestionamento é menor que 7). Outra abordagem adaptativa consiste na alteração dinâmica dos pesos considerados para a rota, modificando seu grau de contribuição para o custo total. Assim, fatores como trânsito e velocidade da via podem ter sua importância modificada no cálculo do custo total.

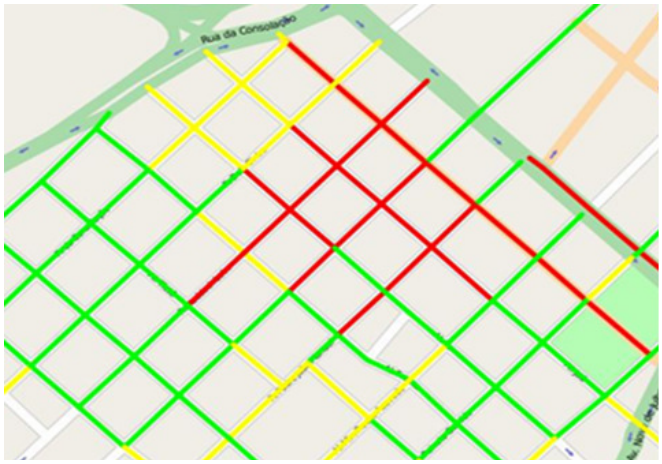


Figura 6. Simulação de trânsito local.

Figura 7. Rota traçada sobre mapa da figura 2.

Para adicionar adaptativade à heurística, é possível modificar os parâmetros da função que avalia o possível custo do nó atual até o destino. A cada invocação da função heurística, o cálculo varia de acordo com o contexto em que está sendo executado.

IV. INTERFACE COM O USUÁRIO

O sistema utiliza o *framework* e os mapas do OpenStreetMap. O OpenStreetMap é um projeto de código aberto que provê mapas de ruas e interface gráfica para visualização desses. Na base de dados geográficos, as vias (ruas, avenidas, pontes, ferrovias, etc.) são representadas através de conjuntos de nós, cada um especificado com latitude e longitude. Na interface do sistema com o usuário, que é baseada na Web, o mapa de São Paulo é apresentado. A partir dele, o usuário clica em dois pontos distintos no mapa e é traçada uma rota entre esses pontos. Caso haja trânsito, o algoritmo desvia do mesmo, passando preferencialmente pelas vias principais e avenidas.

Para representar o trânsito, foram escolhidas as cores vermelho (trânsito ruim), amarelo (trânsito mediano) e verde (trânsito bom). Na figura 2, é apresentado um mapa de uma região de São Paulo, em que o trânsito das ruas é simulado. Na figura 3, encontra-se um exemplo de rota (em azul), que procura evitar o trânsito.

V. ALGORITMO EM PSEUDO-CÓDIGO

A seguir é descrito o código implementado para o Algoritmo Adaptativo proposto nesse projeto.

```
função A*_geral(vértice_origem, vértice_destino,
horário) {
  caminho = A*(vértice_origem, vértice_destino,
  verdadeiro, horário);
   se peso_médio_trânsito(caminho) alto:
     para cada subcaminho de
     subcaminhos_alto_trânsito(caminho):
       novo_subcaminho =
       A*(vértice_inicio(subcaminho),
       vértice_fim(subcaminho), falso,
       horário[vértice_inicio(subcaminho)]);
        incorporar(novo_subcaminho, caminho);
   retorna caminho;
}
```

Um exemplo de implementação seguindo as idéias expostas é mostrado a seguir. O algoritmo principal é o `A*_geral`, que invoca o A* e as funções auxiliares:

- `caminho`: retorna uma lista de vértices que compõem o caminho encontrado;
- `vizinhos`: retorna um conjunto de vértices vizinhos considerando-se o nível passado como parâmetro (se não encontrar nada, modifica o nível até que encontre algum vértice vizinho);
- `decisão`: função que avalia se o determinado vértice vai ser processado ou não (arestas com um trânsito muito congestionado são desconsideradas, como se tivessem sido removidas do grafo);
- `peso`: calcula o custo total da aresta, considerando-se diversos aspectos: comprimento, trânsito e velocidade;
- `duração`: calcula uma duração estimada para o percurso entre os 2 vértices passados como parâmetros com base em banco de dados estatísticos;
- `heurística`: retorna uma estimativa de custo entre o vértice processado e o destino, considerando a distância em linha reta entre os dois vértices e

dados estatísticos de trânsito na região.

VI. CONCLUSÃO

Este artigo apresentou um sistema para o planejamento de rotas de trânsito para a cidade de São Paulo. A tecnologia adaptativa foi de grande importância nesse trabalho, pois facilitou a consideração de fatores adicionais aos pesos geralmente usados em algoritmos de roteamento clássicos, como o A*. Entre esses fatores, estão o trânsito e a velocidade nas vias, os quais são introduzidos em tempo real no modelo do grafo.

Com a adição desses fatores na geração da rota, o sistema passou a apresentar caminhos adequados a um motorista que irá trafegar no trânsito real da cidade de São Paulo, em que há variação das condições do trânsito durante o percurso do motorista. Os resultados obtidos foram adequados, visto que os dados de trânsito, fornecidos em etapas para o algoritmo, simulam a variação contínua do trânsito.

REFERÊNCIAS

[1] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions of Systems Science and Cybernetics*, vol. 4, no. 2, 1968.

[2] S. Russell and P. Norvig, *Artificial Intelligence*, 2nd ed. New Jersey: Prentice Hall, 2003.

[3] H. Pistori and J. J. Neto, "Adaptree," *Conferência Latino Americana de Informática (CLEI)*, 2002.

[4] J. J. Neto, "Adaptatividade e tecnologia adaptativa," *Revista IEEE América Latina*, vol. 5, no. 7, 2007.

[5] H. Pistori, "Tecnologia adaptativa em engenharia de computação," *Tese de Doutorado*, 2003.

Software para o Estudo da Língua Grega Clássica: formalismo subjacente a mecanismos adaptativos

E. G. Cavalcanti, USP, Brasil, Z. M. Zapparoli, USP, Brasil

***Resumo* — Esta pesquisa tem por objeto de estudo a flexão nominal do dialeto ático em ambiente digital. Os objetivos principais são: 1) construção de programa de computador que viabilize a flexão nominal, fornecendo a análise da palavra digitada; 2) criação de corpus do dialeto ático; 3) criação de mecanismo de busca direta no corpus do TLG, Thesaurus Linguae Graecae. São três as contribuições: a primeira – estritamente lingüística –, pela apresentação das desinências nominais sinteticamente; a segunda – interdisciplinar, na convergência das novas tecnologias digitais com as letras clássicas –, pela disponibilidade da análise morfológica em ambiente digital de forma integral e simultânea; a terceira – no emprego das Tecnologias Adaptativas –, pela geração de uma Base de Conhecimentos com os recursos complementares capazes de fornecer a combinação correta do grego ático.**

***Palavras-chave:* flexão nominal; língua grega; dialeto ático; novas tecnologias digitais; programa; lingüística; cálculo morfológico; dispositivos adaptativos; reconhecimento automático de padrões.**

I. Introdução

ESTE trabalho tem por objetivo mostrar os resultados da pesquisa de doutorado, em relação à constituição dos léxicos e à construção da flexão nominal, ambos do dialeto ático, através de *softwares* especificamente criados para esses fins.

Em primeiro lugar, apresentamos o objeto de pesquisa, as justificativas e os objetivos do projeto. Em segundo lugar, discorremos a respeito do desenvolvimento digital, esclarecendo os aspectos relativos ao ambiente de programação e ao banco de dados utilizados. Mostramos, em terceiro lugar, como se realizou a constituição do *corpus* de estudo, em seus diversos desdobramentos. Em quarto lugar, expomos o mecanismo da flexão nominal em ambiente digital: cálculo das combinações – programa *Nomes*. Em último lugar, falamos sobre as tecnologias adaptativas no sistema nominal do grego ático, especificamente no que diz respeito ao reconhecimento automático de padrões do sistema nominal do grego antigo.

II. Objeto de estudo, justificativa e objetivos

Este trabalho tem por objeto de estudo a flexão nominal do dialeto ático – uma das variantes dialetais do grego antigo, juntamente com o jônico, o dórico e o eólico. O interesse em processar o dialeto ático em ambiente digital surgiu em 2001, nas aulas de grego clássico do professor Dr. Henrique Graciano Murachco – Departamento de Letras Clássicas –, nos cursos de extensão da Universidade de São Paulo.

O fato de o método do professor Henrique não privilegiar o modelo das gramáticas tradicionais, considerado por ele impositivo e prescritivo, mas *pensar* as formas lingüísticas – nominais e verbais –, levando em consideração as suas partes fundamentais – a saber, o tema (parte fixa, sede do significado virtual, absoluto ou abstrato) e as desinências (que estabelecem a função, a pessoa, o número, a voz, o modo e o aspecto) – e os processos fonéticos que integram a sua *história*, criou-nos a convicção de sua coerência metodológica, visto desaparecerem as irregularidades morfológicas e, portanto, a idéia de que era possível flexionar as formas em um programa de computador. Bastaria, para isso, criar um banco de dados com todas as desinências nominais e verbais, extrair a desinência da palavra a ser pesquisada e submetê-la a um processo comparativo com as desinências do banco e, desse modo, gerar as combinações possíveis. Criamos, então, um protótipo digital da flexão ática e, para este trabalho, decidimos desenvolver o módulo relativo à flexão nominal.

Embora concordando com que as gramáticas gregas são impositivas, não podemos negar o papel de importância indiscutível que elas exerceram no ensino da língua grega. Recorremos, no nosso estudo, inúmeras vezes a elas, ora coletando informações, ora confrontando-as em pontos duvidosos ou lacunosos, mas *sempre as consultando*. São nelas que, bem ou mal, encontramos o *depósito das formas.* Ainda que façamos uso de programas de computador em nossas pesquisas, nossa intenção não é produzir um discurso de refutação, de desconsideração ou negação da interferência do outro, mas de complementação. Acreditamos que o saber é coletivo, interdiscursivo, dialógico, feito por várias mãos e povoado por várias vozes. Reconhecemos as limitações, mas, também, o valor histórico das gramáticas.

O avanço das novas tecnologias digitais em todos os ramos da vida humana coloca, entretanto, desafios intransferíveis às pesquisas lingüísticas. Não obstante as gramáticas tradicionais tenham desempenhado papel relevante na história do ensino, é imprescindível, hoje, que o conhecimento lingüístico esteja integrado ao conhecimento digital.

Há um *déficit lingüístico* muito grande em estudos de língua grega no Brasil que utilizam os recursos computacionais de forma integrada. Não há ferramenta informática – grego-português – que forneça *simultaneamente* diversas informações lingüísticas sobre as formas nominais. Por causa disso, entendemos que a justificativa inicial de nosso trabalho esteja relacionada ao preenchimento da lacuna nos estudos das línguas clássicas. A nossa contribuição relaciona-se, portanto, à criação de um programa de computador que processe informações sobre as formas nominais, isto é, realize a flexão nominal, estabelecendo, *simultaneamente,* o tema, o caso, o gênero, o número, *o conjunto das combinações* derivadas da *análise e do cálculo morfológicos*, quadros paradigmáticos de flexão, entrada no dicionário e busca rápida em *corpus extenso* com margem de erro desprezível, o qual possibilite o confronto com as formas criadas pelo computador, de forma a apontar para a validade ou não das combinações. Isso realizado, diríamos que à ferramenta caberia o processamento de todas as informações e ao estudioso, a tarefa de compreender os conceitos e interpretar os resultados.

Outro motivo relaciona-se à eficácia fornecida pelas tecnologias digitais. Todos nós sabemos como é penoso "decifrar" o nominativo singular dos nomes da terceira declinação (nomes com temas em consoante, soante, semivogal) ou das formas contratas, no grego antigo. Por causa dos acidentes fonéticos – que provocam as aparentes irregularidades morfológicas –, a busca da entrada nos dicionários torna-se, muitas vezes, desanimadora. É necessário que se tenha pleno domínio dos processos fonéticos, para efetuar a decomposição das formas e, assim, encontrar o nominativo singular e, portanto, a entrada no dicionário. Isso implica em investimento de *muito* tempo em exercícios de tradução para se adquirir o *know-how* satisfatório, o que provoca a desistência, não raras vezes, do estudante da língua grega. Por outro lado, se armazenássemos em um banco de dados todas as desinências nominais, com seus respectivos processos fonéticos, e transformássemos a "rotina" humana de decodificação nominal em um programa de computador, necessariamente o tempo da análise diminuiria absurdamente.

Postulamos, portanto, inequivocamente, o uso das novas tecnologias digitais nos estudos lingüísticos, não como algo *estranho* que se agrega, mas como elemento constitutivo do processo de produção do conhecimento lingüístico. Essa postulação está filiada, como vimos, a uma tendência universal irreversível de informatização generalizada das rotinas humanas que exigem repetição, precisão, volume e rapidez no processamento; tendência, aliás, presente nos processos científicos e acadêmicos de produção do conhecimento dos grandes centros de excelência.

Em coerência com o exposto acima, objetivamos, com este trabalho:

1) construção de programa de computador que viabilize a flexão nominal, fornecendo a análise da palavra digitada, isto é: o tema nominal – em vogal, consoante, soante, semivogal; o caso da forma digitada pelo usuário – nominativo, vocativo, acusativo, genitivo, dativo, locativo e instrumental; o gênero – masculino, feminino, neutro; o número – singular e plural; o particípio – *infectum, aoristo* e *perfectum,* em suas vozes ativa, média e passiva; tabela com todas as combinações propostas pelo programa, com seus respectivos quadros de flexão; entrada no dicionário – nominativo singular; quadros paradigmáticos para eventual confronto com as combinações propostas pelo programa;

2) criação de *corpus* do dialeto ático que permita a geração de um léxico em banco de dados, com indexação diferenciada dos itens lexicais, indicando o autor, a obra e a linha, visando verificar a existência das formas criadas pelo programa – busca indireta – e fornecer subsídios para eventual análise lingüística, através do cálculo dos valores lexicais em cada autor e em cada item lexical. Para isso, usaremos a noção *desvio reduzido* da Estatística Paramétrica, explicitada no método matemático-estatístico-computacional de análise de textos de André Camlong, para a confecção das T.V.Ls. (Tabelas de Valores Lexicais);

3) criação de mecanismo de *busca direta* no *corpus* do TLG, *Thesaurus Linguae Graecae* (Centro de Pesquisa da Universidade da Califórnia, *Irvine*, fundado em 1972, que digitalizou textos literários escritos em grego, desde Homero – século VIII a.C. – até a queda do Império Bizantino em 1453, totalizando 3800 autores, 12000 obras, 99 milhões de palavras). A busca direta nos textos tem por finalidade única verificar, de forma rápida e ampla, a existência das formas propostas pelo programa. A diferença entre as duas buscas – a indireta e a direta – reside no tamanho do universo lingüístico pesquisado e na forma da organização dos dados: a indireta realiza a pesquisa *no léxico* de *11 autores*, dentro de um *banco de dados,* e a direta busca a forma *nos textos* dos *3800 autores*. A intenção de alargar o universo de busca é pôr em prova, através de um *corpus* "gigante", as combinações das formas propostas. Se houver convergência entre as combinações propostas e a busca realizada no *corpus*, isto é, se a busca estornar resultado positivo, isso indica que as formas propostas pelo programa procedem;

4) reorganização das desinências nominais em *conjuntos de desinências*, diferentemente da organização apresentada pelas gramáticas;

5) criação das tabelas de flexão dos particípios. As gramáticas apresentam apenas o nominativo masculino, feminino, neutro singular, em cada aspecto verbal, deixando de desenvolver os demais casos;

6) apresentação do processo fonético que deu origem à determinada forma nominal.

III. DESENVOLVIMENTO: AMBIENTE E BANCO DE DADOS

A escolha do ambiente de programação implicou algumas dificuldades. A primeira delas diz respeito à grande variedade de linguagens de programação existentes no mercado. Precisávamos de uma linguagem que fosse produtiva, confiável e que tivesse acesso a vários bancos de dados. A segunda, à não-existência de códigos abertos (*open source*) ou projetos disponíveis semelhantes para que pudéssemos ter um ponto de partida na construção de um projeto de flexão nominal do dialeto ático. A terceira, ao suporte técnico à linguagem escolhida

Dentre as linguagens do mercado, optamos por trabalhar com as que eram orientadas a objetos. Dentre elas, destacam-se o C++ e o Delphi, em sua versão 7, que têm, como linguagem nativa, o Pascal orientado a objetos. Decidimos pelo Delphi, pelo fato de ser um ambiente de fácil manipulação e de alta produtividade, além de ter suporte técnico disponível, fornecido por várias empresas, no Brasil (filiamo-nos ao Clube do Delphi, sediado na cidade de Avaré, interior do estado de São Paulo).

A escolha do banco de dados também não foi simples. Deveria ser robusto, confiável, com excelente performance e gratuito. Os que se encaixavam nessas características eram: SQL Server 2005, PostGresSQL, FireBird e MySQL. Para efetuarmos a escolha, realizamos um teste de performance com a tabela do léxico do dialeto ático, com 200689 registros. A busca realizou-se sobre o último registro da tabela. O resultado foi o seguinte:

TABELA1: PERFORMANCE DOS BANCOS DE DADOS

200689 registros **Busca do último registro**					
Banco de Dados (Gratis)	***Tempo em ms***	***Performance***		***Problemas de cache***	***Nativo do Delphi 7***
SQL Server 2005	**140**	**21,34%**	**1º**	**N**	**S**
PostGresSQL	**172**	**26,22%**	**2º**	**N**	**N**
FireBird	**188**	**28,66%**	**3º**	**S**	**S**
MySQL	**656**	**100,00%**	**4º**	**S**	**N**

A escolha, a partir do teste de performance, ficou clara: SQL Server 2005. Embora sua capacidade esteja limitada a 4 GB por banco, é extremamente rápido, não dá problemas de *cache* local e pode ser acessado por componente nativo do Delphi – o componente ADO, *Active Data Objects.*

Na atual fase do projeto, utilizamos o SQL Server 2005 e o FireBird. Na etapa final, migraremos todos os dados para um único banco, isto é, o SQL Server 2005.

IV. CONSTITUIÇÃO DO *CORPUS* ÁTICO: FONTES, OBRAS, AUTORES, EXTRATO DOS TEXTOS E LÉXICO

O *corpus* que deu origem ao léxico foi formado a partir da coleta de textos na *Internet,* nos *sites* de domínio público, que disponibilizam obras de autores antigos. Basicamente, foram duas as fontes: a) *Perseus Digital Library,* em http://www.perseus.tufts.edu/; e b) *Thesaurus Linguae Graecae*, na seção aberta ao público, em http://www.tlg.uci.edu/demo.html.

Os textos extraídos da *Internet* foram processados e submetidos a uma indexação diferenciada das encontradas comumente. Para tal, criou-se uma *ferramenta digital*, de nome *Corpus,* que gerou o Léxico com: a) as freqüências de uso dos itens lexicais nas obras processadas, no todo e nas partes – autor por autor; b) o endereço de cada item lexical, contendo: nome do autor, obra e linha em que se deu a ocorrência da palavra; c) o peso (valor) lexical nas obras do *Corpus.*

Tendo em vista o objeto deste trabalho, optamos por selecionar um conjunto de autores que escreveram em grego ático. A obra completa de cada autor está em formato texto, num mesmo arquivo. Isso facilita a busca morfológica e o reconhecimento por qualquer plataforma, como *Windows* e *Linux*.

Pelo fato de utilizarmos o *corpus* do TLG nas buscas rápidas, mantemos a correspondência do nome do autor com o nome do arquivo correspondente no TLG, visando a uma uniformidade. Assim:

TABELA2: AUTORES QUE ESCREVERAM EM GREGO ÁTICO

Aeschylus trag.	**TLG0085.txt**
Aristophanes comic.	**TLG 0019.txt**
Aristoteles	**TLG 0086.txt**
Demosthenes orat.	**TLG 0014.txt**
Euripides trag.	**TLG 0006.txt**
Isocrates orat.	**TLG 0010.txt**
Lysias orat.	**TLG 0540.txt**
Plato	**TLG0059.txt**
Sophocles trag.	**TLG 0011.txt**
Thucydides hist.	**TLG 0003.txt**
Xenophon hist.	**TLG 0032.txt**

As obras dos autores que escreveram em grego ático, no nosso *corpus*, somam um total de 316.

Os textos extraídos na *Internet* passaram por um tratamento informático, através do programa *Corpus*, módulo Ferramentas, com formatação específica. Mantivemos o formato *beta code*, porém em letra minúscula. Seguem-se os extratos dos textos – obras – de ***Aeschylus*** *trag.*, e ***Aristoteles***, com o endereço lexical à esquerda, bem como um recorte do léxico:

Extrato dos textos: *Aeschylus trag.*,

Aesch.Suppl01.000001: {ΙΚΕΤΙΔΕΣ} {ΧΟΡΟΣ ΔΑΝΑΙΔΩΝ} Ζεὺς μὲν ἀφίκτωρ ἐπίδοι προφρόνως
Aesch.Suppl01.000002: στόλον ἡμέτερον ναίον ἀρθέντ'
Aesch.Suppl01.000003: ἀπὸ προστομίων λεπτοψαμάθων
Aesch.Suppl01.000004: Νείλου. Δίαν δὲ λιποῦσαι χθόνα σύγχορτον Συρίᾳ
Aesch.Suppl01.000005: φεύγομεν,
Aesch.Suppl01.000006: οὔτιν' ἐφ' αἵματι δημηλασίαν
Aesch.Suppl01.000007: ψήφῳ πόλεως γνωσθεῖσαι,
Aesch.Suppl01.000008: ἀλλ' αὐτογενεῖ φυξανορίᾳ, γάμον Αἰγύπτου παίδων ἀσεβῆ
Aesch.Suppl01.000009: τ'
Aesch.Suppl01.000010: ὀνοταζόμεναι διάνοιαν.
Aesch.Suppl01.000011: Δαναὸς δὲ πατὴρ καὶ βούλαρχος

[redacted]

οἱ περιλειφθέντες τῶν Σπαρτῶν, ὡς Αἰσχύλος φησίν, ἦσαν
Aesch.Fragm11.015264: Χθόνιος, Οὐδαῖος,
Aesch.Fragm11.015265: ασιος Αἰσχύλος ομα Λακεδαιμόνιον ἀποφαίνει τὸν Αλκμᾶνα· λέγει γὰρ ἐν τοῖς
Aesch.Fragm11.015266: ῾Υακινθ ἄκουσα ταν ἀηδον παρ' Εὐρώτα ταν' Αμυκλα
Aesch.Fragm11.015267: μεναι τατ τον εὐνομω ουσαναυτα
Aesch.Fragm11.015268: ἀρεταν ταν που μέλεσι ταλλαν ταν τ ' Αταρνίδα ἐν
Aesch.Fragm11.015269: · ἐν γὰρ τούτοις γράφειν ταπ ' Αλκμᾶνος τον αλ
Aesch.Fragm11.015270: ἐπιλε

Extrato dos textos: *Aristoteles*

Arist.Analy01.000001: {ΑΝΑΛΥΤΙΚΩΝ ΠΡΟΤΕΡΩΝ Α.} Πρῶτον εἰπεῖν περὶ τί καὶ τίνος
Arist.Analy01.000002: ἐστὶν ἡ σκέψις, ὅτι περὶ ἀπόδειξιν καὶ ἐπιστήμης
Arist.Analy01.000003: ἀποδεικτικῆς· εἶτα διορίσαι τί ἐστι πρότασις καὶ τί ὅρος καὶ τί
Arist.Analy01.000004: συλλογισμός, καὶ ποῖος τέλειος καὶ ποῖος ἀτελής, μετὰ
Arist.Analy01.000005: δὲ ταῦτα τί τὸ ἐν ὅλῳ εἶναι ἢ μὴ εἶναι τόδε τῷδε, καὶ
Arist.Analy01.000006: τί λέγομεν τὸ κατὰ παντὸς
Arist.Analy01.000007: ἢ μηδενὸς κατηγορεῖσθαι.

[redacted]

τοῦ χειμῶνος γίνεσθαι τοὺς ὑετούς ... ἐπιζητήσαι δ' ἄν τις,
Arist.Fragm55.117324: πῶς εἰ τὸ θέρος ξηρόν ἐστιν, ὡς προείρηκε, παρ' οἷς ξηρότατόν ἐστι τὸ θέρος,
Arist.Fragm55.117325: παρὰ τούτοις οἱ ὑετοὶ τοῦ θέρους γίνονται. ἢ αἱ μὲν ἀναθυμιάσεις καὶ αἱ τῶν
Arist.Fragm55.117326: νεφῶν συστάσεις
Arist.Fragm55.117327: οὐκ ἐκεῖ γίνονταί τε καὶ
Arist.Fragm55.117328: γεννῶνται, ἀλλ' ὑπὸ πνευμάτων τῶν ἐτησίων πρὸς τὰ ἐκεῖ ὄρη ὄντα ὑψηλὰ ἀπωθούμεναι συνίστανται, ὡς λέγει, τῆς δὲ ταχείας
Arist.Fragm55.117329: αὐτῶν εἰς ὑετὸν μεταβολῆς ἡ προειρημένη ἀντιπερίστασις αἰτία, ὡς λέγει ἐν
Arist.Fragm55.117330: τοῖς Περὶ τῆς τοῦ Νείλου ἀναβάσεως.

Recorte do léxico:

TABELA 3 – LÉXICO

	Total
Palavras	**3038958**
Itens lexicais	**200689**

Ordem	Palavra	Total	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11
1	α	4	0	0	0	0	0	0	0	2	2	0	0
2	αβρων	1	0	0	0	1	0	0	0	0	0	0	0
3	αβρωνος	3	0	0	0	3	0	0	0	0	0	0	0
4	αδην	1	0	0	0	0	0	0	0	1	0	0	0
5	αγνων	5	0	0	0	0	0	0	0	0	0	5	0
200680	ζωτικην	3	0	0	2	0	0	0	0	1	0	0	0
200681	ζωτικης	1	0	0	0	0	0	0	0	1	0	0	0
200682	ζωτικον	1	0	0	1	0	0	0	0	0	0	0	0
200683	ζωτικον	3	0	0	1	0	0	0	0	1	0	0	1
200684	ζωτικωτερα	1	0	0	1	0	0	0	0	0	0	0	0
200685	ζωτικωτερον	1	0	0	1	0	0	0	0	0	0	0	0
200686	ζωτικῳ	1	0	0	0	0	0	0	0	1	0	0	0
200687	ζωτικωτερους	1	0	0	0	0	0	0	0	0	0	0	1
200688	ζωυφιον	4	4	0	0	0	0	0	0	0	0	0	0
200689	ζχ	2	0	0	2	0	0	0	0	0	0	0	0

V. O MECANISMO DA FLEXÃO NOMINAL EM AMBIENTE DIGITAL: CÁLCULO DAS COMBINAÇÕES – PROGRAMA NOMES

O CÁLCULO MORFOLÓGICO OBEDECE ÀS SEGUINTES ETAPAS:

A) ENTRA-SE COM O VOCÁBULO GREGO

Figura 1 – Entrada do vocábulo no programa Nomes

b) inicia-se um laço (*loop*) com repetição igual ao número de casos, isto é, 10.

laço 1: nominativo singular;
laco 2: vocativo singular;
laço 3: acusativo singular;
laço 4: genitivo singular;
laço 5: locativo, dativo, instrumental singular (D.L.I.);
laço 6: nominativo plural;
laco 7: vocativo plural;
laço 8: acusativo plural;
laço 9: genitivo plural;
laço 10: locativo, dativo, instrumental plural (D.L.I.).

c) abre-se a tabela com as desinências.

d) faz-se uma análise comparativa do vocábulo digitado com as desinências da tabela, da primeira até a última, caso a caso. O ponto de referência é a desinência cadastrada no banco de dados.

Vocábulo digitado:

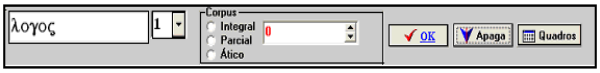


Figura 2 – Entrada do vocábulo no programa Nomes

TABELA 4 - ENTRADA DO VOCÁBULO E DESINÊNCIAS RELACIONADAS AO VOCÁBULO λογος

Singular							Plural				
ORDEM	N.	V.	A.	G.	D.L.I.		N.	V.	A.	G.	D.L.I.
1	ος	ε	ον	ου	ῳ		οι	οι	ους	ων	οις
24	ος		ον	ου	ῳ		οι		ους	ων	οις
36	ς		να	νος	νι		νες		νας	νων	σι

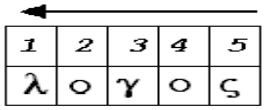
e) calcula-se quantos caracteres formam o vocábulo digitado. Em λογος, p. ex., temos 5 caracteres.

f) estabelece-se um novo *loop* com repetição igual ao número de caracteres do vocábulo.

g) dentro desse novo *loop*, extrai-se a desinência do vocábulo digitado, do fim para o começo, seguindo os critérios:

1. quantidade de caracteres a partir da extensão desinencial (ED), dada no banco.
2. posição do caracter, do último até o limite da ED.

P. ex.: vocábulo λογος → no banco, uma das desinências: **ος, ED=2.**



Quantidade de caracteres: 5.

3. *loop* com repetição 5.

4. Laço 1:

■extração do caracter ς "raiz-tema" λογο-

5. Laço 2 (ED = 2):

■extração do caracter ο "raiz-tema" λογ-

6. efetua-se ou não a combinação.

Situações:

1. a desinência do banco (DB) concorda, por completo, com a desinência do vocábulo (DV), e as flexões procedem – registro 01;

2. a desinência do banco (DB) concorda, por completo, com a desinência do vocábulo (DV), e as flexões não procedem – registros 24 e 36 ;

3. a desinência do banco (DB) concorda, parcialmente, com a desinência do vocábulo (DV), não há flexões – registro 06;

4. a desinência do banco (DB) não concorda, na última posição, com a desinência do vocábulo (DV), não há flexões – registro 02.

Como resolver o empate, quando houver concordância entre DB e DV? Situações 1 e 2. Há algum indicador independente dos indicadores gramaticais? Há algum indicador independente dos indicadores gramaticais?

I. Estando no nominativo, registro 1:

1. Decomposição da desinência do vocábulo de acordo com a desinência do banco:
   a. vocábulo digitado: λογος
   b. tabela → registro 1 → desinência do nominativo: ος (extensão desinencial (ED)2)
   c. decomposição do vocábulo (ED = 2):
      i. extraindo-se o último caracter: λογος = λογο (provável raiz)
         • comparação do ς (posição final) de λογος com ς (posição final) da desinência
         • sendo iguais, continua-se a extração
      ii. extraindo-se os dois últimos caracteres: λογος = λογ (raiz)
         • comparação do ος (posição final+1) de λογος com ος (posição final + 1) da desinência
         • sendo iguais, efetua-se a soma
   d. soma do que sobrou do vocábulo digitado (raiz) com a desinência do banco:
      i. raiz, a partir da decomposição: λογ
      ii. desinência do banco: ος
      iii. soma de ambos: λογ + ος = λογος (combinação)
      iv. combinação = vocábulo digitado
2. Processam-se as combinações.

II. Estando no nominativo, registro 02:

1. Decomposição da desinência do vocábulo de acordo com a desinência do banco:
   a. vocábulo digitado: λογος

b. tabela → registro 2 → desinência do nominativo: ον (extensão desinencial (ED): 2)

c. decomposição do vocábulo (ED = 2):

i. extraindo-se o último caracter: λογος = λογο (provável raiz)

• comparação do ς (posição final) de λογος com ν (posição final) da desinência

• são diferentes. Combinação interrompida, a desinência do banco não corresponde à desinência do vocábulo.

III. Estando no nominativo, registro 06:

1. decomposição da desinência do vocábulo de acordo com a desinência do banco:

a. vocábulo digitado: λογος

b. tabela → registro 06 → desinência do nominativo: ας (extensão desinencial (ED): 2)

c. decomposição do vocábulo (ED = 2):

i. extraindo-se o último caracter: λογος = λογο (provável raiz)

• comparação do ς (posição final) de λογος com ς (posição final) da desinência

• sendo iguais, continua-se a extração

ii. extraindo-se os dois últimos caracteres: λογος = λογ (raiz)

• comparação do ος (posição final+1) de λογος com ας (posição final+1) da desinência

• são diferentes. Combinação interrompida, a desinência do banco não corresponde à desinência do vocábulo.

IV. Estando no nominativo, registro 24:

1. decomposição da desinência do vocábulo de acordo com a desinência do banco:

a. vocábulo digitado: λογος

b. tabela → registro 24 → desinência do nominativo: ος (extensão desinencial (ED): 2)

c. decomposição do vocábulo (ED = 2):

i. extraindo-se o último caracter: λογος = λογο (provável raiz)

• comparação do ς (posição final) de λογος com ς (posição final) da desinência

• sendo iguais, continua-se a extração

ii. extraindo-se os dois últimos caracteres: λογος = λογ (raiz)

• comparação do ος (posição final+1) de λογος com ος (posição final+1) da desinência

• sendo iguais, efetua-se a soma

d. soma do que sobrou do vocábulo digitado (raiz) com a desinência do banco:

i. raiz, a partir da decomposição: λογ

ii. desinência do banco: ος

iii. soma de ambos: λογ + ος = λογος (combinação)

iv. combinação = vocábulo digitado

2. Processam-se as combinações.

V. Estando no nominativo, registro 36:

1. decomposição da desinência do vocábulo de acordo com a desinência do banco:

a. vocábulo digitado: λογος

b. tabela → registro 36 → desinência do nominativo: ς (extensão desinencial (ED): 1)

c. decomposição do vocábulo (ED = 1):

i. extraindo-se o último caracter: λογος = λογο (provável raiz)

• comparação do ς (posição final) de λογος com ς (posição final) da desinência

• sendo iguais, efetua-se a soma

d. soma do que sobrou do vocábulo digitado (raiz) com a desinência do banco:

i. raiz, a partir da decomposição: λογο

ii. desinência do banco: ς

iii. soma de ambos: λογο + ς = λογος (combinação)

iv. combinação = vocábulo digitado

2. processam-se as combinações.

3. processam-se todos os registros em todos os casos.

Diante do processamento realizado, qual decisão deve ser tomada? Qual das combinações é a mais provável? Para tal, é necessário observar os indicadores fornecidos pelo programa, a saber:

1. ED – extensão desinencial – classificação decrescente;

2. % nos casos – busca no léxico ou no *corpus*;

3. existência, no vocabulário, da forma nominativa construída.

O alinhamento desses três indicadores, em vários contextos, aponta para a combinação mais provável. Vejamos:

Para o vocábulo **λογος**, após processamento, há as seguintes combinações**:**

1	1	2	90%	S	1		o	o
24	3	2	100%	S	1		jo	o
36	2	1	30%	S	1		v	v

Figura 3 – Combinações efetuadas pelo programa Nomes

3 - Nominativo Singular: **λογος**
Análise: nominativo relativo masc. singular: **λογος**
Tema em: **ο, λογο-**

λόγος-ου, ὁ, (Τ λόγο-), a palavra, o discurso, a razão

Figura 4 – Combinação 3

2 -Nominativo Singular: λογος
Análise: nominativo relativo interrogativo masc. e fem., relativo indefinido masc. e fem singular: λογος
Tema em: ν, λογον-

λόγος-ου, ὁ, (Τ λόγο-), a palavra, discurso, razão

Figura 5 – Combinação 2

1 - Nominativo Singular: λογος
Análise: nominativo nome masc. e fem., adjetivo masc., relativo masc. singular: λογος
Tema em: ο, λογο-

λόγος-ου, ὁ, (Τ λόγο-), a palavra, o discurso, a razão

Figura 6 – Combinação 1

A partir dos indicadores – extensão desinencial, porcentagem nos casos e existência, no vocabulário, da forma nominativa construída –, a combinação proposta pelo programa é a de número 3: ED = 2, 100% de ocorrência no *corpus* e a forma nominativa encontrada no vocabulário (S). Entretanto, o vocábulo λογος não é um relativo, como quer a combinação 3, mas um nome-substantivo, como mostra o vocabulário. Portanto, a combinação mais provável, imediatamente após a 3, é a de número 1.

VI. Tecnologias Adaptativas no Sistema Nominal do Grego Ático

Tendo em vista que dispositivos adaptativos podem ser utilizados em diversas situações, principalmente nas que pressupõem resoluções de problemas práticos e tomadas de decisão na solução de um problema, como possibilidades de aplicações da Tecnologia Adaptativa na área das Ciências Humanas, mais especificamente na Lingüística, destacamos aspectos da Adaptatividade que podem ser aplicados ao Sistema de Computação para o Estudo da Língua Grega Clássica.

Técnicas, métodos e ferramentas para Tecnologia Adaptativa bem como suas aplicações podem possibilitar que o sistema computacional, enquanto mecanismo subjacente, não-adaptativo, realize operações de automodificação em seu próprio comportamento – código –, em tempo de execução, de forma autônoma, sem interferência externa. Para tanto, é preciso estabelecer um conjunto de funções e regras adaptativas – camada adaptativa e programas adaptativos, respectivamente – para que seja possível alterar, a partir de determinados estímulos e ações, as configurações iniciais (de CF_1 para CF_2, de CF_2 para CF_3, de CF_3 para CF_n).

No nosso projeto de flexão nominal, a criação de regras adaptativas pode ser apropriada nos seguintes pontos:

a) reconhecimento automático de padrões do Sistema Nominal do Dialeto Ático a partir das pesquisas morfológicas feitas pelo Sistema ao Léxico exaustivo do TLG, com o objetivo de validar ou não as combinações dos casos da língua grega propostas pelo cálculo morfológico. A utilização dos recursos de aprendizagem de máquina através da construção de regras adaptativas pode levar à combinação morfológica correta, sem a necessidade de se recorrer ao léxico grego. Isso porque nosso sistema não indica, categoricamente, qual combinação nominal é a "verdadeira", mas propõe um conjunto de combinações morfológicas possíveis, reagrupando-as em ordem decrescente de ocorrências no léxico;

b) alargamento do universo lingüístico de forma a permitir que o dispositivo adaptativo estabeleça a flexão nominal não só para o grego, mas também para outras línguas sintéticas, a exemplo do hebraico, latim, sânscrito etc. – construção de um conjunto de regras para os comportamentos idênticos (desinências que não sofrem maiores alterações fonéticas, em função do que a raiz nominal não é afetada – maioria dos casos) e de um conjunto de regras para os comportamentos diferenciados (casos em que as alterações nas desinências acabam por afetar a raiz nominal). Acreditamos que isso possa ser formalizado e aplicado para todas as línguas sintéticas.

É dessa forma que vislumbramos resultados positivos de uma confluência do nosso sistema com a área da Inteligência Computacional – uma conexão do formalismo já desenvolvido a mecanismos adaptativos será eficiente para a geração de uma Base de Conhecimentos com os recursos complementares capazes de fornecer a combinação correta do grego ático e de permitir aplicação a outras línguas sintéticas.

VII. Conclusão

Enfatizamos, de um lado, que a pesquisa está em curso e, por isso, muitos aspectos podem sofrer alterações; de outro, que a idéia geral do projeto, a saber, a construção de um programa de computador que forneça um *conjunto de informações* sobre a flexão nominal do dialeto ático, está, a cada passo da nossa pesquisa, concretizando-se.

A orientação principal de se constituir um banco de dados para armazenar informações para tratamento computacional da língua grega já é uma realidade neste trabalho. Inúmeros benefícios poderiam ser listados, mas

ficamos apenas com o relacionado à eficácia e alta produção lingüísticas. A análise e o cálculo morfológicos efetuados pelo computador – que possiblita a flexão nominal (e futuramente a flexão verbal) – e a busca das formas nos textos dos autores, num espaço de tempo ínfimo – num *corpus de aproximadamente 99 milhões de palavras* –, são vantagens indiscutíveis do uso das Novas Tecnologias Digitais nas pesquisas lingüísticas. O emprego das Tecnologias Adaptativas descortina a possibilidade do reconhecimento automático de padrões na flexão nominal da língua grega e de outras línguas sintéticas.

REFERÊNCIAS

[1] BAILLY, Anatole. Dictionnaire grec français. 26e ed. Paris: Hachette, 196

[2] CAMLONG, André. Lexicométrie I - La correlation simple. O teste t de Student-Fisher ou a regressão linear simples? Toulouse: Université de Toulouse Le-Mirail, 2003.

[3] _____. Lexicométrie II - La corrélation simple et la corrélation multiple. Toulouse: Université de Toulouse Le-Mirail, 2003.

[4] _____. Lexicométrie III ou La lexicométrie avec Stablex. Toulouse: Université de Toulouse Le-Mirail, 2003.

[5] _____. Méthode d'analyse lexicale textuelle et discursive. Paris: C.R.I.C. & OPHRYS, 1996.

[6] _____. Stablex pratique. Toulouse: Teknea, 1991.

[7] _____; BELTRAN, Thierry. Stablex. Manuel d'utilisation. Toulouse: A.P.I, 1991.

[8] _____; _____; ZAPPAROLI, Zilda Maria. A propósito do Stablex - versão PC. São Paulo: Pirus Tecnologia, 2004.

[9] NETO, João José. Tecnologia Adaptativa. São Paulo: LTA - Laboratório de Linguagens e Tecnologias Adaptativas, set. 2004. Disponível em: http://www.pcs.usp.br/~lta/roteiro_estudo/index.html. Acesso em 6 de novembro de 2008.

[10] CANTÙ, Marcos. Dominando o Delphi 2. Trad. de Edimilson Kazwyoshi Miyasaki. São Paulo: Makron Books, 1996.

[11] CAVALCANTI, Edenis Gois. Construção de Software para o Estudo da Língua Grega Clássica: o Sistema Nominal do Dialeto Ático na Visão Temática. São Paulo, 2008. Tese (Doutorado em Letras - Lingüística) – Faculdade de Filosofia, Letras e Ciências Humanas, Universidade de São Paulo. /em desenvolvimento/

[12] MURACHCO, Henrique. Língua grega: visão semântica, lógica, orgânica e funcional. São Paulo: Discurso Editorial / Vozes, 2001.

[13] ZAPPAROLI, Zilda Maria; CAMLONG, André. Do léxico ao discurso pela Informática. São Paulo: EDUSP/FAPESP, 2002.

Application of Adaptive Decision Tables to Enterprise Service Bus Service Selection

Fabiana S. Santana (fabiana.santana@usp.br)*#, Claudio Barberato (cbarberato@fei.edu.br)#, Renata L. Stange (rlstange@usp.br)*, João J. Neto (joao.jose@poli.usp.br)*, Antonio M. Saraiva (amsaraiv@usp.br)*

** Departamento de Computação e Sistemas Digitais – Escola Politécnica da Universidade de São Paulo – São Paulo – Brazil*
 # Departamento de Ciência da Computação – Centro Universitário da FEI – São Paulo – Brazil

***Abstract* — An Enterprise Service Bus, ESB, is an essential part of a Service-Oriented Architecture, SOA. Among other attributions, an ESB needs to manage service ranking and selection during the application running time. This is a highly complex computational problem. This paper presents a solution for this problem based on adaptive decision tables. One of the main features of the presented solution is to allow the integration of many other algorithms as adaptive functions. This is desirable because many algorithms were already proposed but with different requirements, so it is necessary to propose a solution able to integrate different contexts. Besides, a general solution must be able to support the integration requirements that are necessary to ESB and SOA. In order to illustrate the viability and relevance of the proposal, examples are also presented, along as the future researches that may be developed considering this work as a starting point.**

***Keywords* — Adaptive systems, Adaptive decision tables, Enterprise service bus, Service-oriented architecture, Adaptive algorithms.**

I. NOMENCLATURE

BPEL – Business Process Execution Language
CORBA – Common Object Requesting Broker Architecture
DCOM – Distributed Component Object Model
ERP – Enterprise Resourcing Planning
ESB – Enterprise Service Bus
FTP – File Transfer Protocol
HTTP – HyperText Transfer Protocol
IT – Information Technology
JBI – Java Business Integration
JDBC – Java DataBase Connectivity
JEE – Java platform, Enterprise Edition
POP3 – Post Office Protocol version 3
QoS – Quality of Service
RMI – Remote Method Invocation
SMTP – Simple Mail Transfer Protocol
SOA – Service Oriented Architecture
SOAP – Simple Object Access Protocol
SOC – Service Oriented Computing
UDDI – Universal Description, Discovery and Integration
WSDL – Web Services Description Language
WS-* – Family of W3C standards
XML – Extensible Markup Language
XMPP – Extensible Messaging and Presence Protocol

II. INTRODUCTION

A SYSTEM architecture may be defined as the fundamental organization of the components of a system and their relationships, among themselves and with the external environment [1]. Usually, basic principles, named as *architectural patterns*, conduct the architectural design, development and system evolution over time [5]. Architectural patterns are a main concern to IT professionals, which everyday must face the challenge to improve and maximize the usage of available resources while are steady pressured for reducing development and maintenance costs, in addition to improve the quality skills of each solution. Among the main and most expensive resources are the software programs which have already been developed.

Fig. 1 presents the architectural evolution over time, from Monolithic based to Services based solutions, as discussed in ENDREI et al. [5]. Monolithic is considered the simplest pattern while Services is considered the most sophisticated one. Monolithic applications are self-contained, single-tiered and contain the main program, the user interface and the data access code. In order to attend main concerns of IT professionals, such as reuse and maintenance, monolithic applications are usually not recommended. For instance, if a problem is found, the whole application must be revised, not only the damage part. In this sense, architectural patterns presented in Fig. 1 show evolutions to software engineering by adding layers and establishing communication among them. Distributed systems are required for the last three.

The alternatives presented in Fig. 1 usually are seen as evolutions in architectural paradigms. This is due to each solution claims to be better than previous considering the following criteria [5]: 1) Management of corporative systems; 2) Improvements in system scalability; 3) Cost reduction; 4) Collaboration and reuse of solution; and 5) Integration and interoperable skills. However, requirements of each software packages direct the architectural pattern choice.

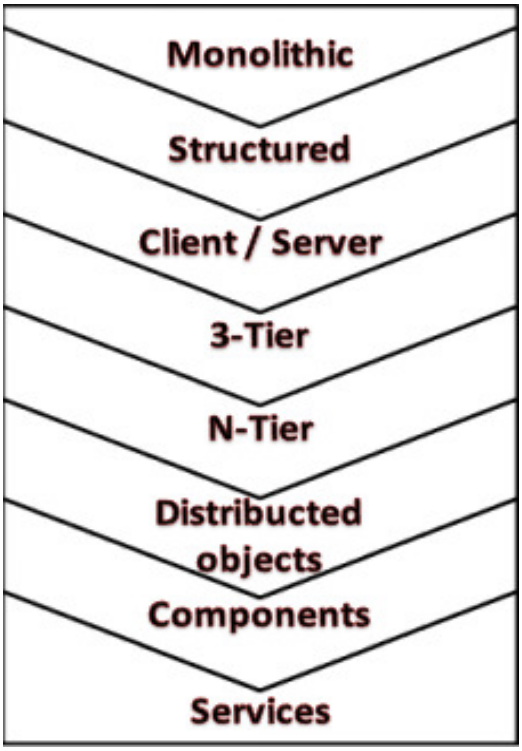


Fig1. Architectural evolution over time.

SOC is a new paradigm to develop software using services as basic units. It has evolved from component-based software frameworks, such as JEE [http://java.sun.com/javaee/], CORBA [http://www.corba.org/], .NET [http://www.microsoft.com/NET/] and DCOM [http://msdn.microsoft.com/en-us/library/ms809340.aspx]. IBM [http://www.ibm.com/], Microsoft [http://www.microsoft.com/], SUN [http://www.sun.com/], Hewlett-Packard [http://www.hp.com/], Oracle [http://www.oracle.com/], SAP [http://www.sap.com/] and other major companies in this area consider Web services as an adequate approach to SOC adoption [9].

Open Internet protocols XML and HTTP are the basis for Web services standards (e.g.: SOAP, WSDL, UDDI, and BPEL), which makes SOC adoption cheaper and easier [20] [9]. Services may be offered by applying many different technologies, but nowadays Web services are probably the most adopted one [13]. As main ideas presented in this work do not depend on any specific service technology and are applied to Web services as well as are applied to many other service technologies, this work will not discuss all of them in details unless strictly necessary, focusing on only this one to discuss the concepts presented, whenever it were necessary.

SOC, Service Oriented Computing, enables the execution of transactions across multiple platforms, providing advanced software interoperability. The construction of a system in SOC demands the integration of services in providers distributed worldwide. Thus, each application may be designed based on a business process, representing the steps to solve the computational problem. A business process may integrate, during its execution, different IT systems, available in different service providers. Process' steps may be provided by internal or external (outsourcing) service providers. Cost and performance are among the main reasons to define how a service will be provided. Once these definitions are made, one of the main challenges in SOC is related to service creation, composition and selection, over the Internet, especially in the case of outsourcing [9], where the conditions are not under the control of applications owners.

SOA is the architectural paradigm related to SOC. Applications which present requirements such as collaboration and reuse of interoperable solutions, high integration needs (e.g.: ERP systems), interoperability skills (e.g.: distributed systems) or reusable components, usually should consider SOA as architectural pattern and SOC as the correspondent development paradigm [11]. This is the case of many problems which concerns to IT professionals.

Initial investments for SOA adoption usually require a financial and computational effort which may be considerable, but, after that, the development cost of each application tends to be reduced over time, as reuse may be applied in large scale. It is also possible to implement SOA for reusing legacy systems. This strongly reduces software programming efforts [5] [11]. However, as wisely said by BROOKS [3], there are "no silver bullets" when the subject is software development and maintenance, and SOA-based systems also have their intrinsic problems, which may not be disregarded.

Most common problems are related to QoS, Quality of Service. QoS represents a wide range of issues that must be treated in order to offer a good solution to the final user. QoS issues in service technologies must consider many different properties [10] [13]. For instance, related to Web services, it is possible to enumerate: 1) Availability: time that a service keep operating, in percentage values; 2) Security: include authentication mechanisms to offer and use services, data integrity and reliability, and data confidentiality, among others; 3) Response time: time taken by a service to respond to requests which have been done; and 4) Throughput: related to the rate a service can process requests, usually as a function of time.

In addition to QoS, others issues must be considered. For instance, consider the case of non-deterministic applications, such as ecological niche modelling [19]. They are typical cases where the correctness of the results is usually more relevant than the response time, nevertheless the response time must be considered anyway. However, to describe each issue related to service selection is out of the bounds of this work and,

whenever necessary.

In an architectural perspective able to disregard the complex details of dealing with services technology, the implementation of a SOC application may be reduced mainly to choose and integrate services. This represents a simple alternative to implement applications in a very fast and simple way, as desired by many IT professionals. But there are problems intrinsic to the services nature which must be addressed. For instance, when a service is requested, it may present some problems related to QoS, such as availability; it is usually an unpredictable problem.

The service orchestration concept allows the design and implementation of software solutions by detailing the requirements (e.g.: interface, inputs, outputs) of the services necessary to the application [9]. If this was performed without to tie each service to specific providers, actions may be taken to prevent and correct application problems due to problems in the services, such as to replace a problematic service by other, in running time. Therefore, a main challenge of SOA-based solutions is service ranking and selection, so as to develop applications able to choose and replace services in running time.

In order to provide an adequate services' management, SOA-based systems use ESBs [9]. Nevertheless, a very much relevant aspect of an ESB is the service ranking and selection itself [4]. This might not be a concern of some researchers yet because services technologies are still incipient in many companies, but certainly will became a problem when many service providers start to compete with each other. When that happens, the main question will be: "How to select a service, if many different providers furnish analogous solutions?". Service selection may be provided by ranking the available services according to specific criteria and applying some selection technique among the many computational techniques.

Adaptive techniques [7] [16] may enable the construction of algorithms and a framework for service ranking and selection. They can be applied to choose the best or a subset of good services among a service list. This list may be indexed and re-indexed according to the service behaviour, considering results of their previous executions. At this point, it is important to note that QoS and other service attributes may not remain fixed over time, and a good service today may loose its quality tomorrow, so the re-indexation, or re-ranking, is a demanding task. Considering this specific aspect of the problem, adaptive technologies seem to be the most appropriate solution. Among many adaptive techniques, adaptive decision tables seem to be more adequate, since its structure is related to UDDIs, which store services information.

This work proposes a solution to construct ESBs using adaptive decision tables, which will be able to manage changes in services issues based on QoS and, considering these changes, to re-ranking the services by changing ranking and selection rules over time. Other services constraints also are considered. The paper starts by presenting some IT backgrounds, where SOA, ESB and adaptive decision tables are discussed. Then, the proposal of adaptive decision table for service ranking and selection is presented. After that, the architectural solution is presented, showing where adaptive decision tables would be included, this time in an architectural context. For the sake of comprehension, this will be explained using a factual system reference architecture. Examples to illustrate the application of the overall proposal of adaptive decision tables for service selection are also presented, and discussion and future works lead to the conclusion of this work.

III. IT BACKGROUND

A. *Service-oriented architectures*

A reference model [5] is a standard to decompose a system in parts, which must be combined to collaboratively solve the original problem. An architectural pattern describes the elements of a software architecture, the relationships among them and the restrictions to specify the fundamental structure of an application so as to obtain a complete architectural solution. A software architecture (or concrete architecture) presents the structure and organization furnished by the system, subsystems and components, and the interactions among them, in order to build systems able to fulfil identified and analyzed properties [5].

SOA is a paradigm to organize distributed competences, controlled by different providers or not [5], where reuse may be more possible than imagined before. Competences, in the system development context, are usually developed to solve problems when they appear. However, in a distributed scenario, a part may have needs compatible with competences already developed by others. Since the competences may already be available, SOA proposes services as the technology to provide that, and the required details to use them are known, a part may use the competences furnished by the other [5]. So, it is possible to define a service as the competence to execute jobs for others.

More specifically, SOA assumes that applications provide functionalities as reusable services [14]. A service is a self-contained component which may be accessible through a standardized and pre-defined interface. Any application may implement and offer services which may be used by other applications. Therefore, it is possible to implement complex business process by combining services from different sources, which is named as service orchestration.

To offer a service, a service provider must register it on an UDDI (or equivalent), which is a central naming

service. Then, when a service was requested, consumer applications have sources to look for available services, retrieve information about connection to the respective service providers, and get service description necessary to define the usage skills of a service. Any service-based technology (e.g.: Web services, SOAP) may be used to implement SOA.

A SOA-conformity solution must [5]: have entities identified as services, according to the reference model definitions; define how visibility is established among services consumers and providers; identify how to mediate interaction; be able to understand how effects of services will be understand; associate description with services; be able to identify the execution context required to support interaction; and identify policies and contracts.

B. Enterprise service bus

The construction of applications based on service orchestration is possible in a distributed scenario even when the service providers do not use the same operating system, programming languages or data models. This flexibility is fundamental for integration, since it enables the link among very different systems and environments.

Therefore, a powerful solution to integrate services based on open standards and able to support SOA is required, and this solution is usually named as Enterprise Service Bus [14]. Thus, an ESB needs to deal with open standards and infrastructures to integrated distributed systems, which requires: 1) Service routing; 2) Service invocation and mediation; 3) Abilities to integrate distributed applications and services with reliability; and 4) Security skills.

ESBs main components are routers, transformers, adapters and bridges, so as to integrate and interoperate applications over different middlewares (software to connect other software components), providing communication skills. [14].

In terms of resources, an ESB should offer support to SOAP, WSDL – Web Services Description Language, UDDI – Universal Description, Discovery and Integration, and WS-*, the family of W3C standards. In addition, communication mechanisms as JBI, RMI, JDBC, SMTP, POP3, FTP or XMPP are also required. For message routing and transportation of sources decoupled from the destinations (allowing a sender to send messages without specifying exact destinations), transformations or translations resources (e.g.: transport protocol, message format and content) are demanding, as much as the usage of a common data format (XSL, for instance, is a powerful tool to use XML messages). Besides, adapters are recommended to connect APIs and data structures, in addition to facilities to administrate the infrastructure, among others identified requirements.

The proposal in this work is that ESBs become able to provide the facilities to accomplish all these tasks, and the solution proposed in this work should be aggregated to ESBs implementation.

C. Adaptive Decision Tables

A conventional decision table [17] is a device composed of conditions rows and actions rows, where the columns represent rules associated to conditions and actions. The basic decision table operation is: the conditions defined by the rules are verified and, if one of them is satisfied, this rule is considered as a valid one and all action associated to that rule are performed. Table 1 illustrates the concept by presenting a conventional decision table where c_1, c_2, …, c_n are the conditions rows,

Table 1 – A conventional decision table.

		0	1	2	3	4	5	6	7	8	9
Condition rows	c_1										
	c_2										
	…										
	c_n										
Actions rows	a_1										
	a_2										
	…										
	a_m										

and a_1, a_2, …, a_m are the actions rows.

Adaptive decision tables [6] [8] are adaptive devices which may be obtained by extending conventional (non-adaptive) decision tables, by adding rows to encode the adaptive actions to be performed. These actions must be able to alter the rules defined in the original decision table, so as to change the behaviour of respective conventional decision table. Adaptive rules are usually checked before and/or after conventional rules. By modifying the rules defined in the original decision table, it may be possible to remove rules, to add rules and to change the behaviour of an established rule [15].

Table 2, adapted from NETO [15] and BRAVO ET AL. [2], presents an adaptive decision table with adaptive function rows, b_{a1}, b_{a2}, …, b_{af}. c_1, c_2, …, c_n are the conditions rows and a_1, a_2, …, a_m are the actions rows, as in the conventional decision table, but they may be changed according to adaptive functions rows.

IV. Adaptive Decision Table for Service Ranking and Selection

A conventional decision table to service ranking and selection may be constructed simply by using the following: 1) A defined number of lines to describe the conditions; 2) A defined number of lines to describe the actions; and 3) Each service will be describe in a column. 1) and 2) are from the original definition, and 3)

introduces the services in the decision table.

The condition rows c_1, c_2, …, c_n, are the n conditions described by QoS of Web services, each attribute represented by a c_i, $1 \leq i \leq n$. Besides availability, security, response time and throughput, the c_i, $1 \leq i \leq n$, conditions also may include other functional and non-functional conditions, such as: 1) Correctness of the answer; 2) Overload of a service in a specific moment; and 3) User preference (when user interaction is available).

The list of criteria may be changed according to the application, then it is desirable that an Adaptive ESB have mechanisms to configure (and adapt) also this list.

There are real conditions which are related to a numerical interval (e.g.: 0 ≤ response time ≤ MAX_TIME, where MAX_TIME is a constant value). In these cases, in fact, the condition may be split in two related conditions c_i, $1 \leq i \leq n$, presenting the minimum and maximum values for each service attribute, so as to offer numeric values to allow the decision table implementation (e.g.: c_1: 0 ≤ response time and c_2: response time ≤ MAX_TIME). The format itself is not that relevant, but the aim of defining a numerical interval must be achieved. Table 3 presents the structure, similar to the structure presented in Adaptive Decision Tables defined to ADAPTGARP algorithm [2]. Each condition c_i, $1 \leq i \leq n$, is split into two other conditions: c_{iA} and c_{iB}, $1 \leq i \leq n$, respectively for the lower and upper bounds of the interval, i.e., c_{iA} means $v_i \geq i_{I1}$ and c_{iA} means $v_i \leq i_{I2}$, where v_i is the variable which is being analyzed (e.g.: response time), and $[i_{I1}, i_{I2}]$ is the acceptable interval (e.g.: [0,10] seconds for response time). The same feature also might be provided by adding extra columns instead of lines. If the interval was not necessary (e.g.: service availability), the previous condition definition may be applied.

Defined the structure of the adaptive decision table, the practical question related to service attributes/issues is: "how to evaluate each of them in order to be represented by numerical values?" The answer is not unique and is not simple, since each attribute must be studied separately. Common QoS attributes will probably have values defined by companies or systems, related to what may or may not be accepted, but other attributes may be more difficult to evaluate. Consider, for instance, the correctness of an answer of a non-deterministic experiment, implemented as a Web service. This is highly dependant of the problem. However, even in this case, some metrics must be adopted by the research community involved with the specific problem, since it is supposed that every model must be evaluated. So, correctness must be studied in the respective community, and the acceptable interval (or decision, in case of Boolean experiments) for this issue may be defined to be inserted in the decision table. Ecological niche modelling, for instance, includes a statistical evaluation of each generated model [19], and there are studies about these statistics. The results of both may drive the interval definition. Other criteria, such as user evaluations, may be more simplistic to define, since it is just necessary to ask the user to attribute a number in the range of a to b, and, then, assume [a,b] as the valid interval (e.g.: marks from 1 to 10). Nevertheless, the main point is that, since a value or an interval was defined, the adaptive decision table for service ranking and selection is ready to be used.

It is necessary to define, then, how to use the adaptive decision table. Since a condition c_i, $1 \leq i \leq n$ (eventually defined by c_{iA} and c_{iB}, $1 \leq i \leq n$), is satisfied, a respective action a_j, $1 \leq j \leq m$, may be associated to that. In this case, as the aim is to ranking and select services, a score may be associated with each number belonging to the numerical interval associated to the condition c_i, $1 \leq i \leq n$ (c_{iA} and c_{iB}, $1 \leq i \leq n$). The score may be a number or a function, depending on the complexity of the criteria.

A scoring algorithm must be defined by the usage of an extra structure to store services information, which would be integrated to the adaptive decision table to include the results obtained after execution of each action a_i, $1 \leq i \leq n$. Several proposals were also presented for achieving

Table 2 – An adaptive decision table.

		0	1	2	3	4	5	6	7	8	9
Condition rows	c_1										
	c_2										
	…										
	c_n										
Actions rows	a_1										
	a_2										
	…										
	a_m										
Adaptive functions rows	ba_1										
	ba_2										
	…										
	ba_f										

that and each situation must consider which of them is more adequate, if there are any. To analyze and discuss all presented proposals for service ranking is out of the bounds of this work but it is a job to be done. If any of the proposals fit, the problems must be a new proposal must be discussed.

Only in the moment when a service was invocated, all services in the table able to match the requirements of that service must be evaluated and their score must be evaluated. After calculating the scores of the fitting services, ranking and selection are simple tasks, which can be reduced, in algorithm theory, to a vector sorting problem. So, the solution described so far, based on conventional decision tables only, would be enough to solve the problem if the attributes of the services would be fixed and reliable, over time.

However, to assume this fact as an absolute true is not possible. There are several issues involved in service technologies which may lead to a wide set of problems [12]. For instance, service providers may present internal errors and they may affect the quality of all services furnished by them, communication problems may occur by many different internal and external factors, a service provider (and thus the required service) may be overloaded, among many others regular problems that Internet heavy users are familiar to deal from time to time.

So, as time goes by, adaptive mechanisms must be considered (and included in the adaptive part of the decision table) to manage adaptive decision table lines, including, removing or altering the lines according to adaptive functions described in b_{a1}, b_{a2}, …, b_{af} lines. It is necessary to evaluate the values of each interval attributed to each c_i, $1 \leq i \leq n$ (eventually defined by c_{iA} and c_{iB}, $1 \leq i \leq n$) condition, and the actions rules defined to each a_j, $1 \leq j \leq m$, in order to including, removing and changing their relevance according to system evolutions. This will permit to change the scoring behaviour so as to perform services ranking and selection.

There are general criteria, once a list of c_i, $1 \leq i \leq n$ (eventually defined by c_{iA} and c_{iB}, $1 \leq i \leq n$) conditions were defined. For instance, the columns related to the available services should be dynamically increased. Suppose availability quality attribute of service; it may receive an initial value, which may be improved or reduced by the adaptive functions each time a service was invoked.

Main problems, again, rely on the adaptive lines which are directly related to each problem, since different problems have different behaviours, and therefore the changes in their behaviours also are dependant of the scope of the system studied. For instance, an Internet Banking must have a response time upper bounded by 15 or 20 seconds, while a modelling system that executes very complicated algorithms may take from hours to days to reach a solution, so a response time of a week would be easily acceptable. Other example, it may be difficult to decide if a faster service is better than a more precise one (e.g.: routing services), or if a faster service is more often available than other [12].

Table 3 – An adaptive decision table with intervals.

		0	1	2	3	4	5	6	7	8	9
Condition rows	c_{1A}: $v_1 \geq i_{11}$										
	c_{1B}: $v_1 \leq i_{12}$										
	c_{2A}: $v_2 \geq i_{21}$										
	c_{2B}: $v_2 \leq i_{22}$										
	…										
	c_{nA}: $v_n \geq i_{n1}$										
	c_{nB}: $v_n \leq i_{n2}$										
Actions rows	a_1										
	a_2										
	…										
	a_m										
Adaptive functions rows	ba_1										
	ba_2										
	…										
	ba_f										

Even though, related problems are a matter of adaptive decision table intervals adjustment and user decisions, and for each problem domain all the criteria may be studied and defined. In any way, this does not reduce the solution credit, since its application still remain valid after an initial configuration. Therefore, the proposal provides a framework to re-evaluate services criteria using adaptive techniques, as desired.

V. ARCHITECTURAL SOLUTION – HOW TO INTEGRATE ADAPTIVE DECISION TABLES IN THE ESB CONTEXT

As an ESB requires dealing with many different issues and the application of each adaptive decision table may be dependant of the domain of each application, a single and fixed adaptive decision table will hardly be the solution for all ranking and selection service problems for SOA application, specially considering a more general architectural context, such as a Web portal. Therefore, the construction of an ESB able to incorporate more than one adaptive decision table is the main challenge of the architectural solution.

From the architectural viewpoint, it must be designed an environment to control this feature in an adequate way, since a company may have a single ESB to integrate many different applications, from many different domains (e.g.: financial companies).

Consider that to implement the adaptive proposal previously presented in this work, any domain must have its respective adaptive decision tables, configured and adequate to the specific conditions of each problem to be solved. In the financial company, for instance, there are systems to both human resource control and financial investments.

At first sight, the implementation of so many adaptive decision tables may seem a huge challenge and a very difficult task, but it is important to remind this is the SOA world, so it only will be necessary to create a service able to generate generic adaptive decision tables integrated to the ESB, able to configure such tables according to each domain. So, the implementation of adaptive decision tables is not an architectural problem by itself (more difficult than to implement that will be to define and configure the criteria, but this problem in not related to the system architecture).

The real architectural problem, which requires an architectural study in order to be solved, is how to incorporate a *Solution Manager* to existing ESBs patterns, in order to enable the adequate usage of adaptive decision tables. This Solution Manager must be able to associate each application to the right table without much effort, so as to avoid performance reduction. In addition, the Solution Manager must be able to treat different sources of services, with different patterns.

However, despite of the challenge to make the Solution Manager real in the ESB pattern, since this component were defined, the selection service and ranking problem will be solved, and many different approaches to solve the selection problem itself will may be integrated to an ESB, improving its skills to perform dynamic service ranking and selection in an intelligent and adaptive environment. Therefore, the effort to solve such a problem is more than valid. In this work, an example of a reference architecture including an service bus will serve as the basis to present the proposal.

In SANTANA ET AL. [18], a reference architecture for ecological niche modelling were proposed [19]. This example was chosen because of its complexity and because its architecture, among others features, details the main characteristics of any ESB, marked in the figure by the lightgrey color, avoiding to tie the solution to any specific platform, such as Java.net OpenESB [https://open-esb.dev.java.net/]. Therefore, it is effective to take a closer look to this architecture, focusing mainly on the ESB features.

At first, however, it is interesting to provide a briefly explanation about ecological niche modelling systems, so as to make the example understandable. Ecological niche [19] is an important concept used as a foundation for determining geographic species (e.g: plants and animals) distribution. It is related to the conditions that allow a species survival, disregarding external factors, such as human influence, biotic interactions and geographic barriers, which might prevent from a species to grow within the ecological niche area. The main hypothesis is that if a species can be found in certain conditions, then it should be able to survive and reproduce in any place with the same conditions. The modelling technique based on this concept aims to obtain areas similar to those where the species is known to occur, resulting in an ecological niche model.

Many different computational algorithms may be applied to obtain a niche model (including adaptive algorithms [2]). These algorithms mainly consider species occurrence and absence data, represented as coordinates of points in the geo-referrenced space of the study area, and data about the environmental conditions that are relevant to the species survival at those same points. Therefore, a model may be obtained in many different forms, and by using services spread worldwide.

Modelling algorithms produce models to represents the probability of finding species under the time–space conditions described by the input data. The model may be projected onto a map of the study region or is applied to obtain projections onto different regions or periods in time (past, present, and future), enabling the studies of the research community in ecological and environmental areas.

Fig. 2 presents the results of six experiments applying ecological niche modelling technique for mapping *Ouratea spectabilis*, a tree that can be found in Open Cerrado, Cerrado sensu stricto and Cerradão, a vegetation

of the Brazilian Savannah. Models are projected onto a map of the State of São Paulo, Brazil, and are related to 2006. Algorithms applied are: 1) Minimum distance; 2) Climate space model; 3) Bioclim; 4) Garp best subsets; 5) Distance to average; and 6) Environmental distance. Blue points are the original species occurrence points and the color scale represents the probability to find the species, where black means the probability is equal to zero. This figure was extracted from [19] so as to illustrate the problem to be solved by the presented architecture.

This technique has already been successfully used to propose scenarios for sustainable use of the environment, to evaluate the potential of invasive species, to evaluate climatic changes impacts on biodiversity, to delineate potential routes of infections and diseases and to indicate potential priority areas for conservation, among others.

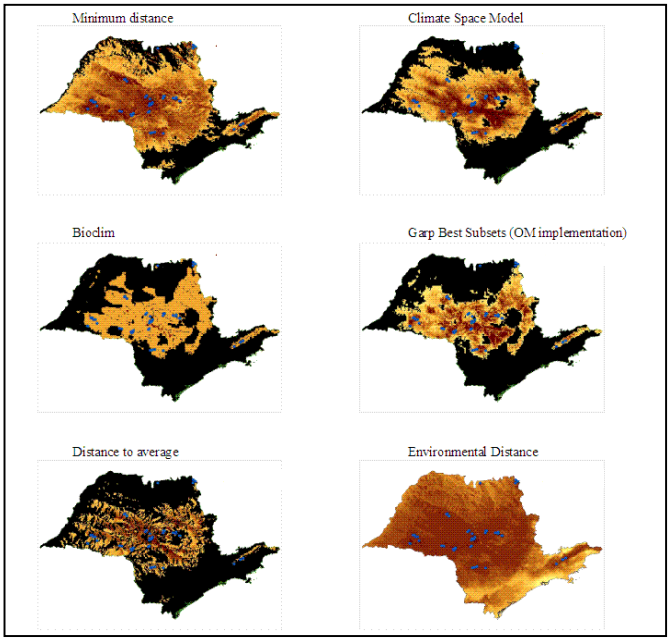


Fig 2. Results of six ecological niche modelling experiments with Ouratea spectabilis *species, using same input data and six different modelling algorithms. Models are projected onto the State Of São Paulo – Brazil.*

Ecological niche model generation is a complex computational process, involving a variety of data, techniques and software packages. It is data-intensive, since environmental variables may be stored in very large data files, of the order of many gigabytes. Data sources are frequently distributed and accessed via Internet, which may require adequate connectivity and bandwidth. Data quality and format are other relevant issues, demanding data pre-processing, cleaning and formatting, so as to provide adequate input data for software packages. A wide variety of methods and spatiotemporal data-analyses can be applied, and specific data conversion may be required to run the same experiment and to compare results obtained by different algorithms. The connection with all the sort of components presented in Fig. 3 may be required.

The reference architecture, presented in Fig. 3 [18], considers mainly the service-oriented architectural style. Service Bus (the equivalent to an ESB within this architectural proposal) receives the requests from applications and calls the appropriate services; after a service processing is finished, answers are stored in the results Repository and the client is notified, receiving a reference to retrieve desired results from the Repository (this may be put available in the Web portal or to be sent to the client, for instance, by e-mail, sms or other communication technology).

Clients may access the applications using the portal so as to offer easy and standard interfaces to the users. Applications must interact with the Service Bus, which invokes necessary services, even if they were distributed over different service providers, using some specific service protocol, such as Web services. The Service Bus must guarantee the requests delivery and, when necessary, must transform the data before communicating with the services. The Repository must store results that require huge processing effort and thus are invoked in an asynchronous way. Other services usually are Web services hosted in application servers on the Internet.

In order to increase the reusability and modifiability of the system, because each layer just knows the neighbouring ones, the architecture is organized into layers. Layers are: *client* – access devices, e.g. *web browsers, wap-phones* or *pagers*, for user input data for processing and receiving answers; *presentation* – offers services using a *web portal*, creating a single entry point for the system; *integration* – integrates applications with services, required for a complete business process, including a *service bus*, for routing and conversion services, and a *repository*, that offers mechanisms for temporary storing results; *business* – services required for the complete execution of a business process, including services based on OGC, *Web Feature Service* (WFS) and *Web Map Service* (WMS) (OGC, 2006); *resources* – data base applications, legacy systems and other devices required for precision agriculture.

Observe that this solution [18] already predicted the inclusion of a Service Engine, even before the adaptive technique was considered. In this proposal, the Service Engine has the attribution to search UDDIs so as to find services based in their description (usually, using BPEL or related W3C patterns).

However, to solve the architectural problem presented in this paper, the Service Engine must be extended to a major component which will encapsulate the current functions in other component, in this work named as *Service Search*. The *Solution Manager* would be other component of the Service Engine, with the function to specifically ranking and select services based on adaptive decision tables, but implementing

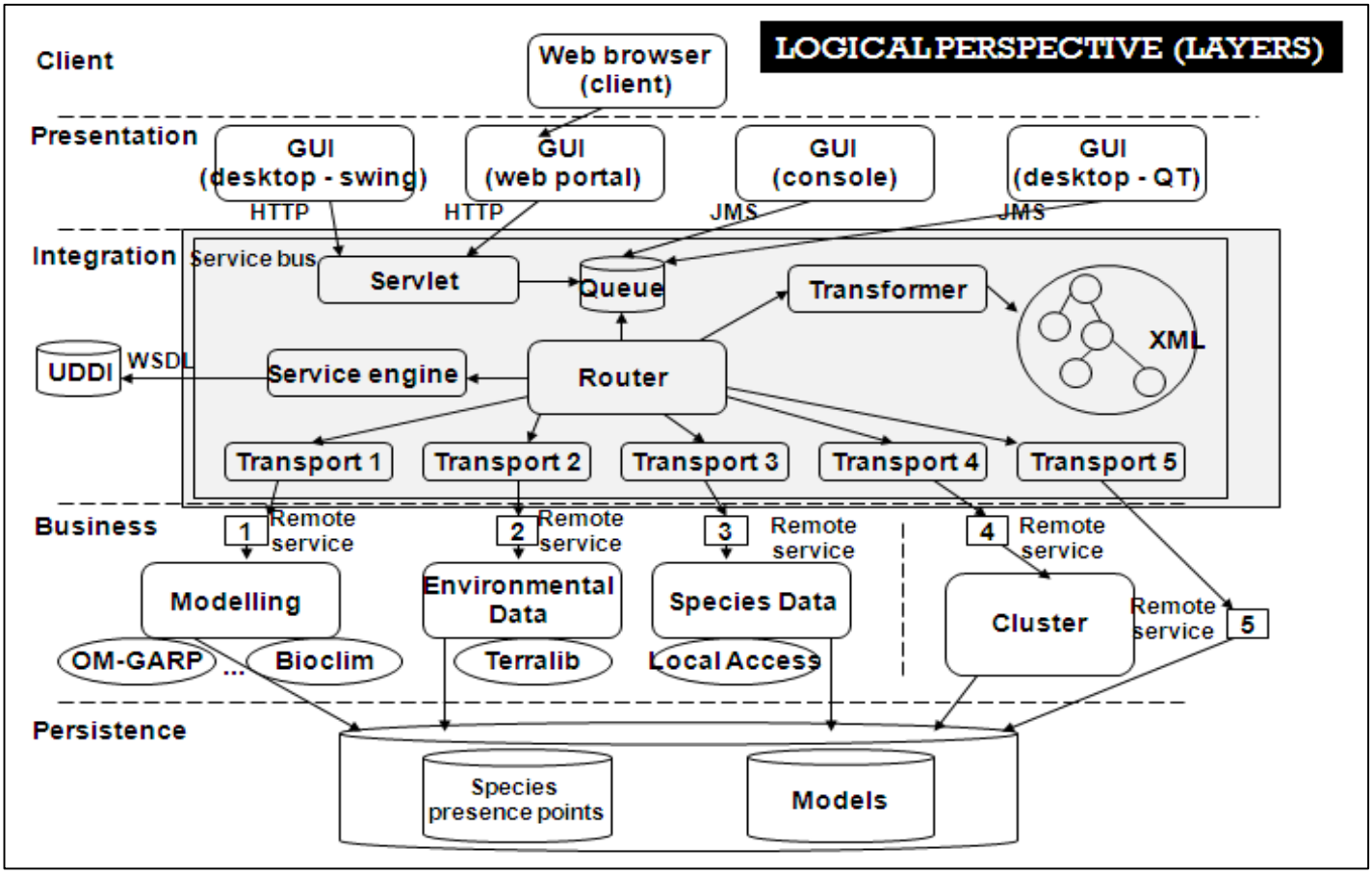


Fig. 3 – Reference architecture for Ecological Niche Modelling System.

the flexibility of constructing more than a single table, so as to be able to attend application from different domains. Finally, as the number of adaptive decision tables is not limited, by definition, an *Adaptive Services Repository* should be incorporated to the Service Engine. The result would be a Service Engine as presented in Fig. 4.

Other ESB solutions, such as openESB [https://open-esb.dev.java.net/], supported by SUN Microsystems, also have functionalities similar to the proposed Service Engine, despite presenting some architectural differences and, nowadays, representing a more complete solution, since many other features were implemented. In such a case, it is named *Application Monitor*. However, as ESB principles are the same, since ESB is mainly an architectural concept, a similar architectural proposal may be proposed to each ESB-based solution created for supporting SOA and SOC in a similar way as the presented before.

VI. Discussion and Future Works

As criteria definition may be an empirical process, conflicts and inaccuracy in the services ranking may occur. Therefore, it will probably be necessary to dedicate time and effort to construct a relatively reliable adaptive decision table to each different domain of problems that the SOA architecture intends to treat.

On the other hand, the adoption of an adaptive decision tables implemented onto ESBs may bring many choices and facilities to the user. It also may avoid misinterpretation of who is guilty when an SOA-based

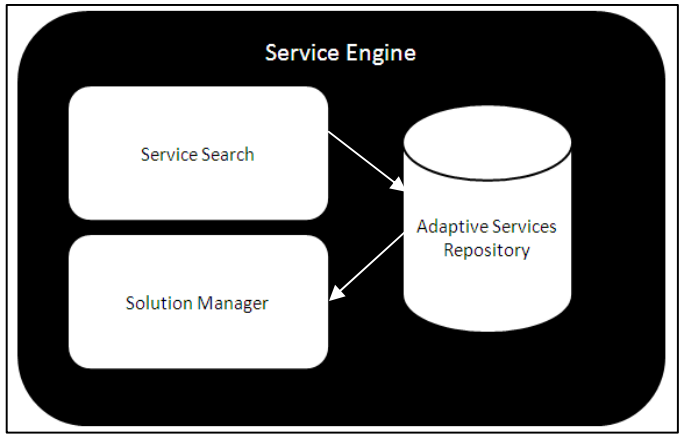


Fig. 4 – A new architectural proposal to a Service Engine

service application does not offer the desirable results, in terms of performance, security, reliability or whatever other quality criteria which be important in the specific context.

An additional advantage is that an adaptive decision table implemented onto ESBs may be used to compare the results of the execution of similar services furnished by different providers. In the situations where service providers have a significant cost, this data may be an important factor to be considered by IT executives.

Besides, as services may be executed in parallel, by using different providers, it is possible to evaluate and, if interesting, to present to the user all results obtained. This is important, for instance, in modelling applications based on non-deterministic algorithms [19].

However, to implement the proposals presented in this paper is not an easy task and, probably, will demand some effort, even after the architectural constraints were solved (and implemented). So, many future works may be proposed, starting from the problem definition.

At first, as the service ranking and selection is nowadays a very hot research area, to study and formalize an architectural proposal to incorporate the new concept of Service Engine to the ESB conceptual definition is a main challenge and many effort will be applied to this.

Solved the architectural problem, a second important point is to define the data structures to implement adaptive decision tables in ESBs. Since services are registered on UDDI or similar devices distributed on the Internet, search mechanisms need to be incorporated to maintain the portability of the solution. Then, an internal language must be defined, using some specific technique (e.g.: ontology, metadata) to enable the service inclusion in the adaptive decision tables.

Other very interesting problem is, supposing services found, how to rank them, considering services attributes. At first, it is necessary to exploit the concepts of attributes of services and, perhaps, this may require some kind of categorization. Then, after solving this problem, the next step is to find an adequate function or algorithm to service ranking or selection. Many proposals were made, but they are usually based on untrue hypothesis, such as all UDDI are oriented to ontology or only QoS factors are relevant. Nevertheless, some approaches are very interesting, and the necessary solution may be, in fact, a combination of a set of proposals, instead of choosing only one. This may depend of several factors, including each problem domain.

After designing a whole scenario, the next step is to propose a solution for a specific problem, so as to present a concept proof. In such a case, as the matter is related to other works, application in the biodiversity field may be the chosen domain, since there are already collaborators interested in evolve software techniques and whose contribution may not be disregarded.

Since a concept proof is implemented, it will need to be validate and then, only them, it is possible to imagine offering this solution as part of a real ESB, for practical research and/or commercial application.

However, as the set of problems is open in this paper and many of them may be solved in parallel, maybe contributions may improve the velocity to reach such a solution, and they will be very welcome.

VII. CONCLUSION

The implementation of adaptive decision tables associated to ESBs is a highly complex problem, but it is also a highly interesting challenge. If the proposals presented in this paper were all achieved in order to adopt the solution, probably a big step will be have given into the direction of the definitive solution to the service selection and ranking problem.

Exclusively from the adaptive viewpoint, the most relevant contribution is that the usage of adaptive decision tables to service ranking and selection. It is not a dream to imagine that, over time, only better services will be selected among many similar available solutions, and the service providers will have to review their priorities. Note that, however, the "better services" may change, since adaptive technologies will make regular re-rankings in the services organization and evaluation criteria.

Finally, adaptive techniques may completely transform the architectural concept of ESB and SOA and, perhaps, even the SOC concept itself, since the constant re-evaluation of the services may lead to a way to evolve the services over time.

ACKNOWLEDGMENTS

Authors are grateful to FAPESP – Fundação de Amparo à Pesquisa do Estado de São Paulo – Brazil, for the support to the openModeller (04/11012-0) and BioAbelha (04/15801-0) projects.

REFERENCES

[1] BASS, L., CLEMENTS, P., KAZMAN, R., 2003. Software Architecture in Practice. Second Edition, Addison-Wesley Professional.

[2] BRAVO, C., NETO, J.J., SANTANA, F.S., SARAIVA, A.M., 2007. *Towards and adaptive implementation of genetic algorithms*. Latin American Workshop on Biodiversity Informatics – INBI 2007 / CLEI 2007 – XXXIII Conferência Latino Americana de Informática. 9–12 Outubro. San José, Costa Rica.

[3] BROOKS, F. P. 1987. No Silver Bullet, IEEE Computer, Vol. 20-4. 1987.

[4] CASATI, F., CASTELLANOS, M., DAYAL , U., SHAN, M. 2004. Probabilistic, Context-Sensitive, and Goal-Oriented Service Selection. ICSOC'04, November 15–19, 2004, New York, New York, USA, pp 316-321.

[5] ENDREI, M. et al., Newling, T. 2004. Patterns - Service-Oriented Architecture and Web Services. IBM Redbook.

[6] GILDERSLEEVE, T. R. 1970. *Decision Tables and Their Practical Application in Data Processing*. Englewood Cliffs, N. J., Prentice Hall.

[7] HOLLAND, J. H. 1975. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor.

[8] HUGHES, M. L., SHANK, R. M., STEIN, E. S. 1968. Decision Tables. Midi Publications, Management Development Institute, Divisions of Information, Industries, Inc., Wayne, Pennsylvania.

[9] HUHNS, M., SINGH, M.P., 2005. Service-Oriented Computing: Key Concepts and Principles. IEEE Internet Computing, v.9, n.1, p.75-81.

[10] LIU, Y., NGU, A. H.H., L. 2004. QoS Computation and Policing in Dynamic Web Service Selection. WWW2004, May 17–22, New York, New York, USA. ACM 1-58113-912-8/04/0005, pp 66-73.

[11] MACKENZIE, C. M. et al., 2006. Oasis Reference Model for Service Oriented Architecture 1.0.

[12] MAXIMILIEN, E. M., SINGH, M. P. 2005. Multiagent System for Dynamic Web Services Selection. In Proceedings of the AAMAS Workshop on Service-Oriented Computing and Agent-Based Engineering (SOCABE), Utrecht, July.

[13] MENASCÉ, D. A. . QoS Issues in Web Services. 2002. IEEE Computer, pp 72-75.

[14] MENGE, F. 2007. Enterprise Service Bus. Free and open source software conference, pp 01-06

[15] NETO, J. J. 2001. *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, July 23-25, Springer-Verlag, 2001, pp. 234-250.

[16] NETO, J. J. 2007. *Um Levantamento da Evolução da Adaptatividade e da Tecnologia Adaptativa***.** Revista IEEE América Latina. Vol. 5, Num. 7, ISSN: 1548-0992, Novembro, pp. 496-505.

[17] POLLACK, S.L., HICKS, H.T., and HARRISON, W.J. 1971. *Decision Tables: Theory and Practice*, Wiley, New York.

[18] SANTANA, F. S, MURAKAMI, E., SARAIVA, A. M., BRAVO, C., CORREA, P. L. P. 2007. Uma arquitetura de referência para sistemas de informação para modelagem de nicho ecológico. In: 6º Congresso Brasileiro de Agroinformática – SBIAgro.

[19] SANTANA, F. S, SIQUEIRA M. F., SARAIVA, A. M. & CORREA, P. L. P. 2008. *A reference business process for ecological niche modeling*. Ecological Informatics v. 3, n. 1, p. 75-86.

[20] STAL, M., 2002. Web Services: Beyond Component-Based Computing. Communications of the ACM. Vol. 45, No. 10, 71-76.

Aplicação de Tecnologia Adaptativa em Redes de Sensores Sem Fio (05 Janeiro 2009)

L. Gonda, C. Cugnasca, A. A. de Castro Jr., J. J. Neto

***Resumo*— Devido ao avanço de diversas tecnologias as Redes de Sensores Sem Fio têm se tornado uma importante tecnologia no monitoramento de diversas aplicações, tais como militar, agricultura, monitoramento de pacientes e outras. Devido à limitação de recursos, essas redes, ao contrário das redes tradicionais, trabalham de maneira colaborativa e os protocolos de comunicação devem se preocupar não só com o transporte de dados, mas também com o consumo de energia na transmissão. Além disso, as RSSF devem possuir a capacidade de auto-organização, através de mecanismos de estabelecimento, prevenção e manutenção da rede, que impõem um mecanismo de funcionamento dinâmico da rede. Este trabalho apresenta uma proposta inicial de aplicação de Tecnologia Adaptativa no desenvolvimento de algoritmos para RSSF.**

Palavras chave — Redes de sensores sem fio, Dispositivos adaptativos, Manutenção de Topologia.

I. INTRODUÇÃO

UMA Rede de Sensores Sem Fio (RSSF) é composta por diversos módulos distribuídos, denominados de nós sensores, que possuem capacidade de sensoriamento, processamento e comunicação [1]. Esta tecnologia foi viabilizada pela evolução de diversas áreas, em especial comunicação sem fio, microprocessadores e dispositivos eletromecânicos, possibilitando que pequenos dispositivos, dispostos em uma área a ser monitorada, pudessem reproduzir, através de sensoriamento, processamento e comunicação com outros nós, fenômenos físicos com mais precisão [2]. Diversas aplicações para RSSF estão surgindo, incluindo medicina, agricultura, controle de ambientes, militar, entre outras [3].

Em geral, os recursos dos nós sensores apresentam limitação em relação à capacidade de processamento, armazenamento, de energia e de alcance do seu rádio de comunicação. Conseqüentemente, os nós sensores devem ser agrupados para executarem tarefas de maneira colaborativa, pois individualmente não são muito eficazes.

Uma das características essenciais em uma RSSF é a capacidade de auto-organização. Dessa forma, ao serem espalhados em grandes quantidades em uma área de monitoramento, os nós sensores devem se organizar automaticamente, para que seja estabelecida a comunicação entre os diversos dispositivos da rede [3]. É importante ressaltar que a auto-organização é necessária, não somente no instante em que a rede está sendo criada, mas também na manutenção da rede, devido a possíveis mudanças em decorrência de falhas e inserção de novos nós sensores.

Com a finalidade de manter as funções da rede e economizar recursos, especialmente energia, a utilização de mecanismos para manutenção e prevenção a falhas são extremamente importantes em protocolos de comunicação para RSSF. Neste sentido, o objetivo deste trabalho é apresentar uma proposta inicial da Tecnologia Adaptativa (TA) no desenvolvimento de mecanismos para RSSF.

O restante deste trabalho está organizando conforme descrito a seguir. A Seção II mostra alguns trabalhos relacionados à comunicação e manutenção em RSSF. Na Seção III são apresentadas as definições de grafos dinâmicos e dispositivos adaptativos, bem como características de RSSF que demonstram a necessidade de um dinamismo em suas organização e topologia, o que pode viabilizar a aplicação de Tecnologia Adaptativa. Por fim, na Seção IV são apresentadas as considerações finais do trabalho.

II. TRABALHOS RELACIONADOS

Em uma RSSF, os nós sensores têm como função coletar dados de um ambiente a ser monitorado e enviar essas informações para um nó especial com a função de *gateway*, também denominado de sorvedouro (*sink*), através de um mecanismo múltiplas transmissões/saltos (*multihop*). Em geral este nó possui mais recursos de processamento e memória e é responsável pelo processamento das informações da rede [1].

Na transmissão de dados em uma RSSF, ao contrário do que acontece nas redes tradicionais, como a Internet, por exemplo, além de se preocupar com o envio de informações, os protocolos de comunicação devem levar em consideração também o consumo de energia, já que na maioria das vezes os nós sensores são alimentados por baterias, que possuem carga limitada [2].

Devido à limitação de recursos nos nós sensores, o estudo de mecanismos e protocolos para gerenciamento em RSSF, tais como, manutenção de topologia em RSSF, cobertura da área a ser monitorada e redução de falhas são importantes para prolongar o funcionamento da rede, sem que haja nenhuma intervenção externa, principalmente, em casos em que os nós sensores estão dispostos em locais de difícil acesso. Diversos mecanismos e protocolos foram descritos na literatura, conforme apresentado a seguir.

Sun *et al.* [4] apresentam um mecanismo denominado REMUDA, que tem por objetivo a construção da topologia da rede, envio de informações manutenção de topologia em RSSF. Neste mecanismo, inicialmente o nó

sorvedouro envia uma mensagem de construção de topologia utilizando *broadcast* para a rede, contendo o contador de saltos (*hop count*), cujo valor inicial é zero, e a identificação do nó sensor (*Node ID*). Ao receber uma informação de controle de topologia, cada nó sensor espera um tempo pré-definido por mais mensagens e armazena o remetente de todas as mensagens como vizinhos. Por fim, cada nó seleciona um de seus vizinhos como pai, incrementa seu contador de saltos e envia a mensagem de construção de topologia (*broadcast*). Os critérios para a escolha do nó pai, em ordem de prioridades são: a qualidade do sinal, o contador de saltos (quanto menor o contador de saltos melhor) e o a identificação do nó. Dessa forma, o protocolo cria uma topologia em árvore.

Toda vez que um nó sensor deseja enviar uma informação coletada, este deve encaminhá-la para o nó pai, com a finalidade de que a mesma chegue até o nó sorvedouro.

Para a manutenção da topologia REMUDA, cada nó sensor envia periodicamente informações de controle para seus pais. Se as mensagens de controle não são respondidas satisfatoriamente, de acordo com os parâmetros ajustados para o algoritmo, o nó sensor inicia o processo de escolha do novo pai, enviando *broadcast* para detectar possíveis vizinhos. Em seguida, escolhe, dentre os nós que responderam sua mensagem, um novo nó pai.

Outro mecanismo de disseminação de dados é apresentado por Figueiredo, Nakamura e Loureiro [5]. Este mecanismo é denominado de Multi e corresponde à utilização alternada de dois outros protocolos: SID (*Source-initiated Dissemination*) e EF-Tree (*Earliest First Tree*). O protocolo SID é utilizado em RSSF onde as aplicações são baseadas em eventos e o tráfego na rede é esporádico. Dessa forma, toda vez que um nó sensor precisa enviar informações coletadas em decorrência de algum evento, deve ser criado uma rota deste nó até o nó sorvedouro. Para a construção do caminho, o nó possui informações a serem transmitidas e envia um *broadcast* na rede. Esta mensagem é propagada pela rede até que chegue ao nó sorvedouro e seja determinada uma rota para o envio de dados. O protocolo EF-Tree mantém uma estrutura de comunicação em árvore, que é construída a partir do nó sorvedouro, que envia uma mensagem de *broadcast* para a construção da topologia, que é propagada pelos demais nós da rede. Cada um dos nós sensores da rede selecionam o primeiro nó do qual recebeu a informação de controle como nó pai na árvore de roteamento. Periodicamente, este algoritmo é executado para a manutenção da rede. O algoritmo Multi proposto em [5] utiliza alternadamente os algoritmos SID e EF-Tree, de acordo com o estado da rede. Inicialmente, o algoritmo SID é escolhido para transmissão de dados até o nó sorvedouro. Porém, caso o tráfego de dados na rede se torne muito intenso, de acordo com parâmetros a serem configurados, o algoritmo Multi passa a utilizar o protocolo EF-Tree.

III. Grafos Dinâmicos – Proposta Adaptativa

A. *Dispositivos Adaptativos*

Um dispositivo formal é dito adaptativo sempre que o seu comportamento muda dinamicamente, em resposta à estímulos de entrada, sem a interferência de agentes externos, mesmo seus usuários.

Em sua formulação geral, os dispositivos adaptativos são caracterizados através de duas camadas conceituais: o formalismo subjacente e o mecanismo adaptativo [6]. O formalismo subjacente é representado por um dispositivo guiado por regras (formalismo convencional, não-adaptativo) e opera, movendo-se de uma configuração para outra, sucessivamente, em resposta a estímulos recebidos em sua entrada. Tais movimentos são definidos por um conjunto de regras que mapeiam cada possível configuração em uma configuração seguinte correspondente. O mecanismo adaptativo responde pelas mudanças incrementais no comportamento do dispositivo subjacente. As propriedades de auto-modificação são representadas pelas funções e ações adaptativas, capazes de alterar o conjunto de regras do dispositivo subjacente, determinando assim, conseqüentemente, seu comportamento a partir de então.

Para isso, a cada uma de suas regras de formação, podem ser associadas ações adaptativas, que têm a função de tornar o dispositivo auto-modificável. Em outras palavras, as eventuais mudanças no comportamento do dispositivo devem ser conhecidas, em qualquer fase do funcionamento em que sejam efetuadas. Portanto, um dispositivo adaptativo deve ser capaz de detectar todas as situações que possam provocar alterações e reagir adequadamente, impondo as alterações correspondentes ao comportamento do dispositivo. O comportamento desta classe de dispositivos formais baseia-se no funcionamento de um formalismo subjacente (não-adaptativo), descrito por algum conjunto finito de regras de formação. Um dispositivo adaptativo pode ser obtido por ações adaptativas inerentes às regras do formalismo subjacente, de modo a que, sempre que uma regra é aplicada, a ação adaptativa associada a ela é executada, provocando as mudanças correspondentes sobre o conjunto de regras do formalismo subjacente (não-adaptativo).

B. *Grafos Dinâmicos*

Na maioria das aplicações de algoritmos em grafos, os grafos são dispositivos estáveis que sofrem pouca ou nenhuma mudança. Entretanto, variações que menos estáveis desses dispositivos formais, que possam sofrer alterações em sua estrutura, tais como inserção e remoção de arestas e vértices, vêm sendo alvo de estudos recentes, demonstrando o crescente interesse no desenvolvimento de aplicações que façam uso de formalismos dinâmicos. Nesta seção, definimos os grafos dinâmicos com o objetivo de utilizá-los como ferramentas para a modelagem de soluções

para problemas relacionados às RSSFs.

De acordo com a definição de Neto[6], um dispositivo adaptativos que usa um grafo como formalismos subjacente (não-adaptativo) pode ser definido como uma 5-upla:

$$ND = (C, NR, S, c_0, A)$$

Onde:

- ND corresponde ao grafo que modela a RSSF em questão;
- C é o conjunto de possíveis configurações do grafo que modela a RSSF. E $c_0 \in C$ é sua configuração inicial, ou seja, a configuração que é montada inicialmente com base nas regras de conexão da RSSF;
- NR é o conjunto de regras que definem o grafo ND pela relação $ND \subseteq C \times S \times C$. As regras $r \in NR$ tem a forma $r = (c_i, s, c_j)$, indicando que, em resposta a um estímulo de entrada $s \in S$, r altera a configuração corrente de c_i para c_j e "consome" o símbolo s. Neste caso, a mudança de configuração representa uma alteração na topologia da RSSF;
- S é um conjunto (finito) de todos os possíveis eventos válidos e considerados como estímulos de entrada para ND, com $\varepsilon \in S$.
- $A \subseteq C$ (resp. $F = C - A$) é dito subconjunto de configuração de aceitação (resp. falha), que corresponde a uma topologia aceita para a RSSF em questão;
- ε denota "vazio", e representa o elemento NULO do conjunto ao qual pertence;

O mecanismo adaptativo é composto pelas operações que promovem a modificação da topologia da rede. Entre os prováveis fatores que podem influenciar nestas alterações estão a falha de um nó, o desligamento temporário de um nó (economia de energia), a inclusão de novos nós, a mudança de local do nó (a interconexão entre os nós é influenciada pela distância), os ajustes para melhorar o desempenho da rede (nós com pouca energia diminuem sua capacidade de comunicação), entre outros.

IV. DESCRIÇÃO DO ALGORITMO PROPOSTO

O algoritmo a ser proposto é composto de duas fases principais: construção da topologia inicial e manutenção da topologia utilizando TA, através de regras adaptativas. Estes mecanismos serão utilizados para reduzir o consumo de energia, aumentando o tempo de vida da rede.

A. Construção da Topologia Inicial

Esta fase do algoritmo é responsável por determinar a topologia inicial da RSSF, de maneira que as informações coletadas pelos nós possam ser encaminhadas ao nó *gateway*. Inicialmente existe um conjunto de vértices que estão isolados (Figura 1(a)). Como o objetivo é fazer com que as informações possam ser enviadas para o *gateway*, este nó é responsável por enviar uma mensagem do tipo *inicio* na RSSF. Esta mensagem é responsável por detectar os nós existentes, bem como, determinar quais nós participarão da RSSF inicialmente. Ao receber uma mensagem do tipo *inicio*, cada nó escolhe o nó remetente como pai (cria-se uma aresta entre os dois nós) e encaminha esta mensagem através de *broadcast*. É importante observar que um nó pode receber a mensagem de *inicio* de mais de um nó. Neste caso, o nó apenas cria uma aresta entre o nó emissor e ele mesmo, porém coloca-a como inativa (neste caso a mensagem *inicio* não é enviada para os demais nós). Na Figura 1(b) é apresentada uma possível configuração após a execução do algoritmo de construção de topologia.

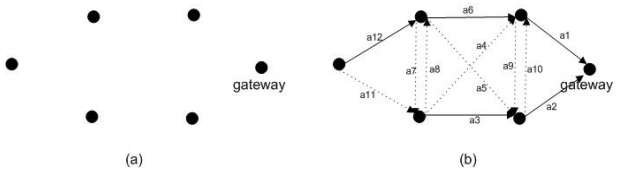


Figura 1. (a) Nós antes da execução da fase de construção de topologia. (b) Uma possível configuração c_0 da RSSF após a execução do algoritmo de construção de topologia.

Na Figura 1(b) as arestas representam que os nós vizinhos possuem algum tipo de conexão, sendo que as arestas pontilhadas indicam conexões inativas e as demais arestas indicam comunicações ativas. É importante observar que o nó *gateway* só possui arestas que chegam e não possuem arestas que saem dele.

Ao final desta etapa é construído um grafo direcionado, cujas regras de funcionamento são adaptativas, possibilitando que arestas sejam adicionadas e retiradas.

B. Manutenção da Topologia da Rede

Apesar dos nós de uma RSSF serem estáticos, mudanças na topologia da rede podem ocorrer durante o tempo de vida da rede. Entre os fatores que podem influenciar estas mudanças, podem ser citados: falha de nós, desligamento temporário de um nó, inclusão de novos nós e mudanças na localização de nós.

Inicialmente pode-se considerar como ND o conjunto de vértices e arestas do grafo apresentado na Figura 1(b). Como serão consideradas apenas falhas de nós, o conjunto S é composto de todas as possíveis falhas de todos os vértices.

Sempre que um nó da rede falha é necessário que regras adaptativas sejam executadas e ações adaptativas sejam tomadas para que a topologia da rede possa ser reconstruída. Para cada um dos vértices da rede, são armazenados, além do pai, os outros vértices adjacentes (arestas pontilhadas na Figura 1(b)), evitando que trocas de mensagens para a escolha de um novo nó pai sejam realizadas quando a falhas de nós sejam detectadas. Além da escolha de novos nós pais, ações adaptativas são necessárias para que a nova topologia da rede seja reestabelecida, ou seja, as regras adaptativas de cada vértice vizinho ao nó que falhou devem ser atualizadas.

Para ilustrar o funcionamento, considere a Figura 2.

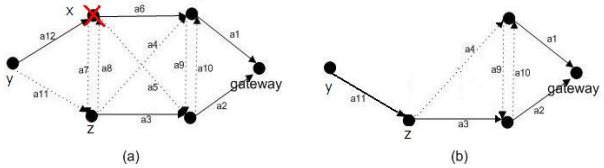


Figura 2. (a) Falha no em um nó da rede. (b) Recuperação da falha do nó x.

Suponha que ocorra uma falha no nó x, Dessa forma, o nó y deve escolher um novo nó pai e o vértice z deve atualizar suas regras.

Para a implementação das regras é possível utilizar uma matriz de adjacências para denotar as relações de vizinhança e parternidade entre os nós e um conjunto de regras associadas à matriz. As regras associadas a essa matriz pertencem a NR.

V. CONCLUSÃO

A tecnologia de RSSF tem se tornado cada vez mais presente e importante em aplicações de monitoramento nas mais diversas áreas. Entretanto, a limitação de recursos, em especial, a energia é um fator que vem se destacando, principalmente, no desenvolvimento de algoritmos de comunicação que permitam uma melhor utilização dos recursos de bateria, sem que haja perdas na comunicação. Além da restrição de energia, um dos fatores que pode influenciar na comunicação em RSSF é o dinamismo da topologia, devido a mecanismos de prevenção e manutenção que são inerentes ao comportamento da rede. Dessa forma, mapear este comportamento dinâmico das RSSF em um dispositivo adaptativo é um desafio extremamente importante do ponto de vista de aplicação da TA, pois pode garantir uma melhor utilização de recursos de energia durante a comunicação. Esta tarefa consiste basicamente em mapear as componentes de uma RSSF nas componentes de um dispositivo adaptativo.

Portanto, a manutenção da topologia pode ser realizada através do mapeamento da mesma em regras adaptativas. Além do controle de topologia, é provável também que problemas de cobertura também possam ser realizados através de tecnologia adaptativa.

REFERÊNCIAS

[1] I. F. Akyildz et al. "Wireless Sensor Networks: a survey", *Computer Networks*, 2002, v. 38, n. 4, pp. 393-422.

[2] D.Cueller, D. Estrin, M. Srivastava. "Overview of Sensor Networks". IEEE Computer Society, 2004, pp. 41-49.

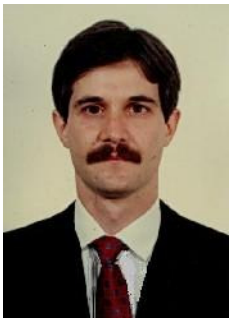
[3] P. Baronti et al. "Wireless Sensor Networks: a survey on the state of the art and the IEEE 802.15.4 and Zigbee Standards", Computer Communications, 2007, v.30, pp. 1655-1695.

[4] L. Sun, T. Yan, Y. Bi. "REMUDA: A Practical Topology Control and Data Forwarding Mechanism for Wireless Sensor Networks", ACTA AUTOMATICA SINICA, 2006, v. 36, n. 6, pp. 867-874.

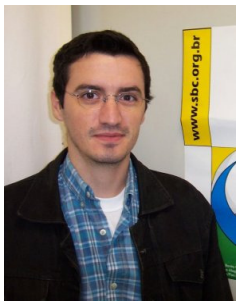
[5] J. J. Neto, "Adaptative rule-driven devices - general formulation and a case study," in CIAA'2001 Sixth International Conference on Implementation and Application of Automata, Pretoria, South Africa, July 2001, pp. 234–250.

[6] C. M. S. Figueiredo, E. F. Nakamura, A.A.F. Loureiro. "Protocolo Adaptativo Híbrido para Disseminação de Dados em Redes de Sensores Sem Fio Auto Organizáveis", in SBRC'2004 XXII Simpósio Brasileiro de Redes de Computadores, Gramado-RS, Brazil, Maio 2004, pp.

Luciano Gonda é graduado em Ciência da Computação pela Universidade Federal de Mato Grosso do Sul (1999) e mestre em Ciência da Computação pela Universidade Federal de Mato Grosso do Sul (2004). Atualmente, é aluno de doutorado da Escola Politécnica da USP, sob a orientação do Prof. Carlos Eduardo Cugnasca. É membro do Laboratório de Automação Agrícola e do Grupo de Pesquisa em Engenharia e Computação da UCDB e professor-pesquisador da Universidade Católica Dom Bosco (UCDB). Tem experiência na área de Ciência da Computação, e seus interesses em pesquisa concentram-se em Redes de Sensores Sem Fio, em especial, aplicações agrícolas e controle de ambientes, Algoritmos Paralelos e Distribuídos e Redes de Computadores.



Carlos Eduardo Cugnasca é graduado em Engenharia de Eletricidade (1980), mestre em Engenharia Elétrica (1988) e doutor em Engenharia Elétrica (1993). É livre-docente (2002) pela Escola Politécnica da Universidade de São Paulo (EPUSP). Atualmente, é professor associado da Escola Politécnica da Universidade de São Paulo, e pesquisador do LAA - Laboratório de Automação Agrícola do PCS - Departamento de Engenharia de Computação e Sistemas Digitais da EPUSP. Tem experiência na área de Supervisão e Controle de Processos e Instrumentação, aplicadas a processos agrícolas e Agricultura de Precisão, atuando principalmente nos seguintes temas: intrumentação inteligente, sistemas embarcados em máquinas agrícolas, monitoração e controle de ambientes protegidos, redes de controle baseados nos padrões CAN, ISO11783 e LonWorks, redes de sensores sem fio e computação pervasiva. É editor da Revista Brasileira de Agroinformática (RBIAgro).



Amaury Antônio de Castro Junior é graduado em Ciência da Computação pela Universidade Federal de Mato Grosso do Sul (1997) e mestre em Ciência da Computação pela Universidade Federal de Mato Grosso do Sul (2003). Atualmente, é aluno de doutorado da Escola Politécnica da USP, sob a orientação do Prof. João José Neto. É membro do Laboratório de Linguagens e Técnicas Adaptativas e do Grupo de Pesquisa em Engenharia e Computação da UCDB e professor assistente da Universidade Federal de Mato Grosso do Sul (UFMS), Campus de Coxim (CPCX). Tem experiência na área de Ciência da Computação, com ênfase em Teoria da Computação, atuando nos seguintes temas: Tecnologias Adaptativas, Autômatos Adaptativos, Projeto de Linguagens de Programação e Modelos de Computação.

João José Neto é graduado em Engenharia de Eletricidade (1971), mestre em Engenharia Elétrica (1975) e doutor em Engenharia Elétrica (1980). É 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 baseados em tecnologia adaptativa.

Alinhamento de duas cadeias usando um transdutor adaptativo

(Novembro 2008)

J. Kinoshita e R. L. A. Rocha

***Resumo*— O algoritmo clássico que alinha duas cadeias (strings) s1 e s2 de tamanho m e n é seqüencial e da ordem de m x n. Nesse artigo propomos o uso de um transdutor adaptativo para resolver o mesmo problema que é baseado no algoritmo clássico, mas pode ser paralelizado porque funções adaptativas podem ser disparadas assim que prontas.**

***Palavras chave*— Autômatos adaptativos (*adaptative automata*), alinhamento de cadeias (*string alignment*), biologia computacional (*computational biology*).**

I. Introdução

Um algoritmo bastante utilizado ao se trabalhar com strings é o de se buscar uma string em um texto (ou uma substring em uma string). Um editor de texto simples deve prover ferramentas para se resolver esse problema e diversos algoritmos já haviam sido propostos na década de 70, porém devido ao aumento de textos em formatos eletrônicos que ocorrem na Web ou que representam seqüências de genes, surgiu um grande interesse em se resolver o problema de se buscar uma string em um texto sendo que a string poderia estar corrompida. Esse tipo de problema ocorre no plágio de uma música: é possível se identificar uma string (uma seqüência de notas musicais) mesmo que haja algumas distorções. Para se resolver esse tipo de problema, houve a necessidade de se usar uma função de distância entre duas strings e dentre várias possíveis, a função de distância de edição proposta por Levenshtein em 1965 ganhou notoriedade. A seguir explicamos a função de distância de edição, o algoritmo clássico que resolve essa função e nossa proposta de um algoritmo baseado em um dispositivo adaptativo.

II. Função Distância de Edição

Usando um editor podemos transformar uma string (a primeira string) em outra (a segunda string) realizando operações sobre cada caractere da primeira string. As operações sobre cada caractere podem ser apenas as 4 operações:

R: *replace* - troca um caractere por outro
I: *insert* - insere um caractere na segunda string. Como ênfase: não existem operações I sobre caracteres da primeira string.
D: *delete* - apaga o caractere da primeira string.
M: *match* - passa o caractere de entrada para a cadeia de saída.

Um exemplo de se editar "gato" e obter "gente" usando essas operações é a seguinte:

```
DDDDIIIII
gato
    gente
```

que significa: *delete* (D) 'g', *delete* 'a' , ou seja, apague todos os caracteres da primeira string e insira (I) todos os caracteres da segunda string.

Segundo Gusfield [1]:

A distância de edição entre duas strings é definida como o menor número de operações de edição: I, R, e D necessárias para transformar a primeira string na segunda. Para efeito de ênfase, observe que Ms não são contados.

Para medirmos uma distância entre as duas strings, associamos um número a cada operação: R-1,M-0,D-1,I-1. A função de distância consiste na soma de todos os valores referentes às operações R, M , D, e I. Como no exemplo que transforma "gato" em "gente" realizamos 4 Ds e 5 Is então a distância, segundo essa edição é de 9. Porém 9 não é o menor número de edições que transformam "gato" em "gente".

Uma outra forma de editar "gato" e obter "gente" é:

```
MRIMR
ga_to
gente
```

Nesse caso, a distância de edição é 3 que é a mínima distância de edição possível entre essas duas strings. As operações de edição são:

M(g) : *match* - mantém g.
R(a,e): *replace* - troque a por e.
I(n): *insert* - Insira n
etc.

A forma clássica de se descobrir a distância de edição é através da programação dinâmica [1]. Nesse algoritmo é montada uma tabela de n+1 linhas por m+1 colunas onde:

m = número de caracteres da primeira string
n = número de caracteres da segunda string.

Essa tabela é preenchida de forma a se obter o menor

J. Kinoshita, R. L. A. Rocha são professores do Departamento de Engnharia de Computação e Sistemas Digitais da Escola Politécnica da Universidade de São Paulo. (emails: jorge.kinoshita@poli.usp.br; luis.rocha@poli.usp.br).

número de operações que transformam uma string em outra. No exemplo, a distância é 3, mas esse valor pode estar associado a uma outra edição:

```
MIRMR
g_ato
gente
```

Ou seja, podem existir várias edições que correspondem à distância mínima de edição.

Existem diversas aplicações para a função de distância de edição [2]:

- correção ortográfica
- reconhecimento de fala
- análise de DNA
- detecção de plágio.

Descobrir a distância de edição entre duas strings, corresponde a uma forma de alinhar as duas strings [1].

III. O ALGORITMO CLÁSSICO

Uma explicação mais detalhada do algoritmo pode ser vista em [1]. O algoritmo usa programação dinâmica e consiste em preencher uma tabela de (n+1)×(m+1) posições onde m é o número de caracteres da primeira string e n é o número de caracteres da segunda string. A tabela possui as linhas variando de zero a m e colunas de zero a n.

A string s1 corresponde a s1[1..m], ou seja, à substring que começa na posição 1 e termina na posição m e s2 corresponde a s2[1..n].

A célula [i,j] da tabela consiste na distância D(i,j) de edição entre a substring s1[1..i] e a substring s2[1..j].

A base para a programação dinâmica é feita preenchendo os valores das células:

D(0,j) corresponde ao menor número de operações que transformam a substring s1[1..0], a cadeia vazia, em s2[1..j]. O valor a ser colocado nessas células é j porque é necessário j Inserções para transformar o vazio em s2[1..j].

D(i,0) corresponde a transformar a s1[1..i] na cadeia vazio e para isso são necessários i Deletes sobre s1[1..i].

A distância D(i,j) entre s1[1..i] e s2[1..j] é calculada a partir do trabalho para se fazer o alinhamento de substrings com 1 caractere a menos (pois as operações são sempre sobre um caractere). D(i,j) pode ser:

- D(i,j-1) + 1: porque para transformar s1[1..i] em s2[1..j] já sabendo como transformar s1[1..i] em s2[1..j-1] basta uma operação (inserir o caractere s2[j] em s2[1..j-1])
- D(i-1,j) + 1: porque para transformar s1[1..i] em s2[1..j] já sabendo como transformar s1[1..i-1] em s2[1..j] basta uma operação (deletar o caractere s1[i]).
- D(i-1,j-1) + t(i,j) onde: se (s1[i] = s2[j]) então t(i,j) = 1, senão t(i,j) = 0. Isso corresponde à operação de Replace (trocar o caractere s1[i] por s2[j]) ou Match (repassar o caractere s1[i] para s2[j]).

O valor a ser preenchido em D(i,j) é o menor valor possível dentre os 3 valores.

A tabela vai sendo preenchida até que se obtém D(n,m).

Aplicando isso às strings "gato" e "gente" obtemos a tabela 1.

TABELA 1
APLICAÇÃO DO ALGORTIMO CLÁSSICO PARA O ALINHAMENTO DE "GATO" E "GENTE".

	0	1-g	2-a	3-t	4-o
0	0	1	2	3	4
1-g	1	0	1	2	3
2-e	2	1	1	2	3
3-n	3	2	2	2	3
4-t	4	3	3	2	3
5-e	5	4	4	3	3

O algoritmo pode ser feito de forma a se preencher uma linha de cada vez, à medida que chegam os caracteres de s2.

O número D(x,y) escolhido em cada célula (x,y) é o menor dentre 3 valores. Por exemplo, D(1,1) que corresponde a transformar a string "g" na string "g" poderia ser obtido de 3 formas:

- A partir de s1 = "g" e s2 = vazio D(0,1), para se obter "g" em s2 deveria haver uma inserção de "g" em s2. O valor de D(1,1) nesse caso seria de 1+ 1 = 2. Isso corresponderia a duas operações "Deletar g de s1 (daí D(0,1) =1) e depois inserir g em s2 (D(1,1) =2)".
- A partir de s1 = "" e s2 = "g", para se obter "g" em s2 deveria se apagar "g" de s1. Nesse caso, isso corresponderia a duas operações: "Inserir "g" em s2 (D(1,0)=1)e depois deletar g em s1".
- A partir de s1 ="" e s2 = "", para se obter g basta fazer Match, ou seja, repassar o "g" de s1 para s2 com peso zero.

Para se descobrir o alinhamento entre "gato" e "gente" basta observar quais foram as operações que levaram a D(n,m) no caso D(5,4) e ir traçando o caminho até D(0,0). Às vezes existe mais de um caminho para se chegar a uma célula. Por exemplo, para se chegar em D(3,2) pode-se ir por D(2,1) e D(2,2). Neste exemplo existem dois caminhos possíveis gerando a distância de edição 3 gerando os dois possíveis alinhamentos mostrados no item I.

IV. O TRANSDUTOR ADAPTATIVO APLICADO À DISTÂNCIA DE EDIÇÃO

Um autômato normal é composto de estados e transições. A cada transição está associado um caractere do alfabeto. Uma transição efetiva uma mudança na configuração do autômato por trocar de um estado a outro e consumir o caractere de entrada. Supondo que não exista uma transição capaz de consumir o caractere de entrada então a cadeia de entrada não é reconhecida.

Um autômato adaptativo é diferente. Ele pode estar preparado para se adaptar a situações não previstas e para isso ele utiliza uma função adaptativa. Caso não haja uma transição a ser disparada, o autômato adaptativo pode criar essa transição.

Um transdutor adaptativo é semelhante ao autômato adaptativo, mas gera uma saída ao se realizar a transição.

O algoritmo para o alinhamento pode ser reescrito usando transdutores adaptativos. A idéia é criar um transdutor que gere como resposta às operações de alinhamento para se obter s2 a partir de s1. O algoritmo vai funcionar de forma semelhante ao algoritmo clássico.

O transdutor vai ser representado por 2 elementos: estados e transições.

Ao se usar autômatos, na nomenclatura normal, não associamos valores a cada estado. Mas aqui, vamos associar a cada estado E, duas funções: a função de distância de edição d(E) e a função de pronto para disparar p(E).

O transdutor adaptativo começa com uma matriz de estados (n+1)×(m+1) e nenhuma transição entre os estados. Todos os estados estão inabilitados para o disparo a menos do estado (0,0). O estado (0,0) pede autorização para criar transições para seus estados vizinhos. À medida que isso ocorre, seus vizinhos se tornam habilitados a disparar transições e o processo continua até o estado (n,m) estiver habilitado ao disparo. Nessa situação, é possível se ter o alinhamento entre s1 e s2.

O transdutor adaptativo é composto de:

1) estados

Geralmente um estado é representado por um número, mas em nosso caso, vamos representar o estado por dois números correspondentes à célula da tabela usada no algoritmo clássico. Assim, teremos os estados de (0,0) a (n,m).

A cada estado associamos:

- função distância de edição

A "distância de edição", d(E) que corresponde ao menor número de operações de edição para transformar a string s1[1..j] na string s2[i..i]. A princípio ela é inicializada com o valor máximo i+j. O valor vai sendo reduzido à medida que transições são criadas para E. Ao final do algoritmo, os valores de d(E) correspondem a D(E) como no exemplo da tabela 1.

- função pronto

Um estado E pode ser destino de no máximo 3 transições e origem de no máximo 3 transições.

Um estado (i,j) pode criar no máximo 3 transições para os estados (i+1,j), (i,j+1) e (i+1,j+1). Se ele estiver na borda haverá menos estados vizinhos. Por exemplo, o estado (n,m) não possui vizinhos com quem se ligar. Da mesma forma, ele pode receber transições de no máximo 3 outros vizinhos.

O estado E só se tornará pronto para criar transições (transições onde E é origem) depois que o valor d(E) for o valor mínimo. Temos certeza que o valor é mínimo somente quando todos os outros estados tentaram criar transições a E. Assim, quando o transdutor adaptativo é criado, ele é inicializado com 3 (se não estiver na borda) e a cada tentativa de se criar uma transição, a função p(E) é executada.

A função "pronto", p(E) faz:

p(E) = p(E) -1;

Se p(E) =0 então, dispara o processo "criação de transições" explicado em 2.1 nesse mesmo item.

2) transição

Cada transição vai ser representada por:

(estado origem, estado destino, caractere consumido, saída).

Exemplo:

((0,0), (0,1), s1[1], D); transita do estado (0,0) para o estado (0,1), consome o caractere s1[1] e escreve D.

O caractere consumido varia dependendo da operação. A operação D consome caracteres de s1, a operação I, caracteres de s2 e R e M consomem caracteres de s1 e s2. No caso das operações R e M. No caso da operação R, vamos representar os dois caracteres consumidos como (s1[i], s2[j]) para o estado (i,j).

As transições para (i,j) provém somente de 3 estados outros estados possíveis e podem ser apenas de 4 tipos:

- ((i,j-1), (i,j), s1[j], D)
- ((i-1,j), (i,j), s2[i], I)
- ((i-1,j-1), (i,j), s1[j], M)
- ((i-1,j-1), (i,j), (s1[i],s2[j]), R)

A princípio, o transdutor começa sem nenhuma transição entre estados.

Em um autômato normal, uma marca (um token) passa de um estado para outro consumindo um caractere da cadeia de entrada. Nosso transdutor adaptativo funciona de forma diferente.

2.1) A criação de transições

Quando um estado (i,j) estiver pronto para criar transições (p(E) =0), ele fará:

p(i,j+1); isto é: avisa que houve uma tentativa para se criar uma transição para o estado (i,j+1).

Se (d(i,j+1) <= d(i,j) + 1) então crie a transição: ((i,j), (i,j+1), s1[j+1], D)

p(i+1,j);

Se (d(i+1,j) <= d(i,j) + 1) então crie a transição: ((i,j), (i+1,j), s2[i+1], I)

p(i+1,j+1);

Se d(i+1,j+1) <= d(i,j) + 1) então

    Se (s1[j+1] igual s2[i+1])) então

        crie a transição: ((i,j), (i+1,j+1), s1[j+1], M)

    senão

        crie a transição: ((i,j), (i+1,j+1), (s1[j+1],s2[i+1]), R)

Inicialização do transdutor

O transdutor é inicializado criando-se os estados de (0,0) a (n,m).

Para o estado (0,0), p(0,0) = 0; ou seja, este estado pode criar outras transições.

Para os estados (0,j) para j de 1 a m, p(0,j) = 1, d(0,j) = j
Para os estados (i,0) para i de 1 a n, p(i,0) = 1, d(i,0) = i
Para os demais estados (i,j), p(i,j) = 3, d(i,j) = i+j

Funcionamento do transdutor

O estado (0,0) executa o processo de criação de transições. Ele criará 3 transições: D, I e R ou M.
Quando essas transições são criadas, o estado (0,1) e o estado (1,0) estão prontos para criar outras transições. As transições sempre são criadas de forma a se reduzir o valor d(E).

Um exemplo

Aplicando esse transdutor adaptativo às strings "gato" e "gente" temos algo semelhante à tabela 1 onde cada célula (i,j) da tabela 1 corresponde ao estado (i,j) e d(i,j) corresponde ao valor da célula. Além disso, o autômato adaptativo teria gerado, entre outras, as transições na tabela 2.

TABELA 2
ALGUMAS TRANSIÇÕES GERADAS PELO TRANSDUTOR ADAPTATIVO NO ALINHAMENTO DE "GATO" E "GENTE"

estado origem	**estado destino**	**consome**	**escreve**
(0,0)	(0,1)	g	D
(0,0)	(1,1)	g	M
(1,1)	(2,1)	e	I
(2,1)	(3,2)	(a,n)	R
(3,2)	(4,3)	t	M
(4,3)	(5,4)	(o,e)	R
(2,2)	(3,3)	(t,n)	R

Para se obter as operações que transformam a string s1 em s2 basta observar o caminho que leva (0,0) a (n,m).
Dado que diversos estados podem ficar prontos para disparar transições, então se essas transições ocorressem em paralelo, esse algoritmo poderia rodar mais rápido do que o tradicional da seguinte forma: logo após a inicialização apenas o estado (0,0) está pronto (1 estado pronto) e pode tornar os estados (0,1) e (1,0) prontos para o disparo (2 estados prontos). Esses estados podem tornar os estados (2,0), (1,1) e (0,2) prontos (3 estados prontos); depois é a vez dos estados (3,0), (2,1), (1,2) e (0,3) (4 estados prontos) e assim por diante, ou seja, a tabela 1 é varrida pelas diagonais podendo processar os seguintes números de estados prontos em paralelo: 1, 2, 3, 4, 5, 5, 4, 3, 2 e 1. Se todos eles levassem uma unidade de tempo para processar e disparar então o algoritmo paralelo rodaria em 10 unidades de tempo, pois cada uma das 10 diagonais seria executada em uma unidade de tempo. A fim de generalizar essa idéia, observamos o seguinte: uma diagonal é formada pelas células (x,y) da tabela (n+1)x(m+1) tais que x+y = N. A primeira diagonal com a célula (0,0) corresponde ao número 0. A segunda diagonal às células (1,0) e (0,1) que somam 1 e assim por diante até que a última diagonal corresponde à soma m+n. Para que uma célula (x,y) qualquer no meio da tabela esteja pronta para o disparo ela depende das células (x-1,y) e (x,y-1) que estão na diagonal x+y-1 e também da célula (x-1,y-1) que está na diagonal x+y-2. Se todas as células das duas diagonais anteriores já tiverem executado seu trabalho então todas as células da diagonal x+y pode ser executada. A célula da diagonal 0 não depende de ninguém e as células da diagonal 1 dependem apenas da (0,0). Assim, por indução, temos que todas as outras diagonais podem ser resolvidas com base apenas nas diagonais anteriores, isto é, o trabalho pode ser realizado em paralelo seguindo pelas diagonais.

Comparando com o tradicional que é da ordem de (m+1) × (n+1), temos que aproveitando o máximo de paralelismo possível, o algoritmo executaria em um tempo da ordem de m+n (que corresponde ao número de diagonais que possui uma tabela (m+1)×(n+1)). Já o algoritmo apresentado em [4] é de ordem exponencial e, portanto, de desempenho pior que o tradicional.

V. CONCLUSÃO E CONSIDERAÇÕES FINAIS

Usando o transdutor adaptativo torna-se possível fazer o alinhamento entre duas strings de forma a se recuperar o caminho. O paralelismo desse algoritmo não é muito grande porque as células sendo geradas dependem das vizinhas, no entanto, é possível se trabalhar com autômatos adaptativos paralelos [3] de forma a se agilizar um pouco o algoritmo. O algoritmo aqui proposto permite um grau maior de paralelismo em relação ao tradicional porque os estados disparam assim que estiverem prontos. Como eles dependem dos estados vizinhos, se todos os estados disparassem assim que estivessem prontos então os estados (x,y) correspondentes a cada diagonal (ou seja, x+y=N) seriam disparados a cada vez para N de zero a m+n. Assim, em m+n disparos paralelos, o algoritmo seria resolvido, ao contrário do tradicional que seria resolvido em um tempo da ordem de m*n.

O transdutor adaptativo aqui proposto gera todas as soluções possíveis porque uma transição de E para V é criada desde que d(V) não aumente. Poderíamos pensar numa forma de se ter uma solução única. Nesse caso, uma transição de E para V é criada desde que d(V) seja reduzido. Existe um mecanismo de se retirar o não determinismo para autômatos adaptativos em [5], no entanto, escolher uma transição qualquer que leve ao d(V) mínimo já resolve o problema do alinhamento e por isso [5] não foi considerado nesse artigo.

A referência [4] trata do mesmo assunto: o alinhamento entre duas cadeias usando autômatos adaptativos. Aqui buscamos enfatizar o algoritmo clássico de alinhamento, entretanto sem enfocar o formalismo que pode ser visto em [4]. Porém [4] não utiliza programação dinâmica para o alinhamento, ou seja, não guarda os estados intermediários e por isso faz muito mais processamento que o necessário.

A forma como implementamos p(E) lembra redes de

Petri. Poderíamos pensar que o estado fica pronto para criar transições apenas quando todos os estados que poderiam se ligar a E através de transições estivessem com marcas. Porém o mecanismo é ainda diferente de redes de Petri porque no caso de redes de Petri, as transições e estados são estáticos.

A pesquisa em alinhamento de strings é bem vasta, sendo [1] uma referência básica sobre o assunto. Ela possui uma forte relação com autômatos. A pesquisa sobre o uso autômatos adaptativos em alinhamento de strings pode evoluir para englobar árvores de sufixos e outros mecanismos que tornem o alinhamento ainda mais rápido.

VI. AGRADECIMENTO

Ao professor João José Neto pelo apoio, críticas e sugestões.

REFERÊNCIAS

[1] Gusfield, Dan "Algorithms on Strings, Trees and Sequences"; Cambridge University Press, 1997

[2] Gonzalo Navarro. A guided tour to approximate string matching. ACM Computing Surveys, 33(1):31–88, 2001.

[3] ROCHA, R. L. A. ; GARANHANI, César Eduardo Cavani . Parallel adaptive finite state automata. In: XII Argentine Congress on Computer Science - CACIC 2006, 2006, Potrero de los Funes. Anales del CACIC 2006. Potrero de Los Funes: : Universidad Nacional de San Luis, 2006. v. 1. p. 1-12.

[4] RODRIGUES, E. S. C. ; RODRIGUES, F. A. ; ROCHA, R. L. A. . Automatos Adaptativos para Emparelhamento de Cadeias. In: Segundo Workshop de Tecnologia Adaptativa - WTA'2008, 2008, São Paulo. Memórias do Segundo Workshop de Tecnologia Adaptativa. São Paulo : Escola Politécnica da USP, 2008. v. 1. p. 27-30.

[5] ROCHA, R. L. A. ; JOSÉ NETO, João . An Adaptive Finite-State Automata Application to the problem of Reducing the Number of States in Approximate String Matching. In: XI Congreso Argentino de Ciencias de la Computación - CACIC 2005, 2005, Concordia. Proceedings of the CACIC'2005. Concordia : Universidad Nacional de Entre Ríos - Argentina, 2005. v. 1. p. 1-9.

Dispositivo Adaptativo na Análise de Modelos de Distribuição de Espécies (05 Dezembro 2009)

Elisângela S. C. Rodrigues, Fabrício A. Rodrigues, Ricardo L. A. Rocha

***Resumo* — A modelagem de distribuição de espécies desempenha um importante papel na conservação da biodiversidade. Este artigo apresenta uma proposta de um método baseado em um dispositivo adaptativo para a análise de resultados da modelagem de distribuição de espécies.**

***Palavras chave* — Modelagem de Distribuição de Espécies (*Modeling of Species Distribution*), Dispositivo Adaptativo (*Adaptive Device*).**

I. INTRODUÇÃO

A destruição da biodiversidade no mundo é um problema que vem crescendo rapidamente nas últimas décadas. Este problema causa a diminuição ou até mesmo a extinção de espécies animais e vegetais no planeta. Os efeitos da perda de biodiversidade são irreversíveis. Por isso, a conservação ambiental é um dos grandes desafios atuais da humanidade.

Segundo Myers et al. [1], é muito difícil a preservação de todas as espécies ameaçadas de extinção, pois o número de espécies a serem protegidas ultrapassa os recursos disponíveis para a conservação. Assim, uma das maneiras de conservar a maior quantidade de espécies com o menor custo possível é a identificação de *hotspots*, ou seja, regiões cuja biodiversidade esteja significativamente ameaçada pela destruição.

O Brasil é o país que abriga a flora mais rica do planeta, cerca de um sexto do total [1]. Desta forma, fica evidente a necessidade do estudo e do desenvolvimento de tecnologias que auxiliem na conservação da biodiversidade.

A modelagem de distribuição geográfica de espécies é uma técnica importante que tem sido aplicada em diferentes estudos da biodiversidade. Por exemplo, avaliação do impacto das mudanças climáticas [2], [3], previsão de invasão de espécies [4], [5] e planejamento da conservação de espécies ameaçadas [6].

Para aplicar a técnica de modelagem de distribuição geográfica de espécies, são necessários dados de ocorrências de espécies e variáveis ambientais, tais como variáveis climáticas e topográficas. Existem muitos métodos disponíveis para a tarefa de modelagem. O método mais conhecido e usado é o GARP (*Genetic Algorithm for Rule-set Prediction*) [7]. Recentemente, um método baseado em Entropia Máxima vem sendo usado e testado por diversos pesquisadores [8] – [10].

Os modelos gerados são interpretados empiricamente por profissionais envolvidos na preservação da biodiversidade. Desta forma, é importante o estudo e o desenvolvimento de um método que verifique a qualidade e a precisão dos resultados obtidos, auxiliando na análise dos modelos. O objetivo deste artigo é apresentar uma proposta de um método baseado em um dispositivo adaptativo para a análise dos modelos resultantes da modelagem de distribuição geográfica de espécies.

Os dispositivos adaptativos possuem diversas características que podem ser úteis na análise de modelos de espécies biológicas. A principal característica e vantagem dos dispositivos adaptativos é a capacidade de automodificação. Esta capacidade, além de permitir a representação do conhecimento, permite a incorporação de novas informações e atividades básicas de aprendizagem [11].

Os dispositivos adaptativos possuem aplicações nas mais variadas áreas. Por exemplo, reconhecimento de formas [12], avaliação de autômatos gerados por algoritmos genéticos [13], casamento de padrões [14], entre outros.

Este trabalho está no escopo do projeto *openModeller* e contribuirá com o componente de pós-análise. O projeto *openModeller* tem como principal objetivo fornecer ferramentas livres e de código aberto a pesquisadores interessados em modelagem de distribuição de espécies [15].

Na seção II será apresentada uma visão geral da modelagem de distribuição de espécies. Da mesma forma, na seção III são descritos os dispositivos adaptativos. A proposta do método baseado em um dispositivo adaptativo para a análise de modelos de espécies biológicas é descrita na seção IV. Na seção V são apresentadas as conclusões e as sugestões de trabalhos futuros.

II. MODELAGEM DE DISTRIBUIÇÃO DE ESPÉCIES

Os resultados da modelagem de distribuição geográfica de espécies são distribuições de probabilidade construídas a partir de um conjunto de dados de ocorrência e um conjunto de variáveis ambientais. Os modelos gerados devem representar algumas características ambientais que provavelmente influenciam a qualidade do ambiente para as espécies [16], [17], apud [9].

Os dois primeiros autores são bolsistas da CAPES. O projeto *openModeller* (04/11012-0) é financiado pela FAPESP.

E. S. C. Rodrigues é aluna de Doutorado da Escola Politécnica da Universidade de São Paulo (e-mail: elisangela.rodrigues@poli.usp.br).

F. A. Rodrigues é aluno de Doutorado da Escola Politécnica da Universidade de São Paulo (e-mail: fabricio.rodrigues@poli.usp.br).

R. L. A. Rocha é professor do Departamento de Engenharia de Computação e Sistemas Digitais da Escola Politécnica da Universidade de São Paulo (e-mail: luis.rocha@poli.usp.br).

Os dados de ocorrência são pontos georeferenciados (latitude e longitude) em uma região de interesse onde a presença ou ausência das espécies foi observada. As variáveis ambientais representam o nicho ecológico da espécie, isto é, "um conjunto de condições ecológicas com as quais as populações conseguem se manter" [18] apud [19]. O conjunto de variáveis ambientais pode conter dados de temperatura, topografia, precipitação etc, e devem pertencer à mesma região de estudo [9].

Existem vários métodos de modelagem de distribuição de espécies disponíveis. Os modelos gerados são projetados em um espaço geográfico, embora representem a qualidade do ambiente em um espaço ecológico [9]. A Fig. 1 ilustra de forma abrangente o processo de modelagem.

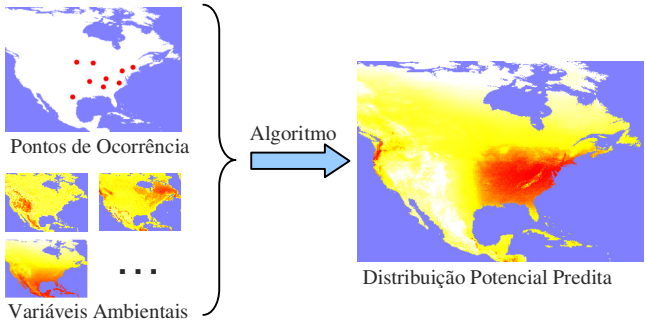


Fig. 1. Processo Modelagem (figura adaptada dos slides referentes a referência [9]).

O modelo criado pelo algoritmo é uma função de probabilidade que mapeia os pontos de ocorrência de uma determinada espécie para um domínio no espaço de variáveis ambientais [19].

Geralmente, existem muitos registros de presença e raros dados de ausência das espécies [9]. Por isso, os métodos que não utilizam dados de ausência são muito empregados. Exemplos de métodos para modelagem de distribuição de espécies que usam somente dados de presença são BIOCLIM [20], [21], e DOMAIN [22].

Na referência [23] são comparados 16 métodos de modelagem de distribuição de espécies em que somente dados de presença são usados na construção dos modelos. Estes métodos foram avaliados como suficientemente precisos para serem usados no planejamento da conservação ambiental e em muitas outras aplicações. Além disso, alguns métodos que ainda não foram aplicados à modelagem de distribuição de espécies se revelaram promissores.

O *openModeller* oferece várias ferramentas aos seus usuários. Dentre elas destacam-se a biblioteca *openModeller* e o *openModeller Desktop*. A biblioteca *openModeller* é capaz de ler dados de ocorrências e de variáveis ambientais, executar algoritmos de modelagem permitindo ao usuário modificar parâmetros referentes a cada algoritmo e escrever de forma apropriada os resultados obtidos. O *openModeller Desktop* é uma ferramenta com uma interface que ajuda o usuário na preparação dos dados, execução dos algoritmos e visualização dos resultados [15]. A Fig. 2 mostra a interface gráfica do *openModeller Desktop*.

Fig. 2. Interface gráfica do *openModeller Desktop*.

III. Dispositivos Adaptativos

Adaptatividade pode ser entendida como a capacidade que um sistema tem de modificar sua própria estrutura, sem a intervenção de agentes externos [24].

Os dispositivos adaptativos são abstrações formais que representam algum sistema cujas operações sejam guiadas por regras. Essas regras definem as próximas situações do sistema e as alterações esperadas no conjunto de regras. Os dispositivos adaptativos são divididos basicamente em duas partes: um dispositivo subjacente, não-adaptativo, e um mecanismo adaptativo [24].

O primeiro dispositivo adaptativo criado foi o Autômato Adaptativo [25]. O dispositivo subjacente do Autômato Adaptativo é o autômato de pilha estruturado. Os autômatos adaptativos são formalismos que possuem equivalência expressiva com as Máquinas de Turing [26]. No entanto, por possuírem as características de um dispositivo adaptativo, tornam-se mais adequados a aplicações cujo comportamento seja dinâmico e que requeiram capacidade de aprendizagem. Além disso, os autômatos adaptativos possuem grande potencial para representar soluções elegantes, eficientes, compactas e práticas para muitos problemas complexos [27].

Posteriormente, outros formalismos adaptativos foram desenvolvidos. A referência [24] apresenta uma revisão bibliográfica ampla de tais formalismos, como gramáticas, *statecharts*, redes de Markov, tabelas de decisão e árvores de decisão.

Os mecanismos adaptativos associados aos dispositivos subjacentes possuem ações adaptativas que permitem alterar o conjunto de regras que definem o sistema. As três ações adaptativas elementares dos mecanismos adaptativos são: Consulta; Remoção e Inserção. A ação de consulta permite a verificação da existência ou não de uma regra com um determinado padrão. A ação de remoção elimina do conjunto de regras todas as regras aderentes a um padrão específico. A ação de inserção insere novas regras no conjunto de regras de acordo com um padrão especificado [28].

IV. PROPOSTA

As subseções seguintes apresentam, respectivamente, uma visão geral do que está implementado no componente de pós-análise do *openModeller* e a proposta de um método baseado em um dispositivo adaptativo para este componente.

A. Pós-análise no openModeller

A validação dos modelos de distribuição de espécies gerados pelo *openModeller* é feita através de algumas medidas consideradas simples. A partir da matriz de confusão – definida abaixo, são calculados: acuidade, erro de omissão e erro de sobreprevisão [29].

A matriz de confusão mostra o número de classificações corretas versus as classificações preditas para cada classe. Esses valores se localizam na diagonal principal da matriz. Os demais elementos da matriz representam erros na classificação, por isso o ideal é que eles sejam iguais a zero. Assim, a matriz de confusão oferece uma medida de precisão do modelo de classificação.

No caso de modelagem de distribuição de espécies, a acuidade é a taxa de acerto de pontos classificados corretamente. Os erros de omissão consistem em não considerar áreas que são habitadas e os erros de sobreprevisão consistem em incluir no modelo áreas que não são habitadas.

Além dessas medidas, o *openModeller* calcula a curva ROC (*Receiver Operating Characteristic*) e a AUC (*Area Under the* ROC *Curve*) [30]. A AUC fornece uma medida de desempenho do modelo [31]. Geralmente, a análise desses valores não é suficiente para a garantia da geração de um bom modelo. Dessa forma, também é necessária a avaliação de um profissional da área.

B. Método de Pós-análise Baseado em um Dispositivo Adaptativo

Segundo Neto [24], os dispositivos adaptativos mais adequados ao auxílio a tomadas de decisões são as tabelas de decisão adaptativas e as árvores de decisão adaptativas.

Como a modelagem de distribuição de espécies é utilizada principalmente para a conservação ambiental e outras tarefas relacionadas a tomadas de decisão, o dispositivo adaptativo usado no método proposto será a tabela de decisão adaptativa. Este dispositivo foi escolhido devido à sua simplicidade de representação e interpretação.

Conforme visto na seção IV-A, além das medidas calculadas pelo *openModeller*, é necessária a análise dos modelos por parte de profissionais especializados. Esta análise é realizada com base em determinadas características dos modelos gerados e muitas vezes fazem parte apenas da experiência pessoal de cada profissional. Em outras palavras, o mesmo modelo pode ser interpretado de forma diferente por profissionais com experiências diferentes.

A mesma situação pode ocorrer na construção dos modelos. Pelo não-determinismo da maioria dos algoritmos de modelagem, diferentes modelos podem ser criados por diferentes algoritmos a partir dos mesmos dados de entrada.

Essa característica dinâmica da modelagem de distribuição de espécies sugere fortemente que os dispositivos adaptativos sejam excelentes candidatos à criação de um método de pós-análise dos modelos gerados.

A primeira parte do método proposto é a identificação das características básicas dos modelos que geralmente são usadas para a identificação da sua qualidade. A partir dessas características, será criado um conjunto de regras que reconheça um bom modelo. Essas regras terão funções adaptativas associadas para inserir outras que permitam o reconhecimento de modelos de boa qualidade, porém com características mais sofisticadas.

Além disso, regras poderão ser simplesmente removidas ou substituídas por outras que caracterizem melhor a qualidade do modelo. Assim, a cada novo modelo gerado, o método aprenderá a reconhecer modelos de melhor qualidade. As características e métricas freqüentemente utilizadas para avaliar a qualidade dos modelos são apresentadas na seção IV-B-1.

Ao término da execução das regras, o usuário deverá receber explicações acerca das inferências realizadas para a obtenção da conclusão final. A Fig. 3 apresenta um esquema do método proposto.

Fig. 3. Esquema do método proposto.

1) Avaliação de modelos

A qualidade de um modelo deve ser avaliada de acordo com o objetivo estabelecido para a modelagem. Por exemplo, suponha que a meta seja prever se uma determinada região possui as condições necessárias para a proliferação de uma praga. Um modelo considerado de boa qualidade para esse problema não é bom, se o objetivo é a conservação da espécie atacada pela praga. Assim, a escolha da métrica de avaliação dos modelos deve levar em consideração o objetivo da modelagem. Isso pode ser feito dando pesos a cada tipo de medida.

Geralmente, as primeiras medidas usadas na avaliação da qualidade de um modelo são as taxas de erro e a precisão. Neste caso, verifica-se qual a porcentagem de pontos de ocorrências preditos corretamente. No entanto, o ideal é que dois conjuntos disjuntos de dados sejam usados, um para criar o modelo e outro para testá-lo.

Quando apenas um conjunto de dados está disponível, a avaliação da qualidade é feita na calibração do modelo, usando as técnicas de validação cruzada, *bootstrap* ou *jackknife*, por exemplo. Quando dois conjuntos de dados independentes estão disponíveis, a avaliação é feita na calibração e pela comparação dos valores preditos com os valores observados [32]. No caso em que existe um conjunto de dados para testar o modelo, a taxa de erro é verificada sobre este conjunto.

A taxa de erro de um classificador h, denotada por

err(*h*) (1), compara os pontos de ocorrência com os pontos preditos pelo algoritmo. O número de exemplos é indicado por n; y_i é a classe associada ao exemplo x_i, em que $1 \leq i \leq n$; e o operador $\| E \|$ retorna 1 se a expressão E for verdadeira, e zero caso contrário [35].

$$err(h) = \frac{1}{n}\sum_{i=1}^{n} \| y_i \neq h(x_i) \| \qquad (1)$$

A precisão do modelo, denotada por *acc*(*h*) é o complemento da taxa de erro (2) [35].

$$acc(h) = 1 - err(h) \qquad (2)$$

Outra forma de avaliar os modelos é pelo tamanho da área predita. Áreas muito pequenas indicam que o modelo pode ter sido super ajustado (*overfitting*) para o conjunto de treinamento e podem ocorrer predições falso-negativas, ou seja, o modelo prevê que a espécie não ocorrerá em uma área que possui as condições para que ela se mantenha. As áreas muito grandes podem resultar na predição de regiões potenciais, mas que não são ocupadas pela espécie [33].

Várias medidas podem ser obtidas a partir das taxas de erro. A curva ROC, por exemplo, é usada para medir a especificidade e a sensibilidade do modelo. A especificidade (3) é a ausência de erros de sobreprevisão e a sensibilidade (4) é a ausência de erros de omissão.

$$especificidade = \frac{TN}{TN + FP} \qquad (3)$$

Em que *TN* é o número de exemplos falsos classificados corretamente como falsos e *FP* é o número de exemplos falsos classificados erroneamente como verdadeiros.

$$sensibilidade = \frac{TP}{TP + FN} \qquad (4)$$

Em que *TP* é o número de exemplos verdadeiros classificados corretamente como verdadeiros e *FN* é o número de exemplos verdadeiros classificados erroneamente como falsos.

A avaliação da qualidade dos modelos envolve muitos fatores, como a adequação das variáveis utilizadas na modelagem, a escala geográfica e a distribuição ambiental e espacial das espécies [34]. Desta forma, o dispositivo adaptativo proposto será capaz de efetuar diversas medidas de acordo com o tipo de dados e com o objetivo da modelagem, se adaptando de acordo com as necessidades do problema.

2) Integração do Mecanismo Adaptativo com o openModeller

O componente de pós-análise da ferramenta *openModeller* é implementado, atualmente, de forma que as medidas calculadas sejam apresentadas ao final da execução do algoritmo de modelagem. Da mesma forma, o resultado obtido pelo dispositivo adaptativo será exibido ao término da construção dos modelos. No entanto, para que a tabela adaptativa seja armazenada e reutilizada novamente, o usuário deverá ser capaz de salvar os resultados e reutilizá-los nas execuções subseqüentes.

Como cada usuário pode gerar modelos com finalidades diferentes, o objetivo da análise deve fazer parte das especificações de entrada. Outras características de entrada poderão ser definidas a partir de um estudo em conjunto com o especialista.

V. Considerações finais

A modelagem de distribuição geográfica de espécie é uma área que tem recebido muita atenção devido à sua importância na conservação da biodiversidade. Como o Brasil é um dos países com a flora mais variada do mundo, é necessário interesse crescente em novas tecnologias que contribuam para a preservação desta riqueza.

Este artigo mostrou uma visão simplificada do processo de modelagem de distribuição de espécies e dos dispositivos adaptativos. Além disso, foi proposto um método baseado em tabelas de decisão adaptativas para a análise da qualidade dos modelos gerados. Esse método está sendo configurado e implementado como uma das contribuições ao projeto *openModeller*.

Como trabalhos futuros, pretende-se implementar o método proposto e utilizar a tecnologia adaptativa em um algoritmo de modelagem baseado em entropia máxima.

Referências

[1] N. Myers, R. A. Mittermeier, C. G. Mittermeier, G. A. B. da Fonseca and J. Kent, "Biodiversity hotspots for conservation priorities", Nature, Vol. 403, pp. 853 – 858, 24 February 2000.

[2] C. D. Thomas, A. Cameron, R. E. Green, M. Bakkenes, L. J. Beaumont, Y. C. Collingham, B. F. N. Erasmus, M. F. de Siqueira, A. Grainger, L. Hannah, L. Hughes, B. Huntley, A. S. van Jaarsveld, G. F. Midgley, L. Miles, M. A. Ortega-Huerta, A. T. Peterson, O. L. Phillips and S. E. Williams, "Extinction risk from climate change", Nature, Vol. 427, pp. 145 – 148, 8 January 2004.

[3] M. F. de Siqueira and A. T. Peterson, "Consequences of Global Climate Change for Geographic Distributions of Cerrado Tree Species", Biota Neotropica, Vol. 3, No. 2, 2003.

[4] A. T. Peterson, M. Papes and D. A. Kluza, "Predicting the potential invasive distributions of four alien plant species in North America", Weed Science, Vol. 51, pp. 863 – 868, 2003.

[5] A. T. Peterson and D. A. Vieglais, "Predicting Species Invasions Using Ecological Niche Modeling: New Approaches from Bioinformatics Attack a Pressing Problem", BioScience, Vol. 51, No. 5, pp. 363 – 372, May 2001.

[6] C. H. Graham, S. Ferrier, F. Huettman, C. Moritz and A. T. Peterson. "New developments in museum-based informatics and applications in biodiversity analysis", TRENDS in Ecology and Evolution, Vol.19, No.9, pp. 497 – 503, September 2004.

[7] D. Stockwell and D. Peters, "The GARP modelling system: problems and solutions to automated spatial prediction", International Journal of Geographical Information Science, Vol. 13, No. 2, pp. 143-158, 1999.

[8] S. J. Phillips, M. Dudík and R. E. Schapire, "A Maximum Entropy Approach to Species Distribution Modeling". Proceedings of the 21st International Conference on Machine Learning, Banff, Canada, 2004.

[9] S. J. Phillips, R. P. Anderson, and R. E. Schapire, "Maximum entropy modeling of species geographic distributions", Ecological Modelling, Vol. 190, pp. 231 – 259, 2006.

[10] S. J. Phillips and M. Dudík, "Modeling of species distributions with Maxent: new extensions and a comprehensive evaluation", Ecography, Vol. 31, pp. 161 – 175, 2008.

[11] J. J. Neto, "Autômatos em Engenharia de Computação - uma Visão Unificada", Primera Semana de Ciencia y Tecnología de la Sociedad Chotana de Ciencias y la Red Mundial de Científicos Peruanos Ciudad de Chota, Perú, Junio 22-27, 2003.

[12] L. C. B. Neto, A. R. Hirakawa e A. M. A. Massola, "Aplicação de Técnicas Adaptativas em Reconhecimento de Formas", Memórias do WTA 2008 – Segundo Workshop de Tecnologia Adaptativa, pp. 1 – 4, 2008.
[13] V. D. Lopes e R. L. A. Rocha, "Aplicação de Autômatos Adaptativos na avaliação de autômatos gerados por algoritmos genéticos", Memórias do WTA 2008 – Segundo Workshop de Tecnologia Adaptativa, pp. 14 – 17, 2008.
[14] E. S. C. Rodrigues, F. A. Rodrigues e R. L. A. Rocha, "Autômatos Adaptativos para Emparelhamento de Cadeias", Memórias do WTA 2008 – Segundo Workshop de Tecnologia Adaptativa, pp. 27 – 30, 2008.
[15] T. Sutton, R. de Giovanni and M. F. de Siqueira, "Introducing openModeller – a fundamental niche modeling framework", OSGEO Journal, Vol.1, 2007. Available in: http://www.osgeo.org/files/journal/final_pdfs/OSGeo_vol1_openModeller.pdf
[16] J. H. Brown and M. V. Lomolino, "Biogeography", 2nd ed., Sinauer Associates, Sunderland, Massachusetts, 1998.
[17] T. Root, "Environmental factors associated with avian distributional boundaries", J. Biogeography 15, pp. 489 – 505, 1988.
[18] G. E. Hutchinson, "Introducción a la Ecologia de Poblaciones", Barcelona, Editorial Blume, 492p., 1981.
[19] M. F. de Siqueira, "Uso de modelagem de nicho fundamental na avaliação do padrão de distribuição geográfica de espécies vegetais", Tese de Doutorado, Departamento de Engenharia Ambiental da Universidade de São Carlos, São Paulo, 2005.
[20] J. R. Busby, "A biogeographical analysis of Nothofagus cunninghamii", (Hook.) Oerst. in southeastern Australia. Aust. J. Ecol. 11, 1 – 7, 1986.
[21] H. Nix, "A biogeographic analysis of Australian elapid snakes", Atlas of Elapid Snakes of Australia, Australian Government Publishing Service, Canberra, Australia, 4 – 15, 1986.
[22] G. Carpenter, A. N. Gillison and J. Winter, "DOMAIN: a flexible modeling procedure for mapping potential distributions of plants, animals", Biodivers. Conserv. 2, 667 – 680, 1993.
[23] J. Elith, C. H. Graham, R. P. Anderson, M. Dudík, S. Ferrier, A. Guisan, R. J. Hijmans, F. Huettmann, J. R. Leathwick, A. Lehmann, J. Li, L. G. Lohmann, B. A. Loiselle, G. Manion, C. Moritz, M. Nakamura, Y. Nakazawa, J. McC. Overton, A. T. Peterson, S. J. Phillips, K. S. Richardson, R. Scachetti-Pereira, R. E. Schapire, J. Soberón, S. Williams, M. S. Wisz and N. E. Zimmermann, "Novel methods improve prediction of species' distributions from occurrence data", -Ecography 29: 129 – 151, 2006.
[24] J. J. Neto, "Um Levantamento da Evolução da Adaptatividade e da Tecnologia Adaptativa", Revista IEEE América Latina, Vol. 5, Num. 7, ISSN: 1548-0992, pp. 496 – 505, Novembro 2007.
[25] J. J. Neto, "Contribuições à Metodologia de Construção de Compiladores", Tese de Livre Docência, Escola Politécnica da USP, São Paulo, 1993.
[26] R. L. A. Rocha e J. J. Neto, "Autômato Adaptativo, limites e complexidade em comparação com Máquina de Turing", In: Proceedings of the Second Congress of Logic Applied to Technology – LAPTEC'2000, São Paulo: Faculdade SENAC de Ciências Exatas e Tecnologia, pp. 33 – 48, 2001.
[27] J. J. Neto, "Solving Problems Efficently with Adaptive Automata", CIAA 2000 – Fifth International Conference on Implementation and Application of Automata, London, Ontário, Canadá, July 2000.
[28] 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 África, Springer Verlag, pp. 234 – 250, July 23 – 25 2002.
[29] Report OM-FAPESP: openModeller - A framework for species modeling. Partial Report N. 2, 2007. Disponível em: http://openmodeller.incubadora.fapesp.br/portal/reports/2007
[30] Report OM-FAPESP: openModeller - A framework for species modeling. Partial Report N. 3, 2008. Disponível em: http://openmodeller.incubadora.fapesp.br/portal/reports/2008
[31] A. H. Fielding, and J. F. Bell, "A review of methods for the assessment of prediction errors in conservation presence/absence models", Environmental Conservation 24: 38 – 49, 1997.
[32] A. Guisan and N. E. Zimmermann, "Predictive habitat distributions models in ecology", Ecological Modelling, Vol. 135, pp. 147 – 186, 2000.
[33] C. J. Raxworthy, C. M. Ingram, N. Rabibisoa and R. G. Pearson, "Applications of Ecological Niche Modeling for Species Delimitation: A Review and Empirical Evaluation Using Day Geckos (Phelsuma) from Madagascar", Systematic Biology, Vol. 56 (6), pp. 907 – 923, 2007.
[34] P. Segurado and M. B. Araújo, "An evaluation of methods for modelling species distributions", Journal of Biogeography, Vol. 31, pp. 1555 – 1568, 2004.
[35] S. O. Rezende – organização, "Sistemas Inteligentes: fundamentos e aplicações", Barueri, SP: Manole, 2003.

Elisângela S. C. Rodrigues nasceu em Pelotas/RS, Brasil, em 18 de abril de 1978. Graduou-se em Informática pela Universidade Federal de Pelotas em 2000 e obteve o título de Mestre em Informática pela Universidade Federal de Campina Grande em 2002.
Iniciou suas atividades docentes em 2002, na Faculdade de Tecnologia e Ciências de Feira de Santana/BA, onde atuou até 2005. Neste mesmo ano, transferiu-se para a Faculdade ÁREA1 em Salvador/BA, onde atuou até 2007. Em 2006, participou como pesquisadora bolsista (com bolsa da Petrobrás) do Projeto Sistema de Gerenciamento de Poços Automatizados (SGPA), na Universidade Federal da Bahia. Atualmente, é aluna regular do Programa de Pós-graduação em Engenharia Elétrica da Escola Politécnica da Universidade de São Paulo, em regime integral.

Fabrício A. Rodrigues nasceu em Mossoró/RN, Brasil, em 09 de agosto de 1973. Graduou-se em Sistemas de Informação pela Universidade Potiguar em 2000 e obteve o título de Mestre em Informática pela Universidade Federal de Campina Grande em 2002.
Iniciou suas atividades docentes em 2002, na Faculdade de Tecnologia e Ciências de Feira de Santana/BA, onde atuou até 2005. Neste mesmo ano, transferiu-se para a Faculdade ÁREA1 em Salvador/BA, onde atuou até 2007, além de atuar como Chefe do Núcleo de Computação. Atualmente, é aluno regular do Programa de Pós-graduação em Engenharia Elétrica da Escola Politécnica da Universidade de São Paulo, em regime integral.

Ricardo Luis A. Rocha é natural do Rio de Janeiro-RJ e nasceu em 29/05/1960. Graduou-se em Engenharia Elétrica modalidade Eletrônica na PUC-RJ, em 1982. É Mestre e Doutor em Engenharia de Computação pela EPUSP (1995 e 2000, respectivamente). Suas áreas de atuação incluem Tecnologias Adaptativas, Fundamentos de Computação e Modelos Computacionais.

Dr. Rocha é membro da ACM (Association for Computing Machinery) e da SBC (Sociedade Brasileira de Computação).

Uma Abordagem de Autômatos Adaptativos Usando Teoria das Categorias

J. L. R. Monteiro e J. E. Kögler Jr., *Senior Member IEEE*

***Resumo* — A teoria das categorias oferece em uma abordagem unificada aspectos de topologia e da álgebra. Seu poder de abstração permite abstrair de maneira prática o tratamento de sistemas e processos de naturezas bastante genéricas. Apresenta-se como potencial benefício ao estudo do formalismo de autômatos. Introduz-se neste trabalho uma proposta para uma abordagem do citado formalismo para autômatos adaptativos.**

***Palavras-chave* — Modelos matemáticos, Teoria das Categorias, Autômatos, Autômatos Adaptativos.**

I. Introdução

A Teoria das Categorias é relativamente recente (apresentada pela primeira vez por S. Mac Lane em 1945, publicada em [1]). Ela facilita o tratamento simultâneo de aspectos topológicos e algébricos. Essa teoria trata de forma conjunta estruturas matemáticas genéricas e relacionamentos entre elas.

Ela fornece uma descrição abstrata de sistemas, estruturas, operações e processos, desta forma se constituindo em um jargão e um ambiente consistente e unificado para o estudo de diversas áreas sob o ponto de vista da matemática. Sua capacidade de generalização, abstração e unificação é o grande mérito de Teoria das Categorias. Suas aplicações para a Ciência da Computação são inúmeras, como por exemplo a Semântica Categorial [2] e a Lógica Categorial[3][4].

Essa teoria permite trabalhar de forma concisa e expressiva diversos conceitos complexos como o formalismo de autômatos adaptativos. Neste trabalho propõe-se um modelo para a o formalismo adaptativo[5] de autômatos, expresso sob essa nova ótica.

Na próxima seção apresentamos os conceitos fundamentais, proporcionando uma visão de em larga escala da teoria das categorias. Na seção 3 ilustra-se a aplicação da teoria para representar, primeiramente, os autômatos finitos, que servirá de inspiração para a próxima construção. Em seguida, na seção 4, revemos a definição formal do autômato adaptativo[6], introduzindo-se uma proposta de modelo via teoria das categorias. Finalmente, na seção 5 faz-se uma breve exploração das conseqüências dessa nova teoria ao paradigma adaptativo.

J.L.R. Monteiro é estudante de doutorado em Engenharia Elétrica da Escola Politécnica da Universidade de São Paulo (e-mail.: jmonteiro@gmail.com).

J.E. Kögler Jr. é pesquisador doutor do Departamento de Engenharia de Sistemas Eletrônicos da Escola Politécnica da Universidade de São Paulo, Av. Prof. Luciano Gualberto, 158, Trav.3 05508-900 São Paulo, SP, Brasil (e-mail.: kogler@lsi.usp.br).

II. Conceitos da Teoria das Categorias

A Teoria das Categorias é, sob certa forma, uma generalização da Teoria dos Conjuntos, embora seu alcance seja muito maior. Conjuntos constituem coleções de objetos que satisfazem a uma dada especificação. As categorias são coleções de objetos e, simulatâneamente, das relações entre eles, denominadas *morfismos*. Um exemplo particular de categoria, é a denominada categoria **Set**, cujos objetos são conjuntos e os morfismos são funções entre cada par de conjuntos. Pode-se criar uma categoria **Aut** tendo autômatos finitos como objetos, dotadas de morfismos que são mapeamentos entre cada dois autômatos.

A seguir serão apresentadas algumas propriedades das categorias.

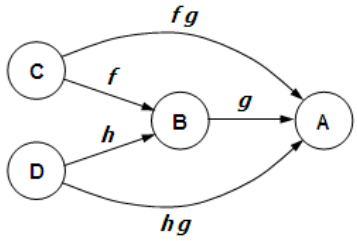


Fig. 1: Composição associativa: nessa figura a composição dos morfismos

f com *g* resulta em *fg*.

A. Propriedades Fundamentais

Uma categoria C é definida por:

- Conjunto Obj(C) dos objetos da categoria;
- Conjunto Mor(C) dos morfismos da categoria, que interrelacionam os elementos de Obj(C);
- Uma operação de composição associativa em Mor(C) x Mor(C), que resulta no morfismo composto entre dois objetos adjacentes a um terceiro (ilustrado na Fig. 1);
- Uma operação de identidade, garantindo que exista um morfismo de cada objeto consigo mesmo.

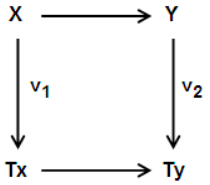


Fig. 2: Diagrama comutativo: nesse diagrama são expressos morfismos comutativos entre *X* e *Tx* e *Y* e *Ty*.

Uma das formas de se exprimir as propriedades em Teoria das Categorias é através de diagramas. Se a composição de todos os caminhos entre dois objetos do diagrama for equivalente, o diagrama é dito *comutativo* (como na Fig. 2). Os diagramas são recursos freqüentemente utilizados na demonstração de teoremas envolvendo categorias.

Uma outra propriedade importante da teoria é a dualidade, através da qual pode-se obter uma categoria inversa ou oposta, bastando inverter o sentido dos seus morfismos. Demonstra-se que as propriedades válidas em uma categoria C são igualmente válidas para a sua categoria dual, representada por C^{op} (exemplificado na Fig. 3).

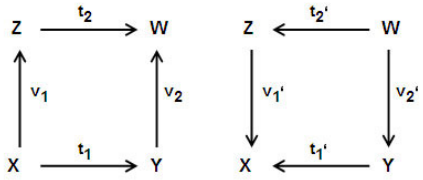


Fig. 3: Categorias duais: a categoria representada do lado direito é a dual à categoria do lado esquerdo.

B. Funtores

Funtores são mapeamentos entre categorias, que preservem a estrutura da categoria domínio na categoria co-domínio, através do mapa. Um funtor *F* (dito *covariante*), da categoria C para a categoria D apresenta as características construtivas:

1. associa para cada objeto X de Obj(C) um objeto F(X) em Obj(D);
2. associa para cada morfismo f : X → Y de Mor(C), um morfismo F(f) : F(X) → F(Y) em Mor(D)

tal que, sendo g de Mor(C), as seguintes propriedades valem:

- $F(id_X) = id_{F(X)}$
- $F(g \circ f) = F(g) \circ F(f)$ para todos os morfismos $f:X \to Y$ e $g:Y \to Z$.

com **o** indicando a composição entre funtores.

C. Tipos especiais de Funtores

Existem diversos tipos notáveis de funtores, destacando-se em especial os seguintes, que utilizaremos em nossas construções posteriores:

Funtor Esquecimento: é um funtor que transforma uma categoria em outra, "esquecendo" parte da estrutura (Fig.4). Por exemplo, o funtor *F: Vect* → *Set*, leva a categoria de todos os espaços vetoriais na categoria dos conjuntos esquecendo a estrutura de espaço vetorial.;

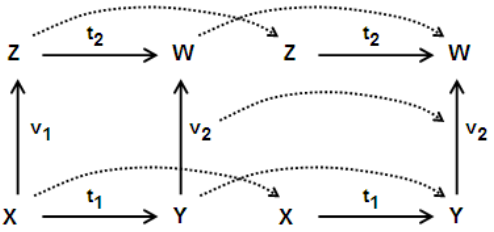


Fig. 4: Funtor Esquecimento: Transforma uma categoria, removendo-lhe parte da estrutura.

Funtor Livre: é o funtor dual do funtor esquecimento, levando cada objeto ao grupo livremente gerado pelo mesmo objeto, permitindo incluir novas estruturas de relacionamento (morfismos) entre os objetos (Fig. 5).

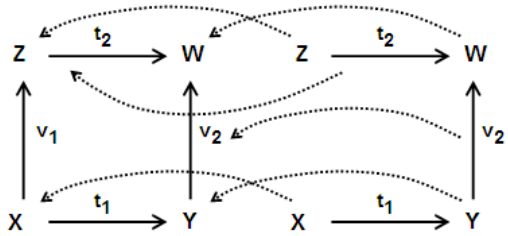


Fig. 5: Funtor Livre: transforma uma categoria, adicionando nova estrutura.

III. Modelagem do Autômato Finito

Primeiramente apresentaremos um modelo dos autômatos finitos usando a teoria das categorias, o que auxiliará na compreensão de um modelo posterior para autômatos adaptativos.

Formalmente, os autômatos finitos podem ser especificados pelas 6-uplas:

A = (I, O, S, s0, t, p) , na qual:

- *I* é o conjunto de símbolos de entrada;
- *O* é o conjunto de símbolos de saída;
- *S* representa o conjunto de estados;
- *s0* é o estado inicial;
- *t: I x S → S* representa a função de transições;
- *p: I x S → O* representa a função de saída.

Sejam *A* e *A'* dois automatos finitos. Um morfismo *f* entre *A* e *A'* escreve-se como:

f = (fI,fO,fS) : A → A', tal que:

fS(s0) = s0' (preservação do estado inicial)

fS(t(i,s)) = t'(fI(i),fS(s)) (preservação das transições)

fO(p(i,s)) = p'(fI(i),fS(s)) (preservação das saídas)

A categoria ***Aut*** dos autômatos finitos pode ser escrita como [7]:

- *Aut=(Obj(Aut), Mor(Aut), Comp(Aut), Ident(Aut))*
- *Obj(Aut): A = (I, O, S, s0, t, p)*
- *Mor(Aut): f = (fI,fO,fS) : A → A'*
- *Comp(Aut)*: a composição é associativa, pois as componentes do morfismo operam nos conjuntos *I, O, S;*
- *Ident(Aut)*: a identidade se dá pelo morfismo que preserva a identidade nos conjuntos *I, O, S.*

Como a categoria ***Aut*** é definida dentro de conjuntos, é relativamente simples observar que a composição de morfismos entre autômatos finitos é associativa, e que a identidade é preservada [7].

IV. Formalizando o Autômato Adaptativo

Com base no entendimento da aplicação da teoria para o autômato finito, explorado na seção anterior, pode-se similarmente formalizar os autômatos adaptativos através da teoria das categorias.

Os autômatos finitos adaptativos são especificados pela 8-upla abaixo [5]:

ADk = (I, O, Sk, sk, tk, p, BA, AA) , na qual:

- *Sk* é o conjunto dos estados, na iteração *k*;
- *s0* é o estado inicial;
- *tk: AA x I x S x BA → S* é a função de transições adaptativa;
- *p: I x S → O* representa a função de saída;
- *BA*: conjunto das funções adaptativas que ocorrem antes;
- *AA*: conjunto das funções adaptativas que ocorrem depois.

Note que as várias iterações dos autômatos adaptativos pressupõem uma indexação através de índices k. Isso se faz necessário devido ao próprio conceito de adaptação, que implica em mudanças estruturais acontecendo de uma situação (iteração) para a seguinte.

Introduziremos agora a categoria ***AAut*** dos autômatos adaptativos. Ela se apresenta como uma coleção de autômatos adaptativos e de morfismos entre eles. Note que ela deve ser uma supercategoria sobre a categoria de ***Aut***. Isto é, cada autômato adaptativo componente de ***AAut*** deve ser visto individualmente como uma categoria menor, tipo ***Aut***, em que seus elementos constituem as diversas iterações compatíveis com a adaptação de um dado autômato aplicando-se suas funções adaptativas convenientemente. Então, os morfismos entre os objetos de ***AAut*** são funtores entre autômatos de ***Aut***:

- *AAut=(Obj(AAut),Mor(AAut),Comp(AAut), Ident(AAut))*
- *Obj(AAut):AA = {ADk, | ADk = (I, O, Sk, sk, tk, p, BA, AA)* }
- *Mor(AAut):F : AA → AA'*
- *Comp(AAut)*: a composição associativa de funtores;
- *Ident(AAut)*: funtor identidade.

Os funtores acima propostos devem levar autômatos adaptativos em autômatos adaptativos, mapeando adequadamente os elementos homólogos. Isso requer que funções adaptativas sejam mapeadas em funções adaptativas. Entre as dificuldades técnicas que se tem de resolver nesse cenário, encontra-se o caso em que duas classes de topologias distintas devem ser relacionadas. Por exemplo, a remoção de estados e transições pode ser considerada através da aplicação de funtores tipo esquecimento. Portanto embora internamente cada categoria ***Aut*** deva satisfazer às condições a ela anteriormente impostas, a categoria ***AAut*** não precisa satisfazê-las, podendo ser bem mais geral. Da mesma forma, pode-se introduzir novos estados e transições, o que se consegue através de funtores do tipo livre, adicionando estados e transições entre objetos de ***AAut***.

Um dos benefícios desse tipo de abordagem é a investigação de otimização estrutural, conseguida através de construtos comuns em teoria das categorias que conduzem a generalizações, como produto e limites e seus duais, os coprodutos e os colimites [1]. Por exemplo, poder-se-ia investigar qual a realização mínima de um dado autômato adaptativo que satisfaz a uma determinada restrição.

V. Conclusões

De uma forma geral, o formalismo de categorias permite formalizar adequadamente os autômatos adaptativos, porém ainda é preciso demonstrar formalmente que a especificação proposta satisfaz às condições construtivas de uma categoria, o que se constitui em um trabalho em andamento. Todavia, por ser uma supercategoria, ***AAut*** herda naturalmente as propriedades construtivas das categorias, o que por si só constiui uma pré-prova. Talvez o aspecto mais difícil se encontre na determinação de funtores para aplicações a situações de particular interesse e no estabelecimento de teoremas que caracterizem os princípios de otimização, por exemplo, aplicáveis à determinação das realizações mínimas.

A abordagem de autômatos adaptativos via teoria das categorias possibilita também sua potencial adoção na realização de estruturas evolutivas que caracterizam processos cognitivos [8]. Realizações como o MES (Memory Evolutive System) [9][10][11], trazem um poder de representação de processos cognitivos fundamentado na teoria de categorias. Nesse cenário, um aspecto interessante de se explorar constitui-se no estudo de como a indexação utilizada na estruturação de cada objeto da supercategoria ***AAut*** pode ser relacionada com a indexação de um outro objeto, criando-se um modelo de temporização local interna, característico de sistemas pulsados assíncronos [9].

REFERÊNCIAS

[1] Mac Lane, S. Categories for the Working Mathematician. Springer. ISBN 0-38-798403-8, 1972.

[2] Pitts, A.M. Categorical Logic. Chapter 2 of S. Abramsky and D. M. Gabbay and T. S. E. Maibaum (Eds) Handbook of Logic in Computer Science, Volume 5. Algebraic and Logical Structures, Oxford University Press, 2000.

[3] Lambek, J. & Scott, P.J. Introduction to Higher Order Categorical Logic, Cambridge University Press, Cambridge, UK, 1986.

[4] Barr, M. and Wells, C. Category Theory for Computing Science, Hemel Hempstead, UK, 1990.

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

[6] Neto, J. J. & Pariente, C. A. B. Adaptive Automata - a Revisited Proposal. Lecture Notes in Computer Science. J.M. Champarnaud, D. Maurel (Eds.): Implementation and Application of Automata 7th International Conference, CIAA 2002, 2002, Vol.2608, Tours, France, July 3-5, Springer-Verlag, 2002, pp. 158-168

[7] Martini, A.; Ehrig, H. & Nunes, D. Elements of basic category theory. Technical Report 96-5, Technical University Berlin, Março 1996.

[8] Kogler Jr.,J.E. & Inojosa,R. – First Steps Towards a Cognitive Architecture Based on Adaptive Automata. Proceedings of the Workshop on Modelling Adaptive and Cognitive Systems – ADAPCOG – Salvador, Brazil, October 30, 2008

[9] Ehresmann, A.; Vanbremeersch, J.-P. - *Memory Evolutive Systems – Hierarchy, Emergence, Cognition*, Studies in Multidisciplinarity V4, Elsevier, Amsterdam, 2007

[10] Monteiro, J.L.R., Ribeiro, J.H.R., Kogler Jr., J.E., Netto, M.L. – On the Requirements for Simulating a Memory Evolutive System – Proceeedings of Brain Inspired Cognitive Systems, BICS, São Luiz, Brazil,June 24-27, 2008

[11] Monteiro, J.L.R., Kogler Jr., J.E., Ribeiro, J.H.R., Netto, M.L. – On Building a Memory Evolutive System for Application to Learning and Cognition Modelling – submitted - Brain Inspired Cognitive Systems – Ed. A. Hussein and V Ctsuridis – Springer – 2009 – To appear.

Julio Lima do Rego Monteiro . Bacharel em Ciências Moleculares pela Universidade de São Paulo (1999). Mestre em Engenharia Elétrica – Sistemas Digitais, pela Universidade de São Paulo (2004) com dissertação sobre inteligência artificial, sistemas multiagentes e simulação social. Atualmente está cursando o doutorado em Engenharia Elétrica, também na Universidade de São Paulo, estudando a simulação de propriedades e processos cognitivos para resolver problemas relacionados com a modelagem computacional da memória, fundamentado na teoria das categorias e modelando o sistema nervoso como um sistema complexo e emergente. Pesquisador do grupo Cognitio, o Núcleo de Apoia à Pesquisa em ciências cognitivas da Universidade de São Paulo. Participou recentemente de estágio de pesquisa no INRIA – Futurs, França.

João Eduardo Kögler Junior . Engenheiro Eletricista pelo Instituto Maua de Tecnologia e bacharel em Física pela Universidade de São Paulo , mestre em Engenharia Elétrica pela Universidade de São Paulo, com ênfase em engenharia biomédica e doutor em Engenharia Elétrica pela Universidade de São Paulo. Foi pesquisador visitante do Siemens Corporate Research, em Princeton, Estados Unidos, durante 1991-92 e do INRIA - Sophia Antipolis, na França, em 1994. Atualmente dedica-se à pesquisa e ensino na Escola Politécnica da Universidade de São Paulo, atuando nas áreas de Visão Computacional, Processamento de Imagens e Inteligência Computacional. É membro senior do IEEE – Computational Intelligence Society, da Sociedade Brasileira de Computação e da Sociedade Brasileira de Física. Além de ter atuado em pesquisa e ensino, também foi consultor e empresário na iniciativa privada, na área de Visão de Máquina e Automação Inteligente, como alliance member da National Instruments. Foi engenheiro de projetos e estudos de sistemas de geração e transmissão de energia elétrica da Companhia Energética de São Paulo e da Themag Engenharia.

ARTIGOS

AdapLib: Uma Biblioteca para Execução de Dispositivos Adaptativos

F. L. Siqueira

***Resumo*— Neste artigo apresenta-se o embasamento teórico e a arquitetura (seguindo o modelo 4+1) da biblioteca para execução de dispositivos adaptativos, *AdapLib*. Essa biblioteca foi criada buscando tratar com fidelidade a teoria de dispositivos adaptativos e, ao mesmo tempo, permitir que ela seja estendida com facilidade para absorver os detalhes específicos de soluções que empreguem esse tipo de dispositivo. Como exemplo disso será apresentado um estudo de caso em que a biblioteca foi usada na criação de uma prova de conceito para o monitoramento e o diagnóstico de problemas em um portal de notícias *online*.**

***Palavras chaves*—Arquitetura (*architecture*), ferramenta de software (*software tool*), Sistemas adaptativos (*adaptive systems*).**

I. Introdução

Os dispositivos adaptativos são uma proposta de formulação teórica para formalismos baseados em regras auto-modificáveis. Seguindo essa teoria, existem na literatura propostas de dispositivos como autômatos [1], árvores de decisão [2], tabelas de decisão [3], state-charts [4], entre outros. Por mais que as formulações definam e provem o funcionamento desses dispositivos, ainda é necessário implementá-los – de alguma forma – para permitir a sua execução (normalmente, usando uma linguagem de programação).

Entretanto os usuários dessas teorias encontram pouco ou nenhum suporte ferramental para aplicá-las na prática. Por mais que esses usuários decidam fazer uma implementação, muitas vezes ela é específica para um problema, indisponível para a comunidade, ou até mesmo apresenta problemas (não é eficiente, possui defeitos, é limitada conceitualmente, etc), o que inviabiliza uma reutilização e pode gerar novamente a necessidade de criar uma nova implementação para o formalismo em questão.

Considerando esse contexto, neste trabalho é apresentada a biblioteca AdapLib, discutindo principalmente sua arquitetura. Essa biblioteca implementa a teoria de dispositivos adaptativos usando os conceitos de orientação a objetos, buscando ao mesmo tempo fidelidade à teoria, eficiência e extensibilidade. A intenção foi criar uma biblioteca de código aberto que possa ser usada em aplicações (comerciais ou não comerciais) sem a preocupação de como os conceitos são implementados; esses detalhes são tratados pela AdapLib.

F. L. Siqueira é doutorando em Engenharia de Computação na Escola Politécnica da Universidade de São Paulo (POLI-USP), Av. Prof. Luciano Gualberto, Trav.3, n.158, 05508-900 (e-mail: levy.siqueira@usp.br).

Para apresentar essa biblioteca, este artigo está organizado da seguinte forma: na Seção II é apresentada a fundamentação teórica a respeito de dispositivos adaptativos. Em seguida, na Seção III é apresentada a biblioteca AdapLib, discutindo sua aderência à teoria, seus objetivos, os trabalhos relacionados e a sua arquitetura. Na Seção IV é apresentado um estudo de caso em que a biblioteca foi usada e os resultados obtidos. Por fim, na Seção V é apresentada a conclusão e as perspectivas de trabalhos futuros.

II. Dispositivos Adaptativos

A teoria de dispositivo adaptativo é uma formulação geral de formalismos auto-modificáveis baseados em regras que busca uma maior proximidade com o formalismo não adaptativo original [3]. Dessa forma, há uma clara separação entre a parte não adaptativa, chamada de camada subjacente, e a parte que permite alterar o comportamento do dispositivo de forma dinâmica, chamada de camada adaptativa. Essa formulação pode ser vista como uma generalização da proposta de autômatos adaptativos feita em [1] em decorrência da aplicação de seus conceitos em outras representações [2].

A camada subjacente representa um dispositivo baseado em um conjunto finito de regras. A partir de uma configuração inicial, essas regras são aplicadas em resposta a eventos de entrada, fazendo com que o dispositivo mude a sua configuração. Ao final do processamento dos eventos, o dispositivo atinge uma determinada configuração que pode aceitar ou rejeitar a entrada em questão. Segundo Neto [3] esse dispositivo pode ser formalmente definido como D=(C, R, S, c0, A, O), onde:

- C é o conjunto de possíveis configurações, com c_0 sendo a configuração inicial do dispositivo;
- S é o conjunto de eventos válidos para o dispositivo, considerando $\varepsilon \in S$;
- $A \subseteq C$ é o conjunto de configurações que aceitam a entrada; e
- O é o conjunto de símbolos que podem ser saída após a execução de uma determinada regra.
- $R \subseteq C \times S \times C \times O$ é o conjunto finito de regras definidas para o dispositivo. Uma regra $r \in R$ é representada por $r=(c_i, s, c_f, o)$, significando que a partir de uma configuração atual $c_i \in C$, um evento $s \in S$ estimula a passagem para uma configuração $c_f \in C$ e gera o símbolo $o \in O$ na saída.

É importante ressaltar que essa formulação permite que haja mais de uma regra aplicável ao considerar uma determinada configuração atual e um evento de entrada.

Caso o dispositivo possua apenas uma regra possível para cada par configuração atual e evento, ele é dito determinístico; caso contrário, ele é dito não-determinístico. Neste último caso, o dispositivo deve, teoricamente, aplicar cada regra possível em paralelo, obtendo como resposta diversas configurações possíveis. Se a subseqüente aplicação de regras em cada uma dessas possíveis configurações (que podem causar ainda mais não-determinismos) levar a uma configuração de aceitação, o dispositivo então aceita a cadeia, caso contrário a rejeita.

Sobre esse tipo de dispositivo pode ser colocada uma camada adaptativa que permite alterar o conjunto de regras e de configurações. Essa alteração é feita através de ações adaptativas que ficam associadas às regras da camada subjacente e são executadas antes da aplicação da regra (ações anteriores) ou após (ações posteriores). Essas ações adaptativas podem ser de três tipos: de inserção (que inserem regras), de remoção (que removem regras) e de busca (que apenas obtêm regras e dados3).

Ao aplicar ações adaptativas que alteram a camada subjacente se tem, conceitualmente, um novo dispositivo adaptativo. Com isso, a aplicação das ações faz com que se transite por um espaço de dispositivos válidos a partir de um dispositivo inicial. Dessa maneira, um dispositivo adaptativo pode ser definido como ADk=(Dk, AM), considerando k execuções de regras com ações adaptativas (ou passos). No caso, Dk é o dispositivo não adaptativo (camada subjacente) e AM é o mecanismo adaptativo que define as ações adaptativas a serem aplicadas a partir de cada regra da camada subjacente. De uma forma mais detalhada, segundo [3] o dispositivo adaptativo pode também ser formalmente definido como ADk=(Ck, ARk, S, ck, Ak, O, BA, AA) em que:

- C_k é o conjunto de possíveis configurações após a execução de k passos, com c_k sendo a configuração inicial do dispositivo nesse passo;
- $AR_k \subseteq BA \times C_k \times S \times C_k \times O \times AA$ é o conjunto de regras adaptativas;
- S é o conjunto de eventos válidos para o dispositivo, considerando $\varepsilon \in S$;
- $A_k \subseteq C_k$ é o conjunto de configurações que aceitam a entrada no passo k;
- O é o conjunto de símbolos que podem ser saída após a execução de uma determinada regra; e
- BA e AA são conjuntos de ações adaptativas anteriores e posteriores, respectivamente (com isso, o mecanismo adaptativo $AM \subseteq BA \times R_k \times AA$, considerando as regras R_k da camada subjacente).

Em geral, as ações adaptativas anteriores e posteriores são organizadas em funções adaptativas que além de agrupar as ações também recebem parâmetros, o que permite um maior reuso das ações adaptativas.

[3] As ações adaptativas de busca não alteram a estrutura da camada subjacente, mas obtêm regras ou dados que são usados pelas ações de inserção e de remoção.

III. ADAPLIB

A teoria de dispositivos adaptativos define formalmente os elementos que constituem esse tipo de dispositivo e apresenta a forma de executá-los (através de um pseudo-algoritmo [3]). A biblioteca AdapLib, implementada em Java, busca seguir com fidelidade essa teoria ao prover uma implementação aderente ao formalismo original que possa ser usada em aplicações. Dessa maneira, a biblioteca permite representar dispositivos não adaptativos e acoplá-los como camada subjacente de um dispositivo adaptativo. Seguindo a definição, os dispositivos representados na biblioteca definem configurações, regras, eventos e símbolos, enquanto que o mecanismo adaptativo trata das regras, ações e funções adaptativas.

Além de buscar uma proximidade à teoria, um outro objetivo da AdapLib é ser extensível. Desenvolvedores podem implementar outros dispositivos não adaptativos e usar a infra-estrutura da biblioteca para usá-los como camada subjacente de um dispositivo adaptativo e executá-los. Mais que isso, usuários da biblioteca podem estender o formalismo original de um dispositivo para adequar a teoria às suas necessidades práticas. Pode-se, por exemplo, fazer com que ao aplicar uma regra seja executado um código específico (em linguagem de programação), executar um código ao passar para uma determinada configuração, ou até mesmo definir novos elementos para um dispositivo.

A AdapLib, portanto, tem como objetivo prover um ambiente de execução de dispositivos adaptativos que seja extensível e eficiente e que possa ser usado em aplicações.

A. *Funcionalidades e limitações*

Considerando os objetivos da biblioteca, a seguir são apresentadas as suas principais funcionalidades em sua versão atual (2.0):

- Separação da camada subjacente da camada adaptativa.
- Uma camada adaptativa pode ser usada como camada subjacente para uma outra camada adaptativa (levando em conta a consideração discutida posteriormente nesta seção).
- Possibilidade de definir comportamentos específicos ao chegar a uma determinada configuração ou ao aplicar uma regra.
- Criação de ações adaptativas de remoção que podem considerar qualquer combinação de configuração inicial, configuração final e evento (menos a que remove todas as regras do dispositivo).
- Criação de ações adaptativas de inserção, definindo até chamadas de função adaptativas anteriores e posteriores na regra a ser inserida.
- Possibilidade de definir funções adaptativas:
  - Para cada função pode-se definir ações anteriores e posteriores à execução das ações definidas nas funções.

- Definição de geradores, ou seja, configurações que são criadas ao executar a função (sem limites para o número de geradores).
- Possibilidade de definir parâmetros para a função (sem limite no número de parâmetros) e usá-los nas ações adaptativas definidas no contexto da função.
- As ações adaptativas de inserção e de remoção não têm uma ordem de execução definida.

- Representação e execução de autômatos finitos e autômatos finitos adaptativos determinísticos.
- Permitir a definição ou alteração do comportamento das regras, configurações e eventos, ou até mesmo da execução do dispositivo subjacente ou do dispositivo adaptativo.

Existência de log para facilitar o entendimento da execução do dispositivo.

Ao definir essas funcionalidades pretendeu-se cobrir os principais conceitos da teoria de dispositivos adaptativos, permitindo que um usuário da biblioteca possa representar um dispositivo teórico com fidelidade ao usá-la. Apesar disso, ainda existem algumas considerações e limitações na versão atual da biblioteca. Talvez a principal delas seja a respeito do não determinismo: por enquanto a biblioteca não o permite. Por mais que a existência do não determinismo seja útil em diversas aplicações, existem algumas dificuldades de implementá-lo considerando a existência de uma camada adaptativa. Entre as dificuldades pode-se considerar a necessidade de criação de uma "onda de clones" [5] (ou seja, cópias da estrutura do dispositivo já que ao aplicar as regras adaptativas se tem diferentes dispositivos), a possível ineficiência em relação a tempo e/ou a memória de implementações de algoritmos de backtracking ou paralelismo.

Por mais que a biblioteca atualmente não permita o não determinismo, ainda é possível criar regras que não consomem símbolos de entrada. Entretanto, considerou-se que elas têm baixa prioridade, ou seja, regras que consomem símbolos devem ser aplicadas preferencialmente, caso haja as duas opções. Caso a camada subjacente em questão apresente mais de uma regra possível, a classe responsável pela execução (Executor) usa a primeira regra passada (de uma lista de regras possíveis).

Uma outra limitação é a respeito do uso de ações adaptativas de busca, que atualmente não são definidas. Por causa disso também não é possível definir variáveis (além dos geradores) dentro de uma função adaptativa. O problema, nesse caso, é como tratar a questão da ineficiência inerente de se fazer uma busca por regras e usar os resultados em outras ações adaptativas. Por não ser um problema tão complexo ou grave, essa funcionalidade será possivelmente implementada nas próximas versões.

Seguindo a formulação dos dispositivos adaptativos, uma outra consideração é em relação à possibilidade de existir adaptabilidade em diversos níveis (ou seja, uma camada adaptativa ser considerada como uma camada subjacente para uma outra camada adaptativa). Existem algumas possíveis interpretações de quais são as regras e as configurações da camada adaptativa subjacente A versão atual da biblioteca adota, por simplicidade, que as regras são as regras adaptativas e as configurações são as mesmas da camada subjacente não adaptativa. Mas uma outra possível resposta é que as ações adaptativas são as regras – e nesse caso é necessária uma análise mais cuidadosa para definir o que seriam as configurações. Dessa forma, ainda é necessária uma melhor análise dessa questão do ponto de vista teórico para considerar que a implementação existente atualmente na biblioteca é suficientemente adequada.

Por fim, atualmente está disponível apenas a implementação de autômatos de estados finitos como camada subjacente. Entretanto, considerando a arquitetura adotada – buscando seguir com fidelidade o formalismo original, imagina-se que a implementação de outros tipos de dispositivos como camada subjacente seja facilmente realizada (na Seção III.B.1. é descrito como um desenvolvedor pode fazê-lo).

B. Arquitetura

Para apresentar de uma forma mais técnica a biblioteca AdapLib, será descrita a arquitetura de software idealizada, ou seja, "a organização fundamental de um software, envolvendo seus componentes, a relação entre eles e o ambiente e os princípios que guiam o seu projeto e sua evolução" [6]. No caso será usado o modelo 4+1 para a definição dos pontos de vista relevantes para a arquitetura [7]. Esse modelo propõe 5 pontos de vista:

- O ponto de vista lógico trata principalmente dos requisitos funcionais do software, descrevendo suas abstrações principais e seus relacionamentos. Em um projeto Orientado a Objetos, um dos principais diagramas dessa visão é aquele que representa suas classes.
- O ponto de vista de implementação trata da organização do código fonte, componentes, arquivos de dados e outros artefatos. Dessa forma, esse ponto de vista trata do ambiente de desenvolvimento e de aspectos da gerência de configuração do software.
- O ponto de vista de processo trata das questões de concorrência do software, cobrindo os requisitos não funcionais de desempenho, escalabilidade e taxa de transferência.
- O ponto de vista de implantação trata do mapeamento dos elementos de software nos elementos de hardware. Com isso, são tratadas as questões de instalação, implantação e também de desempenho do software.
- Por fim, o ponto de vista de caso de uso junta os elementos dos outros pontos de vista, representando os cenários de uso do software e abstraindo seus principais requisitos.

Seguindo essa divisão, a seguir serão apresentadas cada um dos pontos de vista considerando as funcionalidades descritas anteriormente4.

[4] Algumas classes da biblioteca são parametrizáveis. Entretanto por simplicidade essas classes serão apresentadas sem seus parâmetros.

1) Ponto de vista lógico

A atual implementação da biblioteca possui 37 classes e interfaces que serão apresentadas nesta seção de uma forma simplificada, focando em alguns aspectos principais: a descrição do dispositivo (Fig. 1), as funções adaptativas (Fig. 2) e seus parâmetros (Fig. 3), a execução (Fig. 4) e o autômato como camada subjacente (Fig. 5).

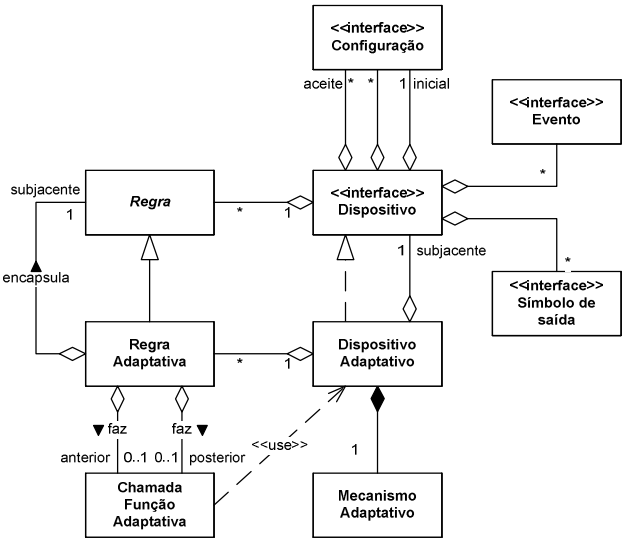


Fig. 1. As classes que tratam do dispositivo e do dispositivo adaptativo.

Na Fig. 1 são apresentados os principais elementos que descrevem um dispositivo e um dispositivo adaptativo. Seguindo a definição apresentada na Seção II, um *Dispositivo* possui *Eventos*, *Configurações*, *Símbolos de saída* e *Regras*. Um *Dispositivo Adaptativo* é também um *Dispositivo*, sendo que as *Regras* são especializadas considerando que elas podem fazer *Chamadas a funções adaptativas*. Além disso, o *Dispositivo Adaptativo* tem um *Mecanismo Adaptativo* que é responsável por tratar das mudanças da estrutura da camada subjacente (usadas pelas ações adaptativas).

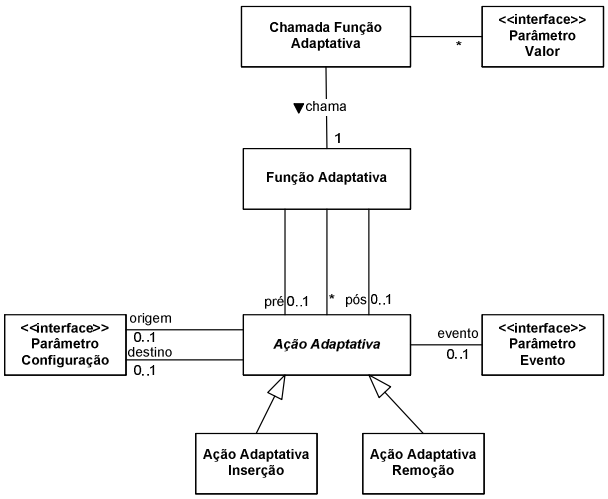


Fig. 2. As classes que tratam das funções adaptativas.

Especificamente sobre a função adaptativa, na Fig. 2 são apresentadas as classes que a representa. As *Chamadas de Função Adaptativas* (que são parte de uma regra adaptativa) recebem parâmetros passados por valor (*Parâmetro Valor*). Com esses valores é então chamada uma *Função Adaptativa* que executa uma *Ação Adaptativa* antes da execução das demais ações (pré) e uma ação que é executada após as demais ações (pós). Essas ações podem ser *Ação Adaptativa Inserção* e *Ação Adaptativa Remoção*. Uma *Ação Adaptativa* define parâmetros que estabelecem padrões para a configuração de origem e de destino e também para o evento (ações de inserção precisam definir todas essas informações, seja com valores ou com referência a valores que serão resolvidas durante a chamada da função, enquanto que as ações de remoção devem definir pelo menos uma dessas informações).

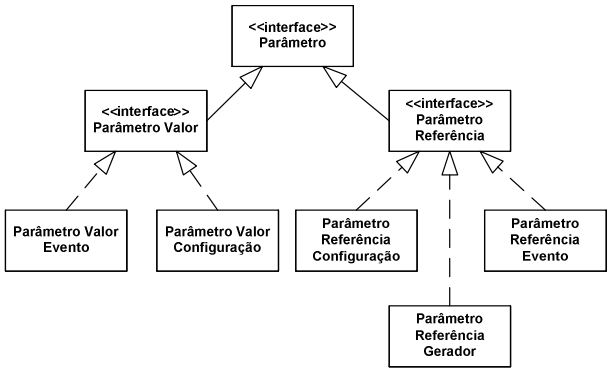


Fig. 3. A hierarquia de classes dos parâmetros.

Uma questão importante desse modelo é em relação aos parâmetros: existe uma hierarquia de tipos de parâmetros que busca evitar que o usuário da biblioteca defina de forma equivocada as funções adaptativas e suas chamadas. Na Fig. 3 é apresentado um diagrama de classes a esse respeito. Todos os parâmetros são do tipo Parâmetro e se têm duas outras hierarquias: se é por valor (*Parâmetro Valor*) ou por referência (*Parâmetro Referência*), ou se trata de configuração (*Parâmetro Configuração*)[5] ou de evento (*Parâmetro Evento*). A idéia dessa diferenciação é que a *Chamada de Função Adaptativa* deve apenas considerar valores, enquanto que as *Ações Adaptativas* podem considerar valores e referências (que são resolvidas durante a execução). Ao definir as ações, entretanto, é fundamental que se diferencie o que é configuração e o que é evento. Dessa forma, caso um desenvolvedor use por engano um parâmetro que representa um evento em uma função adaptativa que deveria receber uma configuração, em tempo de execução haverá um conflito de tipos, sendo gerada uma exceção (e não tentando executar a função adaptativa). Além disso, em tempo de compilação se detecta o uso de variáveis de tipos errados (como, por exemplo, ao tentar criar uma ação adaptativa de inserção em que a configuração inicial é representada equivocadamente como um evento).

[5] Existe um tipo especial de parâmetro de configuração e que também é parâmetro de referência que trata dos geradores (*Parâmetro Gerador*).

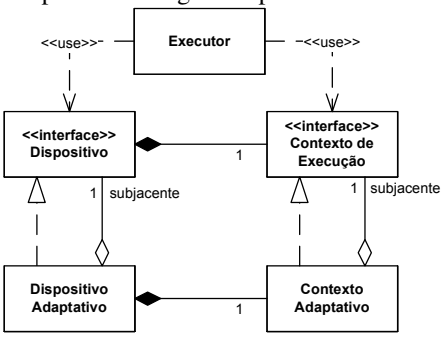


Fig. 4. As classes de execução.

Outra parte fundamental da biblioteca são as classes de execução, apresentadas na Fig. 4. A execução em si é feita pela classe *Executor* que usa um *Contexto de Execução*. Essa separação é brevemente discutida a seguir, na Seção III.B.3. Um exemplo simplificado da execução da biblioteca é apresentado na forma de um diagrama de seqüência na Fig. 8. Buscou-se seguir o pseudo-algoritmo descrito em [3], levando em conta as considerações e limitações da biblioteca. Com isso, a idéia é que o *Executor* chama o *Contexto* para obter as regras e aplicá-las. Por sua vez, o *Contexto* pede para que a *Regra Adaptativa* seja aplicada, fazendo a *Chamada de Função Adaptativa* e aplicando a *Regra* subjacente (não descrito no diagrama, mas usando o *Contexto* subjacente).

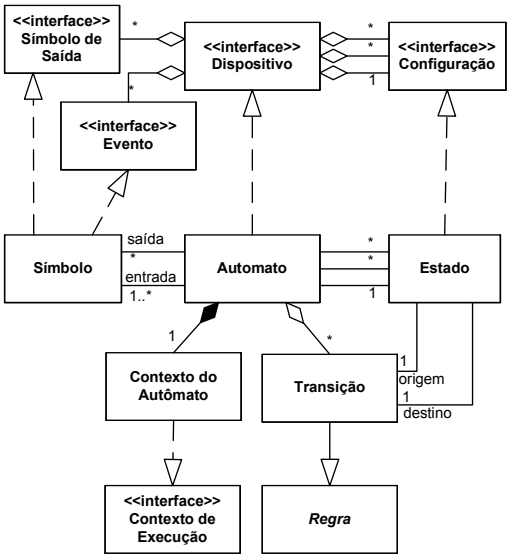


Fig. 5. As classes que tratam do autômato como dispositivo subjacente.

Por fim, na Fig. 5 são apresentadas as classes que tratam da implementação do autômato como dispositivo subjacente. Atualmente a biblioteca disponibiliza apenas esse dispositivo, mas caso seja necessário um outro dispositivo, o desenvolvedor deverá implementar as interfaces e classes abstratas que representam os conceitos de dispositivo e os conceitos associados a ele, ou seja, *Dispositivo*, *Contexto de Execução*, *Regra*, *Configuração*, *Evento* e *Símbolo de Saída*. Por exemplo, no caso do autômato, as classes criadas foram respectivamente *Autômato*, *Contexto do Autômato*, *Transição*, *Estado* e *Símbolo* (implementando ao mesmo tempo *Evento* e *Símbolo de Saída*). O desenvolvedor pode reusar essas classes para implementar outros tipos de autômato como dispositivos, apenas redefinindo e adicionando operações e atributos que sejam necessários.

Aos interessados em maiores detalhes sobre a execução e as classes, recomenda-se olhar o código fonte6.

2) Ponto de vista de Implementação

As classes definidas para a biblioteca foram organizadas em 4 pacotes, conforme apresentado no diagrama de pacotes da Fig. 6. O pacote raiz contém as interfaces e classes que definem o que é dispositivo e permitem a sua execução. O pacote Adaptativo contém os elementos referentes a adaptatividade e no seu sub-pacote Função estão as classes e interfaces que tratam das funções adaptativas e a hierarquia de tipos de parâmetros. As classes referentes ao mecanismo subjacente estão organizadas no pacote Subjacente. Por enquanto nesse pacote estão disponíveis apenas as classes que permitem um autômato como dispositivo subjacente.

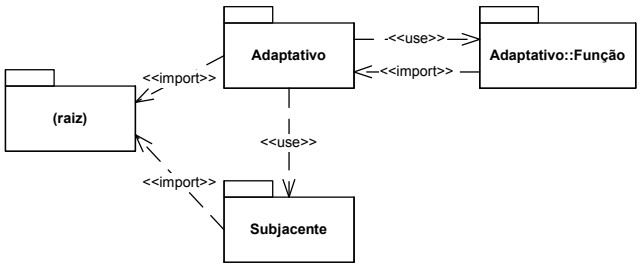


Fig. 6. Os pacotes definidos para organizar as classes da biblioteca.

As únicas dependências externas da biblioteca tratam do *log*, usando o Log4j [8]. Isso é apresentado na forma de um diagrama de componentes na Fig. 7. O artefato "log4j.xml" é necessário para definir qual o nível de *log* permitido e onde as informações de *log* serão apresentadas.

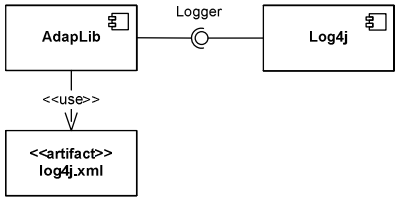


Fig. 7. As dependências externas da *AdapLib*.

3) Ponto de vista de Processo

Como atualmente o software utiliza apenas um *thread* de execução e não são especificados requisitos não funcionais de desempenho, este ponto de vista não será detalhado. Em futuras versões em que será tratada a questão do não determinismo, possivelmente haverá uma necessidade de trabalhar com mais de um *thread* de execução. Considerando essa necessidade, decidiu-se separar a execução do dispositivo do dispositivo em si, criando a classe *Contexto de Execução*.

[6] O código fonte da biblioteca *AdapLib* está disponível em: http://www.geocities.com/fabiolevy/projetos/adaplib/.

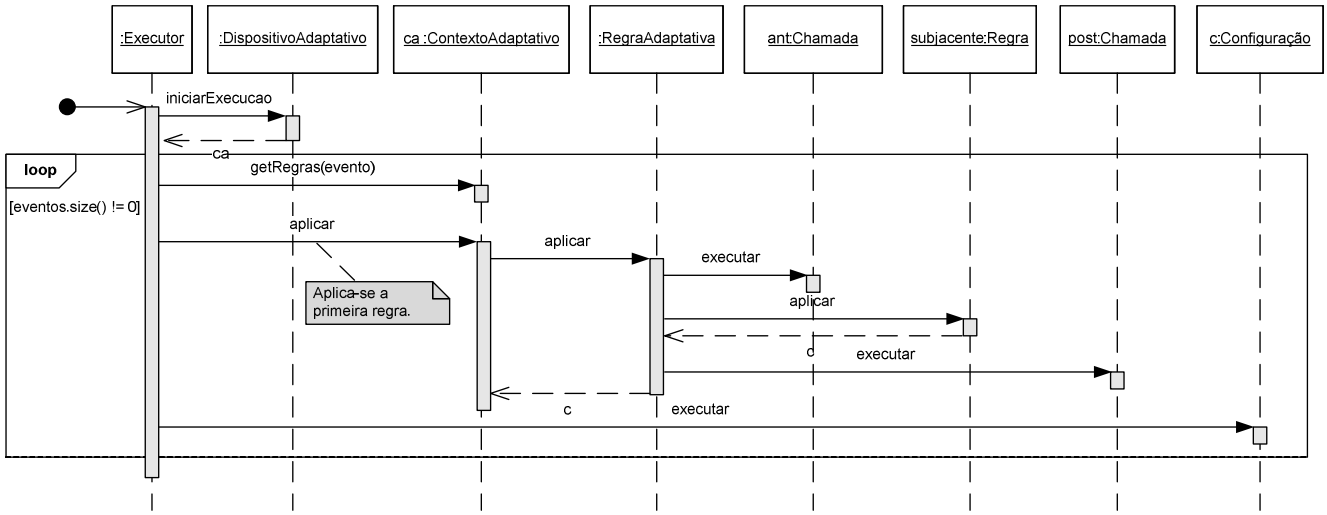


Fig. 8. O cenário de execução de um dispositivo adaptativo.

A intenção é permitir que existam diferentes contextos, um para cada execução possível. Entretanto, como ainda não foram analisados os detalhes de possíveis soluções, provavelmente serão necessárias outras mudanças no modelo em questão.

4) Ponto de vista de Implantação

Por ser uma biblioteca, existem poucas questões de implantação. Atualmente a biblioteca é distribuída em duas versões: o código fonte e um arquivo Jar (já que a biblioteca é feita em Java). Esse arquivo Jar permite agregar todos os arquivos compilados em um só, podendo ser usado diretamente por desenvolvedores que trabalham com a biblioteca.

5) Ponto de vista de caso de uso

O principal cenário de uso da *AdapLib* é a execução de um dispositivo adaptativo. Seguindo a filosofia da biblioteca, o cenário busca transpor para o software o algoritmo de funcionamento de um dispositivo adaptativo apresentado em [3] – considerando as limitações existentes. Esse cenário de execução é apresentado de forma simplificada na Fig. 8.

O responsável pela execução de um dispositivo é a classe *Executor*. Ela solicita ao *Dispositivo Adaptativo* um *Contexto de Execução Adaptativo* e assim começa a executar o dispositivo, usando as informações desse contexto. Enquanto houver eventos[7], pede-se ao contexto *Regras Adaptativas* que possam ser aplicadas. Caso não haja regras a execução termina, rejeitando a entrada. Se houver regras, aplica-se uma delas (a primeira de uma seqüência). Essa aplicação é feita ao chamar a função adaptativa anterior (se ela estiver definida), ao aplicar a regra subjacente e ao chamar a função adaptativa posterior (se ela estiver definida), nessa ordem. Se ao chamar a função adaptativa anterior à estrutura do dispositivo for alterada de tal forma que a regra que está sendo aplicada não exista mais, interrompe-se a aplicação da regra e o laço é reiniciado. Entretanto, caso a regra atual seja removida ao executar a função adaptativa posterior, nada é feito. Com a aplicação da regra, o dispositivo atinge uma nova configuração, que é executada e que com isso se volta ao laço para aplicar mais regras.

C. Trabalhos relacionados

Na literatura existem alguns outros trabalhos que propõem criar ambientes de execução para dispositivos adaptativos. O principal deles é o *AdapTools*, que é uma ferramenta para aprendizado, implementação e depuração de dispositivos adaptativos [9]. Nessa ferramenta é possível representar autômatos de pilhas estruturados e autômatos de estados finitos usando uma linguagem própria (linguagem *AdapTools*). Além disso, a ferramenta também provê um compilador para a linguagem *AdapMap*, gerando código na linguagem da ferramenta. Dentre as principais funcionalidades dessa ferramenta está a possibilidade de ver em tempo de execução o funcionamento do autômato, as regras aplicadas, criadas e removidas (organizadas em uma tabela) e a possibilidade de criar rotinas semânticas, executadas após o término do processamento da cadeia. Além disso, é também possível trabalhar com não-determinismos e armazenar as regras em um banco de dados (buscando melhorar a eficiência da execução do autômato).

O *AdapTools* é, portanto, um ambiente mais completo e maduro (desenvolvido desde 2001) que a *AdapLib* – que é apenas uma biblioteca para a execução. Por mais que o *AdapTools* possa ser usado de forma "embutida" em outra aplicação [2], não há muitas informações de como fazê-lo – ou até mesmo de como estendê-la para tratar de detalhes específicos de algumas soluções. Além disso, a ferramenta trata de autômatos e não dispositivos adaptativos em geral. Dessa maneira, a *AdapLib* pretende ser uma biblioteca simples e focada apenas na execução de dispositivos, sendo possível projetar um dispositivo usando as abstrações definidas na própria linguagem de programação – usando para isso os conceitos da orientação a objetos.

Alguns outros trabalhos não apresentam ferramentas, mas provas de conceito que podem ser usadas para simulação de autômatos adaptativos. Werneck [10] analisa a viabilidade de uma implementação usando recursão para a aplicação das

[7] Na realidade a condição para continuar a execução é que ou haja eventos de entrada ou que a configuração atual não seja final e existam regras que não consomem eventos. Isso foi feito para permitir a existência de regras sem eventos e ainda assim não implementar o não determinismo.

regras, ao invés de um laço (que seria a forma tradicional). Para isso foi criado um simulador como prova de conceito que usa uma linguagem específica para definição de autômatos adaptativos (SDMBA) e que permite executar um autômato adaptativo a partir de uma entrada. Esse simulador, entretanto, apresenta algumas limitações frente à teoria de autômatos adaptativos: não é possível trabalhar com ações de remoção e de busca, não é possível passar parâmetros para as funções adaptativas e trata-se apenas de autômatos determinísticos.

De forma mais abstrata, Vega [5] discutiu o projeto de um ambiente de execução de autômatos adaptativos, apresentado como um mini-curso no WTA 2008. Por mais que não seja proposta uma ferramenta em si, uma prova de conceito foi discutida e apresentada – servindo como base para a discussão de problemas e de detalhes de implementação.

IV. Estudo de Caso

Um dos principais objetivos da biblioteca *AdapLib* é que ela possa ser facilmente absorvida por aplicações que usem os conceitos de dispositivos adaptativos. Para discutir como a biblioteca pode ser usada – em um ponto de vista de implementação – será apresentado um projeto que usou a biblioteca para a criação de uma prova de conceito (POC).

O projeto em questão era a criação de um software para o monitoramento e diagnóstico de problemas em um portal de notícias *online*. O monitoramento precisaria considerar os ambientes de produção e de administração do portal, levando em conta as máquinas existentes (servidores *web*, servidores de aplicação e servidores de banco de dados) e alguns aplicativos específicos. Com as informações obtidas através de sondas existentes nas máquinas e nos aplicativos, era necessário diagnosticar o problema existente, buscando assim direcionar o trabalho da equipe de manutenção do portal. A solução adotada deveria considerar a grande quantidade de acessos ao portal e a possibilidade de inserção de novas máquinas e aplicativos, além da preocupação em se obter uma alta disponibilidade para o portal. Considerando que o objetivo do estudo de caso neste trabalho é apenas analisar o uso da *AdapLib*, não serão aqui discutidos maiores detalhes sobre o contexto e o racional para se chegar a solução aqui apresentada.

A solução adotada para a prova de conceito foi a criação de um autômato de estados finitos adaptativo que usava como alfabeto de entrada as possíveis informações obtidas por sondas nas máquinas e nos aplicativos. A partir dessas entradas, projetou-se um autômato que permitia diagnosticar problemas de desempenho e de exceções causadas nos aplicativos. Na Fig. 9 é apresentada uma parte do autômato que trata do alarme de exceção – a que trata da análise do problema ocorrida no *Servidor de Aplicação 1*. O estado 1 é o estado inicial e, a partir da informação que o problema foi de exceção (símbolo EXCE), o autômato passa ao estado 2. Nesse estado existem transições para as diversas máquinas consideradas, mas na figura há apenas uma transição para o estado 3 caso o problema seja no *Servidor de Aplicação 1* (consumindo o símbolo SA1). Nesse estado é analisado se o *Servidor de Banco de Dados 1* está ativo ou não (representado por "Obter BD1 ativo"). Se não estiver ativo (símbolo INAT), nada é feito (um outro alarme irá avisar a equipe de manutenção que o *Servidor de Banco de Dados 1* está inativo), seguindo para o estado 4 que apenas consome o restante da entrada e vai para o estado final 9 em que nada é feito. Entretanto, caso o *Servidor de Banco de Dados 1* estiver ativo, se verifica em qual página ocorreu a exceção (usando um coletor de nomes definido nas transições do estado 5 e 6 que usam uma codificação da página em binário). Caso ocorram 5 erros na mesma página é emitido um aviso de erro de página, levando ao estado 10 (isso é feito pela função adaptativa 1_conta5). Das próximas vezes que o erro acontecer, ele é desprezado, criando uma transição para o estado 7. Enquanto não ocorrerem os 5 erros, nada é feito, indo ao estado 4 e posteriormente ao estado 9.

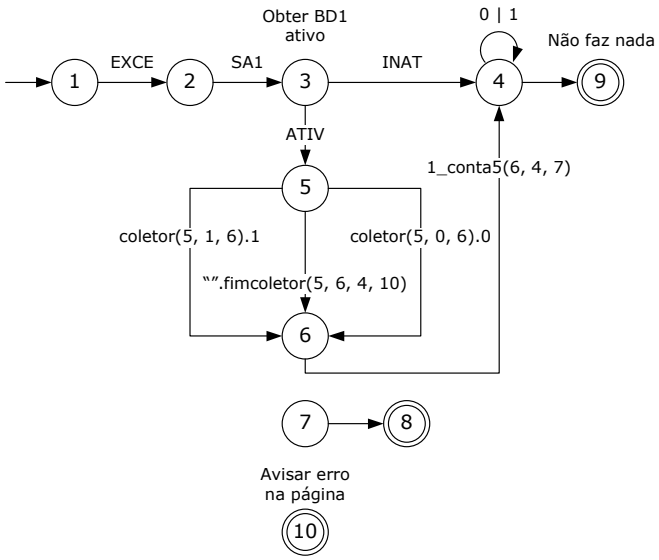


Fig. 9. Parte do autômato de estados finitos adaptativo implementado (os círculos representam estados, as setas transições e os círculos com bordas duplas estados finais).

Para executar o autômato de estados finito projetado foi utilizada a *AdapLib*. Como a biblioteca já possui uma implementação de autômato de estados finitos como dispositivo subjacente (apresentada anteriormente na Seção III.B.1.), só foi necessário adicionar algumas classes para definir o comportamento específico para o projeto. Considerando o contexto, foram necessárias apenas duas mudanças no formalismo de autômato de estados finitos adaptativos: (1) permitir que os estados obtivessem informações e (2) que os estados finais emitissem avisos sobre o problema diagnosticado. A mudança (1) fez com que fosse criado um tipo especial de estado que obtém informações, representado na classe *Estado Obtentor de Informação* e a solução (2) fez com que fossem criados dois outros tipos de estados: um estado final que executava alguma ação na interface homem-computador (*Estado Final Ação*) e um estado final que informava à interface homem-computador o problema diagnosticado (*Estado Final Problema*)[8]. Para representar esses três novos tipos de estado foi necessário apenas criar especializações da classe *Estado* e redefinir a

[8] Isso poderia ser feito ao escrever um símbolo específico na cadeia de saída e processá-lo para gerar a informação adequada na interface homem-computador. Entretanto essa solução exigiria a criação de um símbolo diferente para cada tipo de saída possível – o que, no caso, iria gerar um grande número de novos símbolos. Por causa disso preferiu-se a solução apresentada.

operação *executar*, que permite que um estado tenha um comportamento ao ser alcançado.

```
// Criando o autômato
Automato a = new Automato();

// Colocando uma camada adaptativa no autômato
DispositivoAdaptativo aa =
     new DispositivoAdaptativo(a);

// Estado 1
Estado e1 = new Estado("1");
a.adicionarConfiguracao(e1, true, false); // inicial

// Estado 2
Estado e2 = new Estado("2");
a.adicionarConfiguracao(e2, false, false);

// Estado 3
Estado e3 = new EstadoObtenedorDeInformacao("3",
     "O BD1 está ativo?");
a.adicionarConfiguracao(e3, false, false);

// os outros estados
...

// Criando as transições
a.adicionarRegra(e1, "EXCE", e2); // (1, "EXCE, 2)
a.adicionarRegra(e2, "SA1", e3);  // (2, "SA1", 3)
a.adicionarRegra(e3, "INAT", e4); // (3, "INAT", 4)
a.adicionarRegra(e3, "ATIV", e5); // (3, "ATIV", 5)

// (5, coletor(5, "0", 6)."0", 6)
ArrayList p = new ArrayList();
p.add(new ParametroValorConfiguracao("5"));
p.add(new ParametroValorEvento("0"));
p.add(new ParametroValorConfiguracao("6"));

aa.getMecanismoAdaptativo().adicionarRegraAdaptativa(
     new ChamadaFuncaoAdaptativa(coletor, p),
     e5, "0", e6,
     null);

// as demais transições
...
```

Fig. 10. Fragmento de código exemplificando o uso da *AdapLib* – sublinhado está a referência a um objeto com a função adaptativa do coletor.

Após criar essas novas classes para permitir uma comunicação diferenciada com a interface homem-computador, foi necessário passar para a linguagem de programação o autômato e as funções adaptativas projetadas. Como exemplo disso, uma parte desse código em Java está apresentada na Fig. 10, sublinhando uma variável que referencia a função adaptativa para o coletor de nomes (na transição do estado 5 para o 6)[9]. Cada estado é representado por um objeto e as transições entre eles são adicionadas à camada subjacente (caso não haja chamadas de funções adaptativas) ou à camada adaptativa (que se encarrega de registrar na camada subjacente a regra subjacente).

Uma parte da função adaptativa que trata do coletor de nomes é apresentada na Fig. 11. A função também é um objeto ao qual se deve atrelar as ações adaptativas que sejam necessárias.

[9] Por simplicidade e consistência à descrição arquitetural apresentada (que não representou os parâmetros das classes), o código é apresentado sem os recursos de *Generics* do Java.

```
FuncaoAdaptativa coletor =
     new FuncaoAdaptativa("coletor");
coletor.setGeradores(1);

// - (n0, n1, n2)
AcaoAdaptativaRemocao remocao =
     new AcaoAdaptativaRemocao(
          new ParametroReferenciaConfiguracao(0),
          new ParametroReferenciaEvento(1),
          new ParametroReferenciaConfiguracao(2));
coletor.adicionarAcao(remocao);

// +(n0, n1, g1)
AcaoAdaptativaInsercao insercao =
     new AcaoAdaptativaInsercao(
     new ParametroReferenciaConfiguracao(0),
     new ParametroReferenciaEvento(1),
     new ParametroReferenciaGerador(0));
coletor.adicionarAcao(insercao);

// +(g0, coletor(g0, 0, n2).0, n2)
// Primeiro, definindo os parâmetros da chamada da
// função adaptativa "coletor"
ArrayList parametros = new ArrayList();
parametros.add(new ParametroReferenciaGerador(0));
parametros.add(new ParametroValorEvento("0"));
parametros.add(new ParametroReferenciaConfiguracao(2));

// Criando a ação adaptativa de inserção em si
insercao = new AcaoAdaptativaInsercao(
     new ParametroReferenciaGerador(0),
     new ParametroValorEvento("0"),
     new ParametroReferenciaConfiguracao(2),
     "coletor",  // nome da função adaptativa anterior
                 // que a ser regra inserida chama
     parametros, // os parâmetros da função adaptativa
                 // anterior
     null,       // não há função adaptativa posterior
     null);      // sem parâmetros para a função
                 // adaptativa posterior
coletor.adicionarAcao(insercao);

// as demais ações adaptativas
...
```

Fig. 11. Parte do código que define a função adaptativa "coletor".

Com o autômato completamente representado na linguagem Java foram necessários diversos testes para garantir que o autômato idealizado estava corretamente descrito na linguagem de programação. Além dos erros que aconteceram na passagem para a linguagem de programação, também haviam erros no autômato idealizado – o que dificultou ainda mais a atividade de teste. Entretanto, considerando que o autômato possuía por volta de 50 estados e diversas transições, era esperado que problemas desse tipo acontecessem, principalmente por que o autômato foi apenas executado durante a atividade de testes.

V. CONCLUSÃO

Neste trabalho foi apresentada a fundamentação teórica e a arquitetura da biblioteca para execução de dispositivos adaptativos *AdapLib*. Essa biblioteca busca seguir com fidelidade a teoria de dispositivos adaptativos, tendo como funcionalidades os principais conceitos dessa teoria. Um outro requisito importante no projeto da biblioteca foi permitir que os usuários a estendam, buscando fazer com que aplicações possam usar a biblioteca com facilidade – mesmo nos casos em que a teoria não possa ser seguida à risca ao considerar questões práticas. Além dessas questões, um outro ponto importante na construção da biblioteca foi a eficiência.

Entretanto esse requisito não funcional não foi discutido neste trabalho, cabendo a trabalhos futuros fazê-lo.

Durante a realização do estudo de caso percebeu-se que uma das dificuldades do uso da biblioteca é na depuração do dispositivo (no caso autômato) projetado. Por mais que a biblioteca permita habilitar durante o desenvolvimento informações detalhadas de *log*, não é possível ter uma representação geral do dispositivo que apresente, em tempo real, as mudanças ocorridas na estrutura do dispositivo. Para isso, ferramentas como o *AdapTools* [9] são mais adequadas, já que há uma representação gráfica da execução do autômato. Nesse sentido é importante ressaltar que o intuito não é que a biblioteca *AdapLib* seja uma ferramenta completa para o projeto e a implementação de dispositivos adaptativos. O objetivo é apenas permitir a execução desse tipo de dispositivo.

Na versão atual da biblioteca alguns conceitos definidos pela teoria de dispositivos adaptativos ainda não foram implementados: ação adaptativa de busca e o não determinismo. Além disso, o único dispositivo disponível pela biblioteca é o autômato (o que não impede que outros dispositivos sejam criados) e ainda é preciso uma análise mais criteriosa para permitir múltiplos níveis de adaptabilidade. Em trabalhos futuros pretende-se tratar dessas limitações e adicionar outras funcionalidades à biblioteca, sejam elas para tratar de aspectos teóricos ou de implementação.

AGRADECIMENTOS

À Reginaldo Inojosa Filho pelo auxílio na versão inicial da biblioteca e pela realização em conjunto do estudo de caso.

À FAPESP pelo apoio na realização de meu doutorado, durante o qual foi feito este trabalho.

REFERÊNCIAS

[1] J. J. Neto, "Adaptive automata for context-dependent languages", ACM SIGPLAN Notices, vol. 29, i. 9, pp. 115-124, Sep. 1994.

[2] H. Pistori, "Tecnologia adaptativa em engenharia de computação: Estado da arte e aplicações", Tese de doutorado, orientada por J. J. Neto, Universidade de São Paulo, São Paulo, Brasil, 2003.

[3] J. J. Neto, "Adaptive Rule-Driven Devices - General Formulation and Case Study", Lecture Notes in Computer Science: Implementation and Application of Automata, vol. 2494, pp. 466-470, 2002.

[4] J. R. Almeida, "STAD – Uma ferramenta para representação e simulação de sistemas através de statecharts adaptativos", Tese de doutorado, orientada por J. J. Neto, Universidade de São Paulo, São Paulo, Brasil, 1995.

[5] I. S. Vega, "Implementação OO para Autômatos Adaptativos: Impactos das Tomadas de decisão efetuadas durante o Design", Resumo do Mini-curso, Segundo Workshop de Tecnologia Adaptativa (WTA 2008), São Paulo, pp. viii-ix, 2008.

[6] Systems and software engineering - Recommended practice for architectural description of software-intensive systems, ISO/IEC 42010 (IEEE 1471-2000), Jul. 2007.

[7] P. Kruchten, "The 4+1 View Model of Architecture", IEEE Software, vol. 12, no. 6, pp. 42-50, Nov. 1995.

[8] Apache Software Foundation, "Apache Log4J", disponível em: http://logging.apache.org/log4j/, Acesso em 08 de outubro de 2008.

[9] L. Jesus, D. G. Santos, A. A. Castro JR. e H. Pistori, "AdapTools 2.0: Aspectos de Implementação e Utilização", IEEE Latin America Transactions, vol. 5, no. 7, pp. 527-532, Nov. 2007.

[10] N. L. Werneck, "Biblioteca para autômatos adaptativos com regras de transição armazenadas em objetos feita em C++", Segundo Workshop de Tecnologia Adaptativa (WTA 2008), São Paulo, pp.22-26, 2008.

Fábio Levy Siqueira é Engenheiro de Computação formando na Escola Politécnica da Universidade de São Paulo (2002) e Mestre em Engenharia Elétrica pela mesma instituição (2005).

Atualmente é doutorando em Engenharia Elétrica e pesquisador na Escola Politécnica da Universidade de São Paulo. Tem experiência na área de Engenharia da Computação, com ênfase em Engenharia de Software, atuando principalmente nas seguintes áreas: engenharia de requisitos, qualidade de software e desenvolvimento distribuído de software.

Recomendação de Recursos utilizando Autômato Adaptativo

P. R. M. Cereda, R. A. Gotardo e S. D. Zorzo

***Resumo*— Sistemas de Recomendação identificam as preferências de um dado usuário, bem como do restante da comunidade, em relação aos recursos disponibilizados, visando oferecer recursos personalizados aos usuários. Este artigo apresenta um sistema de recomendação que utiliza um autômato adaptativo para analisar a relação entre recursos e usuários e determinar as possíveis recomendações personalizadas. São apresentados os aspectos de implementação e um experimento para verificação e análise das recomendações geradas no sistema desenvolvido.**

***Palavras-chave:*— Sistemas de Recomendação, Autômato Adaptativo, Personalização.**

I. Introdução

Com o crescimento e difusão da Internet, a grande maioria dos websites passou a oferecer serviços e recursos personalizados aos usuários. As informações relativas aos seus acessos passaram a ser valiosas para os sites, a ponto de realizarem a coleta destas informações, muitas vezes sem consentimento explícito.

A oferta de serviços e recursos ao usuário é possível com o emprego de diversos métodos. Um deles é o Sistema de Recomendação, que tenta identificar quais são os recursos mais importantes para este. Esses recursos constituem sugestões a serem apresentadas ao usuário, de acordo com um determinado critério.

Este artigo apresenta um sistema de recomendação, denominado *RecomAA*, que utiliza um autômato adaptativo para verificar a existência de alguma relação entre recursos. O autômato adaptativo foi escolhido devido à sua simplicidade, capacidade de modificação autônoma e poder computacional.

A organização deste artigo é a seguinte: a Seção II apresenta as principais vantagens da personalização de serviços e recursos, e as funcionalidades oferecidas aos sites. A Seção III contextualiza os sistemas de recomendação e suas principais características. O sistema proposto, utilizando um autômato adaptativo, é apresentado na Seção IV. A Seção V apresenta um experimento realizado para analisar as recomendações geradas. Os aspectos de implementação são definidos na Seção VI. As conclusões são apresentadas na Seção VII.

II. Personalização de Serviços e Recursos

A personalização pode ser descrita como tornar algo pessoal, individual, dependente das características e dos interesses humanos. Ao personalizar um objeto de acordo com um usuário, cria-se uma relação de afinidade. De acordo com Grande [1], "um produto ou serviço pode atender as necessidades fundamentais de uma pessoa por suas funcionalidades e características primárias. Além disso, um serviço, através da personalização, pode possuir determinadas características que o torna mais parecido com um indivíduo. Essas qualidades secundárias são consideradas tão importantes que em muitos casos a escolha do produto ou serviço é regida somente através delas."

Para utilizar a personalização, algumas informações relevantes necessitam ser obtidas para saber algo a respeito da preferência de um usuário [2].

A personalização apresenta algumas vantagens, tais como:

- **Sistema "mais próximo" do usuário:** o usuário sente-se mais à vontade, pois a maioria das informações que o site retorna o agrada.
- **Perfil diferente para cada usuário:** um site de e-commerce que oferece perfis diferentes para os usuários pode aumentar suas vendas.
- **Tendências de comportamento:** a partir dos perfis dos usuários, é possível prever tendências de comportamento e estabelecer grupos de interesse.
- **Melhorias na navegabilidade:** um site pode prover menus de navegação diferentes, dependendo dos interesses dos usuários.

A personalização pode oferecer a um site algumas funcionalidades. Kimball e Merz [3] citam algumas delas:

- **Reconhecimento de re-visitas:** ter a possibilidade de identificar o usuário como um novo cliente ou um cliente já existente.
- **Interface de usuário e personalização de conteúdo:** o usuário pode ter vários tipos de menus de acesso rápidos, links de produtos, propagandas e anúncios, de acordo com sua preferência.
- **Vendas colaterais:** são recomendações a partir de conhecimentos prévios do cliente.
- **Vendas por impulso:** são sugestões geralmente relacionadas a um determinado grupo de interesse. Por exemplo, se o cliente comprou um determinado livro, o site poderá relatar outros produtos comprados por outros usuários que também compraram o mesmo livro. Freqüentemente, esse tipo de personalização tem o título de "*Clientes que compraram este produto também compraram:*".
- **Filtragem colaborativa ativa:** são sugestões e indicações de clientes. Por exemplo, o cliente tem a

Paulo Roberto Massa Cereda, Reginaldo Aparecido Gotardo e Sérgio Donizetti Zorzo.

Os autores são do Departamento de Computação, Universidade Federal de São Carlos, Rodovia Washington Luís, Km 235, Caixa Postal 676, 13565-905 – SP, (e-mails: paulo_cereda@dc.ufscar.br, reginaldo_gotardo@dc.ufscar.br e zorzo@dc.ufscar.br).

possibilidade de dar nota a um determinado produto, escrever um comentário ou recomendar a um amigo.

- **Eventos de calendário:** os produtos podem ser relacionados de acordo com eventos baseados em feriados ou datas importantes, por exemplo, dia das Mães, Páscoa, Natal, entre outros.
- **Eventos de estilo de vida:** os produtos são relacionados de acordo com situações comuns do dia-a-dia, por exemplo, nascimento de um filho, casamento, formatura, entre outros.
- **Localização:** o conhecimento prévio de um cliente permite a personalização de um site em relação a seu idioma ou localização geográfica.

Há um grande benefício na utilização da personalização por sites, principalmente os de e-commerce. Com isso, diversas técnicas de coleta e ferramentas são criadas ou melhoradas para facilitar a tarefa de prover personalização.

Existem diversos mecanismos para coleta de dados para oferecer personalização na Internet. Geralmente são utilizados análise de dados em formulários e análise de navegação do usuário [4]. Com os dados obtidos, são aplicadas técnicas de mineração de dados para determinar preferências, tendências, entre outros.

É importante ressaltar que os próprios navegadores enviam informações interessantes, como sistema operacional do usuário, idioma, tipo de navegador, a página de referência, e outros, podendo essas informações ser utilizadas para serviços de personalização.

Os mecanismos mais comuns de coleta de dados podem ser implementados por *cookies*, *clickstream*, *logs* de servidor, *web bugs* e formulários, que são detalhados a seguir.

O *cookie* é um grupo de dados trocados entre o usuário e o servidor, armazenados em um arquivo texto criado no computador do primeiro [5]. As informações armazenadas nos *cookies* podem refletir algo sobre o perfil do usuário. O *cookie* tem o objetivo de manter a persistência de sessões HTTP, mantendo o último estado antes de terminar a conexão, e retorná-lo no próximo acesso.

Pelo fato deste mecanismo estar incluído em todos os navegadores (e, na maioria dos casos, ativado por padrão), ele persistirá durante muito tempo como a ferramenta primária para prover personalização [3].

O *clickstream*, também conhecido como seqüência de cliques ou *clickpath*, representa o caminho que o usuário percorre enquanto está visitando um determinado site. A seqüência de cliques obtida, quando aplicadas de forma correta, pode proporcionar informações muito importantes para as empresas [4].

Os dados de *clickstream* também podem ser coletados por Provedores de Serviço de Internet (ISPs), painéis de acesso ou mesmo manualmente, apesar da coleta através dos arquivos de *log* dos servidores ser mais comum [6].

O *clickstream* é muito importante, do ponto de vista comercial, pois é possível identificar as preferências e os padrões de comportamento do usuário; isso inclui qual área lhe interessa, a freqüência que a procura e quais as informações úteis para criar estratégias de marketing mais direcionadas ao usuário e, conseqüente, maior chance de sucesso [7].

Todos os servidores Web têm a capacidade de registrar as requisições em arquivos de *log* ou banco de dados. Os dados de *log* de um servidor Web são a fonte primária de dados do *clickstream*, pois toda vez que o servidor Web responde uma requisição HTTP, uma entrada é anotada no arquivo de *log*. As entradas do arquivo de *log* seguem um padrão estendido chamado *Extended Common Log Format* (ECLF) [3].

Esse tipo de análise de *log* oferece um desafio: o servidor pode estar, em um determinado momento, mantendo centenas ou até milhares de sessões de usuário simultaneamente. Os registros individuais que abrangem os rastros da sessão de um determinado usuário estarão dispersos por todo o log [3].

Outro problema relatado sobre os arquivos de *log* diz respeito aos *Crawlers*, *Bots*, *Spiders* e *Robots*, que são pequenos programas que são executados com a intenção de coletarem dados ao visitarem páginas Web. Esses programas interferem significativamente nos padrões de *clickstream*, poluindo um arquivo de *log* com informações que não foram geradas por um verdadeiro usuário [8].

Web Bugs são mecanismos que tentam obter algum tipo de identificação de um usuário. Geralmente, estão inseridos em mensagens de e-mail ou páginas da Web. Uma página Web ou um e-mail podem ser visualizados sem que se perceba a presença de um *Web Bug*. No momento de carregamento da página Web ou do e-mail, a imagem referente ao *Web Bug* é requisitada a um servidor distinto que, por sua vez, obtém as informações do cabeçalho HTTP da requisição, contendo as informações a respeito do usuário.

Os *Web Bugs* podem trabalhar em conjunto com os cookies, monitorando quais sites o usuário visita. Dessa forma, os banners de anúncios são específicos para cada um [9].

A linguagem HTML apresenta controles que permitem enviar informações para um determinado processamento no servidor. Tais informações são preenchidas utilizando formulários eletrônicos; através dos métodos POST ou GET fornecidos pelo protocolo HTTP, o servidor recebe a requisição e realiza o tratamento das informações submetidas.

Através dos formulários, podem ser realizados dois tipos de coleta: a coleta explícita, onde o envio das informações pelo usuário envolve seu consentimento, e a coleta implícita, onde o usuário pode enviar informações adicionais, sem tomar conhecimento da existência delas. Em ambos os casos, as informações são armazenadas no lado do servidor, por exemplo, em um banco de dados.

Os mecanismos de coleta podem ser combinados para um particular sistema de recomendação.

A *Mineração de Dados* é um conjunto de técnicas que visam a aquisição de conhecimentos em bancos de dados. A motivação para esse mecanismo vem da dificuldade em obter conhecimento relevante de grandes volumes de dados, logo, é necessário o uso de ferramentas e técnicas que facilitem essa árdua tarefa [10].

Na Web, a descoberta de informações relevantes de padrões de navegação do usuário, a ponto de descrever seu comportamento, é extremamente valiosa para empresas e

organizações que trabalham com e-commerce.

As técnicas de mineração podem utilizar as mais diversas abordagens de implementação. As mais comuns incluem uso de técnicas estatísticas, raciocínio baseado em casos, redes neurais, árvores de decisão, algoritmos genéticos, entre outros.

A utilização da Mineração de Dados por empresas de e-commerce é imprescindível, pois pode revelar informações valiosas que não podem ser obtidas pelos métodos tradicionais.

Os dados obtidos através dos mecanismos de coleta podem ser analisados utilizando-se mineração de dados para revelar perfis de navegação e comportamento dos usuários.

III. Sistemas de Recomendação

O *Sistema de Recomendação* é um método tradicional para retornar informações relevantes ao usuário. De um modo geral, um sistema de recomendação tenta identificar quais são os recursos mais importantes para um determinado usuário e recomendá-los [11], [12].

Para recomendar um determinado recurso a um usuário, os sistemas de recomendação podem utilizar-se de duas técnicas para tal:

- **Filtragem colaborativa:** a recomendação é baseada nas preferências do usuário. Para isso, utiliza informações obtidas de outros usuários. Em outras palavras, se os interesses do usuário u_i são semelhantes aos do usuário u_j, os recursos que constituem as preferências de u_j podem ser recomendados a u_i. A filtragem colaborativa tenta avaliar as semelhanças entre os usuários através de seus históricos (associando recursos) ou perfis (associando características).
- **Filtragem baseada em conteúdo:** a recomendação é baseada no perfil do usuário. Para isso, classifica os seus recursos associados e recomenda recursos semelhantes aos que o usuário já avaliou ou possui.

As duas técnicas possuem algumas deficiências. Na filtragem colaborativa, não é possível tratar eficientemente novos usuários ou novos recursos, além da qualidade depender de um grande conjunto de dados históricos. Na filtragem baseada em conteúdo, não é possível explorar novas categorias de recursos além das que o usuário avaliou no passado, e também é necessário um especialista de domínio para avaliar a aquisição de conhecimento. Para minimizar os problemas encontrados, foram criados os *sistemas híbridos*, que combinam a filtragem colaborativa e a filtragem baseada em conteúdo, utilizando os pontos fortes de cada técnica e permitindo uma recomendação mais inerente às necessidades do usuário.

Segundo Reategui e Cazella [13], "os websites de comércio eletrônico são atualmente o maior foco de utilização dos sistemas de recomendação, empregando diferentes técnicas para encontrar os produtos mais adequados para seus clientes e aumentar deste modo sua lucratividade."

IV. Recomendação utilizando Autômato Adaptativo

Esta seção apresenta o sistema de recomendação proposto utilizando um autômato adaptativo para verificar se uma determinada recomendação é válida.

A. Autômato Adaptativo

O *Autômato Adaptativo*, proposto por Neto [14], 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 [15].

Um *autômato adaptativo M* é definido por $M = (Q, S, \sum, \Gamma, P, q_0, Z_0, F)$, 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 sub-máquinas,
- $\sum$ é o alfabeto de entrada, $\sum \subset \sum^A$, $\sum^A$ 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 $P: Q^A \times \sum^A \times \Gamma^A \rightarrow Q^A \times (\sum^A \cup \{ \varepsilon \}) \times (\Gamma^A \cup \{ \varepsilon \}) \times H^0 \times H^0 \}$, H^0 definido a seguir,
- $q_0 \in Q$ é o estado inicial,
- $Z_0 \in \Gamma$ é o símbolo inicial da pilha, e
- $F \subset Q$ é o conjunto de estados finais.

Os conjuntos "^{A}" ("*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 "^{A}" [16].

H^0 é o conjunto de todas as funções adaptativas no autômato adaptativo M. Define-se $H^0 = \{ f \mid f: E \times G_1 \times G_2 \times \ldots \times G_k \rightarrow E \}$, onde f é uma função, $k \in \mathbb{N}$ é o número de argumentos em f, e $G_i = Q^A \cup \sum^A \cup \Gamma^A$ [16].

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 $N = (Q', \sum', \Gamma', P', q_0, Z_0, F)$, onde $Q' \subset Q^A$, $\sum' \subset \sum^A$, $\Gamma' \subset \Gamma^A$, $P': Q^A \times \sum^A \times \Gamma^A \rightarrow Q^A \times (\sum^A \cup \{ \varepsilon \}) \times (\Gamma^A \cup \{ \varepsilon \}) \times H^0 \times H^0 \}$. Observe que q_0, Z_0 e F são os mesmos em qualquer $N \in E$ [16].

O conjunto de todas as sub-máquinas do autômato adaptativo M é representado por S. Cada *sub-máquina* s_i é definida como $s_i = (Q_i, \sum_i, P_i, q_{i0}, F_i)$, onde:

- $Q_i \subseteq Q$ é o conjunto de estados da sub-máquina s_i,
- $\sum_i \subseteq \sum$ é o alfabeto de entrada da sub-máquina s_i,
- $P_i \subseteq P$ é o mapeamento da sub-máquina s_i,
- $q_{i0} \in Q_i$ é o estado inicial da sub-máquina s_i, e
- $F_i \subseteq Q_i$ é o conjunto de estados finais da sub-máquina s_i.

A linguagem aceita por um autômato adaptativo M é dada por $L(M) = \{ w \in \sum^* \mid (q_0, w, Z_0) \vdash^* (q_f, \varepsilon, Z_0)$, onde $q_f \in F \}$. A capacidade de auto-modificação confere ao autômato

adaptativo o poder computacional de uma Máquina de Turing [17].

B. Sistema de Recomendação utilizando Autômato Adaptativo

O autômato adaptativo foi utilizado para modelar um sistema de recomendação simples, chamado *RecomAA*. Dados dois recursos, o sistema verifica se existe alguma relação entre eles. Se a relação existe, o segundo recurso pode ser recomendado em função do primeiro e vice-versa. Por exemplo, dados os recursos *agenda* e *livro*, deseja-se saber se são relacionados; em outras palavras, o sistema responde *sim* ou *não* à seguinte pergunta: *dado o recurso agenda, é possível recomendar o recurso livro?*, ou *dado o recurso livro, é possível recomendar o recurso agenda?*.

Além de verificar se os recursos são relacionados, o sistema *RecomAA* permite que dois recursos sejam avaliados para uma possível criação de relação entre eles. Por exemplo, suponha que os recursos *agenda* e *livro* não são relacionados; entretanto, existem vários clientes que, ao comprar uma agenda, também incluem um livro em sua compra. Os recursos *agenda* e *livro* não são relacionados, mas a *distância* entre eles diminui; isso significa que, em um determinado momento, a distância entre eles indicará a criação de uma relação entre *agenda* e *livro*.

Inicialmente, o autômato adaptativo M do *RecomAA* é definido com uma configuração contendo todos os recursos existentes em um determinado cenário, por exemplo, todos os produtos de uma loja de comércio eletrônico. Durante seu tempo de vida, M sofrerá alterações em sua topologia para reduzir distâncias entre recursos até o estabelecimento de relações. Isto é feito através da função adaptativa f de M, que pode alterar o autômato.

Suponha que o sistema *RecomAA* possua um conjunto $\Pi = \{ p_1, p_2, \ldots, p_n \}$ de recursos. Assim, o autômato adaptativo M possuirá um alfabeto $\sum = \Pi \cup \{ link \}$, onde *link* é um símbolo reservado utilizado para reduzir as distâncias entre recursos.

As cadeias a serem submetidas ao autômato adaptativo M podem ser nos seguintes formatos:

- $p_i\ p_k$, representado uma consulta sobre a existência de uma relação entre dois recursos p_i e p_k, e
- $<link>p_i\ p_k$, representando uma avaliação sobre a relação entre os dois recursos p_i e p_k,

com p_i, $p_k \in \Pi$. Assim, a linguagem reconhecida pelo autômato adaptativo M denota a existência de uma relação válida entre dois recursos ou uma avaliação para uma possível criação de uma relação entre tais recursos.

A Figura 1 apresenta um exemplo de configuração inicial do autômato adaptativo M utilizado no sistema de recomendação *RecomAA*. O cenário representado possui apenas dois recursos, p_1 e p_2 não relacionados entre si.

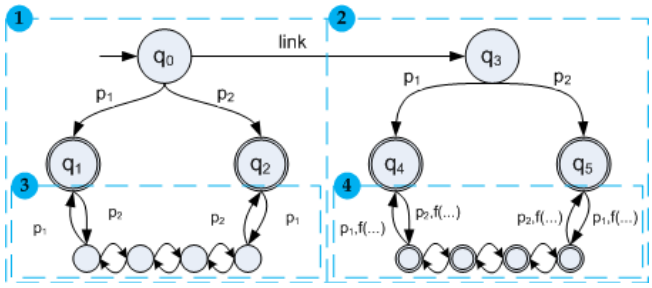


Figura 8. Exemplo do Autômato Adaptativo M utilizado no sistema *RecomAA*.

Para ilustrar a execução, o autômato adaptativo M (Figura 1) foi dividido em 4 regiões, apresentadas a seguir.

- **Região 1**: constitui a verificação das relações entre os recursos. No exemplo, os recursos p_1 e p_2 estão definidos nessa região.
- **Região 2:** essa região trata da avaliação dos recursos. A partir de cada estado alcançável por uma transição consumindo um símbolo representando um recurso, existe um caminho definido na região 3 que leva até os outros estados. Um *caminho* é constituído por um conjunto de estados intermediários (no exemplo, é representado por estados de tamanho menor) que possuem transições entre si e que conduzem a um determinado estado de destino. O objetivo de um caminho é representar a distância existente para a recomendação de um recurso, dado um outro recurso. No exemplo, como existem apenas dois recursos, existirá apenas um caminho entre q_4 e q_5, com transições consumindo p_2 de q_4 até q_5 e transições consumindo p_1 de q_5 até q_4.
- **Região 3:** a região 3 contém todos caminhos de cada estado até os demais. Quando o caminho for representado por apenas duas transições ligando dois estados que representam os recursos, sem estados intermediários compondo esse caminho, existirá relação entre esses dois recursos. Por exemplo, se q_1 possuir uma transição consumindo p_2 até q_2 e se q_2 possuir uma transição consumindo p_1 até q_1, os dois recursos têm relação entre si e podem ser recomendados.

  Seja $\Pi = \{ p_1, p_2, \ldots, p_n \}$ o conjunto de recursos. O total de caminhos existentes no autômato adaptativo M será igual à $C(n, 2)$ para a região 3 e $C(n, 2)$ para a região 4. $C_{n,i}$ é uma combinação simples de n elementos, i a i, representada pela seguinte fórmula:

$$C_{n,i} = \frac{n!}{i!\,(n-i)!} \qquad (1)$$

  $C_{n,i}$ (Fórmula 1) denota o número total de combinações de n elementos tomados i a i, onde i é o número de elementos presentes em cada combinação.
- **Região 4:** essa região possui os caminhos definidos para cada estado que representa um recurso. Como o caminho representa a distância existente para a recomendação de um recurso, para cada transição existente no caminho, existe uma função adaptativa f associada. O objetivo da função f é remover um estado intermediário desse

caminho na região 4 e também na região 3, reduzindo assim a distância do caminho existente para a recomendação de um recurso, até que existam apenas duas transições entre os estados de origem e destino desse caminho.

O *tamanho de um caminho*, isto é, o número de estados intermediários componentes, deve refletir um número razoável de ocorrências de um determinado recurso, dado outro recurso, para que a recomendação seja válida. Valores menores podem criar recomendações não desejáveis, ao passo que valores maiores podem retardar a criação de recomendações mais propensas. No exemplo apresentado na Figura 1, o tamanho do caminho é igual a 4.

É interessante contar com um *especialista de domínio* para analisar o domínio desejado e traçar tamanhos de caminhos mais inerentes à aplicação. Além disso, é possível realizar, de tempos em tempos, análises e eventuais alterações na configuração do autômato de modo a representar com maior consistência as recomendações do sistema.

V. Experimento e Análise

Para verificar as recomendações do sistema *RecomAA*, foi realizado um experimento com 15 alunos de graduação e pós-graduação do Departamento de Computação da Universidade Federal de São Carlos. A primeira parte do experimento consistiu no preenchimento de uma ficha contendo uma relação com 10 livros, apresentados na Tabela I, e 5 campos para resposta. Cada aluno simulou 5 compras e anotou os produtos comprados na ficha. A amostra utilizada no experimento é descrita na Tabela II.

Tabela I
Livros utilizados no experimento.

Índice	Nome do Livro
p_1	O caçador de pipas
p_2	Quando Nietzsche chorou
p_3	Ensaio sobre a cegueira
p_4	O vendedor de sonhos
p_5	Crime e Castigo
p_6	Crepúsculo
p_7	A cidade e sol
p_8	A menina que roubava livros
p_9	O guardião de memórias
p_{10}	O vencedor está só

Tabela II
Amostra utilizada no experimento.

Aluno	Compra 1	Compra 2	Compra 3	Compra 4	Compra 5
1	p_2, p_3, p_4, p_7, p_9	$p_1, p_4, p_5, P_6, p_{10}$	p_1, p_2, p_4, p_7, p_8	p_1, p_4, p_5, p_{10}	p_6, p_7, p_8, p_9
2	p_1, p_3, p_4, p_9	p_4, p_5, p_7, p_{10}	p_1, p_2 ,p_8, p_9	p_6, p_7	p_1, p_2, p_8
3	p_1, p_4, p_8, p_{10}	p_2, p_3	p_1, p_2	p_4, p_7, p_9	p_1, p_{10}
4	p_3, p_7, p_{10}	p_1, p_2	p_4, p_8, p_9	p_4, p_7, p_8	p_3, p_4, p_5
5	p_2, p_5	p_1, p_7, p_8, p_9	p_1, p_2, p_7, p_8	p_7, p_9	p_2, p_5
6	p_3, p_4, p_8	p_2, p_4	p_1, p_9	p_7, p_8	p_3, p_8
7	p_1, p_2, p_7	p_2, p_3, p_5	p_4, p_9	p_{10}	p_1, p_4
8	p_{10}	p_7, p_8	p_5, p_9	p_3, p_9	p_1
9	p_2, p_4, p_6	p_3, p_7, p_9	p_{10}	p_7, p_8	p_4
10	p_4, p_8	p_1, p_2	p_3, p_4	p_9, p_{10}	p_3, p_4, p_7, p_8
11	p_1, p_2, p_3	p_1, p_4, p_7	p_6, p_9, p_{10}	p_2, p_8, p_{10}	p_1, p_8
12	p_5	p_3, p_4	p_{10}	p_7, p_9	p_1, p_8
13	p_3	p_1	p_2	p_8	p_{10}
14	p_3, p_8	p_3, p_9, p_{10}	p_1, p_2	$p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8, p_9, p_{10}$	p_6, p_8, p_9
15	p_6	p_1, p_{10}	p_1, p_4, p_7	p_6, p_8, p_9	p_1, p_4, p_7

A freqüência dos produtos utilizados no experimento (Tabela I) é ilustrada na Figura 2.

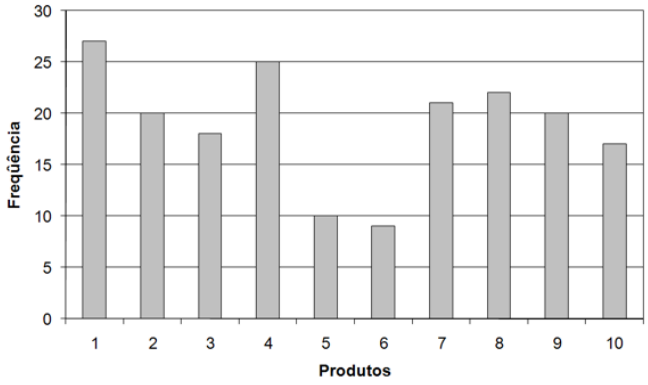


Figura 9. Freqüência dos produtos utilizados no experimento.

As fichas preenchidas pelos alunos foram coletadas e cada produto foi relacionado com todos os outros produtos da mesma compra. O número de ocorrências de cada relação de todas as compras de todas as fichas foi registrado na Tabela III. É importante observar que o valor contido na célula (p_i, p_j) corresponde ao mesmo valor da célula (p_j, p_i), pois representam a mesma relação (comprar um *livro* e uma *agenda* também corresponde a comprar uma *agenda* e um *livro*). Compras que possuem apenas um produto não constam na tabela.

Tabela III
Produtos relacionados de acordo com o experimento realizado.

	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8	p_9	p_{10}
p_1	-	10	3	7	3	1	7	8	5	5
p_2	10	-	5	6	4	2	5	5	3	2
p_3	3	5	-	7	3	1	6	4	7	3
p_4	7	6	6	-	5	3	8	7	5	5
p_5	3	4	4	5	-	1	1	0	1	3
p_6	1	2	2	3	1	-	3	2	3	3
p_7	7	5	5	8	1	3	-	9	8	3
p_8	8	5	5	7	0	2	9	-	5	3
p_9	5	3	3	5	1	3	8	5	-	4
p_{10}	5	2	2	5	3	3	3	3	4	-

O autômato adaptativo *M* do experimento foi modelado contemplando os 10 produtos apresentados anteriormente, $\Pi = \{ p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8, p_9, p_{10} \}$ e alfabeto $\sum = \Pi \cup \{link\}$. Foi assumido o valor 8 como tamanho dos caminhos para *M*.

A segunda parte do experimento consistiu na submissão de cadeias ao autômato adaptativo *M* do sistema *RecomAA*, de acordo com os dados obtidos na Tabela III. Cada cadeia possuia o formato $w = <link> \; p_i \; p_j$ e submetida *k* vezes ao autômato, onde *k* é o valor da célula (p_i, p_j). Por exemplo, a cadeia $<link> \; p_3 \; p_4$ foi submetida 7 vezes ao autômato adaptativo *M*. As cadeias no formato $<link> \; p_j \; p_i$ não foram consideradas por representarem a mesma relação que $<link> \; p_i \; p_j$.

Após a submissão de todas as cadeias no formato $w = <link> \; p_i \; p_j$ ao autômato adaptativo *M*, foram realizadas consultas sobre a recomendação de um produto em função de outro. Para tal, 45 cadeias no formato $u = p_i \; p_j$ foram submetidas ao autômato, verificando sua pertinência para *L(M)*. As cadeias no formato $p_j \; p_i$ não foram consideradas por representarem a mesma relação que $p_i \; p_j$.

A Tabela IV apresenta o resultado das submissões das cadeias no formato $u = p_i \; p_j$ ao autômato adaptativo *M*, onde:

- ✓ se $u \in L(M)$, ou
- × se $u \notin L(M)$.

Tabela IV
RESULTADO DAS SUBMISSÕES DAS CADEIAS NO FORMATO $u = p_i \; p_j$ AO AUTÔMATO ADAPTATIVO *M*.

	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8	p_9	p_{10}
p_1	-	✓	×	×	×	×	×	✓	×	×
p_2	✓	-	×	×	×	×	×	×	×	×
p_3	×	×	-	×	×	×	×	×	×	×
p_4	×	×	×	-	×	×	✓	×	×	×
p_5	×	×	×	×	-	×	×	×	×	×
p_6	×	×	×	×	×	-	×	×	×	×
p_7	×	×	×	✓	×	×	-	✓	✓	×
p_8	✓	×	×	×	×	×	✓	-	×	×
p_9	×	×	×	×	×	×	✓	×	-	×
p_{10}	×	×	×	×	×	×	×	×	×	-

A Tabela IV representa as relações existentes entre os produtos. A existência de uma relação entre p_i e p_j indica que p_j é recomendável dado p_i e vice-versa. Por exemplo, de acordo com os perfis de compra dos usuários participantes do experimento, o produto p_8 pode ser recomendado ao usuário, caso este já tenha comprado ou esteja comprando o produto p_1. Por outro lado, o produto p_8 não é recomendado, caso o usuário esteja comprando o produto p_2.

VI. IMPLEMENTAÇÃO DO SISTEMA DE RECOMENDAÇÃO UTILIZANDO AUTÔMATO ADAPTATIVO

O Sistema de Recomendação utilizando Autômato Adaptativo (*RecomAA*) foi implementado em Python. A linguagem possui algumas características que contribuíram para sua escolha, tais como velocidade de execução, flexibilidade, sintaxe simples, grande número de módulos de extensão disponíveis e integração com servidores Web.

O conjunto de recursos $\Pi = \{ p_1, \ldots, p_n \}$ é representado no *RecomAA* utilizando a linguagem de marcação XML. Cada *tag* do arquivo XML representa um recurso $p_i \in \Pi$. Um exemplo de arquivo XML pode ser visto a seguir.

```
<?xml version="1.0"?>
<recursos>
  <recurso>recurso 1</recurso>
  <recurso>recurso 2</recurso>
  ...
  <recurso>recurso n</recurso>
</recursos>
```

A partir do arquivo XML contendo o conjunto de recursos, o sistema *RecomAA* constrói o autômato adaptativo.

A configuração e execução do *RecomAA* no *shell* do Python é apresentada a seguir. O arquivo XML contendo o conjunto dos recursos foi criado de acordo com o experimento apresentado na seção anterior.

```
Python 2.5.2 (r252:60911, Feb 21 2008,
  13:11:45) [MSC v.1310 32 bit (Intel)]
  on win32
...
>>> from RecomAA import *
>>> sistema = RecomAAConf()
>>> sistema.load('recursos.xml')
True
```

Foi criado o objeto `sistema` da classe de configuração do *RecomAA* (`RecomAAConf`). O arquivo `recursos.xml` foi carregado no sistema através do método `load()` e, a partir dos dados contidos nesse arquivo, o autômato adaptativo foi criado.

```
>>> automato = sistema.getAutomaton()
True
>>> automato.getInfo()
RecomAA v1.0.10 (codename wood)
:: Automato Adaptativo criado em
:: 12/10/2008 10:14 (GMT -003)
:: Recursos disponíveis: 10
:: Tamanho do caminho: 8 estados
```

O objeto `automato` é uma instância da classe que implementa o autômato adaptativo do *RecomAA* (`AAutomaton`). O método `getInfo()` retorna as informações sobre o objeto, como data de criação, recursos contemplados pelo autômato e o tamanho do caminho.

A submissão de cadeias ao autômato adaptativo `automato` pode ser realizada através do método `query(...)`. O método retorna `true` se a cadeia pertence ao autômato, ou `false` caso contrário. A submissão de uma cadeia que denota uma consulta sobre a relação entre os produtos p_1 e p_8 é ilustrada a seguir.

```
>>> automato.query('p1','p8')
False
```

O método `query('p1','p8')` retornou `False`, o que denota que os produtos p_1 e p_8 não possuem relação entre si.

A próxima etapa da execução consistiu na submissão de cadeias no formato $w = <link> \; p_i \; p_j \; k$ vezes ao autômato, onde

k é o valor da célula (p_i, p_j) da Tabela III. Essa etapa é igual à segunda parte do experimento apresentado na seção anterior.

```
>>> automato.query('link','p1','p2')
True
>>> automato.query('link','p1','p2')
True
...
>>> automato.query('link','p9','p10')
True
```

Após as submissões das cadeias no formato w = <*link*> p_i p_j, a submissão de uma cadeia no formato $u = p_i\ p_j$ deverá coincidir com os resultados obtidos na Tabela IV. Por exemplo, nesse momento a submissão da cadeia $p_1\ p_8$ retornará `True`, de acordo com a Tabela IV.

```
>>> automato.query('p1','p8')
True
```

A implementação do sistema *RecomAA* também permite sua utilização em páginas Web, disponível através do módulo `RecomAAWebInterface`, também escrito em Python. A Figura 3 apresenta a interface Web do *RecomAA*.

Figura 10. Interface Web do *RecomAA*.

A interface Web disponibiliza um campo para consulta que possibilita verificar a existência de alguma relação entre dois recursos. A Figura 4 ilustra a resposta de uma consulta.

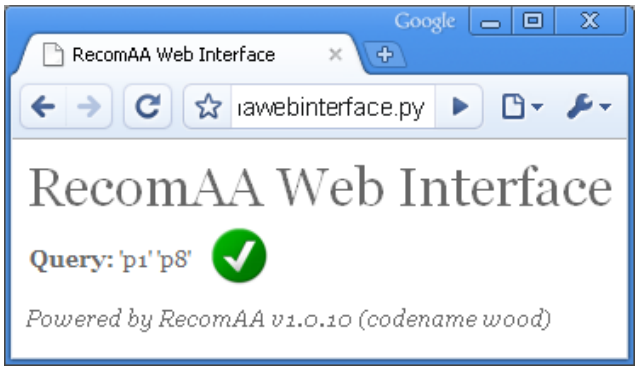


Figura 11. Resposta de uma consulta através da Interface Web do *RecomAA*.

O módulo `RecomAAWebInterface` é um script Python CGI que pode ser executado em um servidor Web, através do endereço:

```
/cgi-bin/recomaawebinterface.py
```

A implementação do *RecomAA* tem como objetivo fornecer um conjunto de classes que realizam uma recomendação simples de recursos, utilizando um autômato adaptativo, para os sistemas existentes.

VII. Conclusões

Este artigo apresentou um sistema de recomendação utilizando um autômato adaptativo para verificar as relações entre recursos. O autômato adaptativo, devido à característica de auto-modificação, é uma escolha interessante para auxiliar na avaliação de quais recursos são mais importantes para um determinado usuário.

A recomendação de recursos é viável do ponto de vista comercial, pois permite que um ambiente torne-se "mais próximo" do usuário, conhecendo seus interesses e hábitos de navegação, e sugerindo recursos que o agradem. A utilização da tecnologia adaptativa na área de personalização de serviços e recursos proporciona soluções consistentes que atendem às necessidades inerentes a esse domínio de aplicação.

Referências

[1] R. E. Grande, "Sistema de integração de técnicas de proteção de privacidade que permitem personalização", Dissertação de Mestrado, Departamento de Computação, Universidade Federal de São Carlos, 2006.

[2] M. Koch and K. Moeslein, "User representation in ecommerce and collaboration applications". In *BLED 2003 Proccedings*, 2003.

[3] R. Kimball and R. Merz, *Data Webhouse: construindo o data warehouse para a Web*. Rio de Janeiro: Campus, 2000.

[4] S. D. Zorzo, R. A. Gotardo, P. R. M. Cereda, B. Y. L. Kimura, R. A. Rios and R. E. Grande, "Web privacy controlled by user: an approach to treat the user's preferences about personal data". In *European Computing Conference 2007*, 2007.

[5] D. Kristol and L. Montulli, "HTTP state management mechanism", Bell Laboratories, Lucent Technologies, RFC 2965, 2000.

[6] A. L. Montgomery, "Using clickstream data to predict www usage", University of Maryland, Tech. Report, 2003.

[7] A. L. B. Nogueira and L. R. Oliveira Jr., *Uma análise da aplicabilidade de Data Warehouse em ambientes empresariais*, Faculdade Ruy Barbosa, Salvador, 2004.

[8] S. Azambuja, "Estudo e implementação da análise de agrupamento em ambientes virtuais de aprendizagem", Dissertação de Mestrado, Universidade Federal do Rio de Janeiro, 2005.

[9] W. Aiello and P. McDaniel, *Lecture 1, Intro: Privacy*, Stern School of Business, NYU, 2004.

[10] M. P. Silva, C. Boscarioli and S. M. Peres, "Análise de logs da web por meio de técnicas de data mining". In *I Congresso de Tecnologias para Gestão de Dados e Metadados do Cone Sul*, Universidade Estadual de Ponta Grossa, 2003.

[11] R. A. Gotardo, C. A. C. Teixeira and S. D. Zorzo, "Predicting user's interests in web-based educational systems using a collaborative filtering weighted method". In *11th IEEE International Conference on Computational Science and Engineering – Workshop on Computational Science and Engineering – CSE 2008*, 2008.

[12] S. Sae-Tang and V. Esichaikul, "Web personalization techniques for e-commerce". In *Active Media Technology: 6th International Computer Science Conference, 2005*.

[13] E. B. Reategui and S. C. Cazella, "Sistemas de recomendação". In *XXV Congresso da Sociedade Brasileira de Computação – A universalidade da computação: um agente de inovação e conhecimento*, 2005.

[14] J. J. Neto, "Contribuições à metodologia de construção de compiladores", Tese de Livre Docência, Escola Politécnica da Universidade de São Paulo, São Paulo, 1993.
[15] _____, "Adaptive automata for context-dependent languages", *SIGPLAN Notices*, vol. 29, no. 9, PP. 115-124, 1994.
[16] P. R. M. Cereda, "Modelo de controle de acesso adaptativo", Dissertação de Mestrado, Departamento de Computação, Universidade Federal de São Carlos, 2008.
[17] R. L. A. Rocha and J. J. Neto, "Autômato adaptativo, limites e complexidade em comparação com máquina de Turing". In *Proceedings of the Second Congress of Logic Applied to Technology – LAPTEC*. São Paulo, Brasil: Faculdade SENAC de Ciências Exatas e Tecnologia, 2001.

Paulo Roberto Massa Cereda obteve o mestrado em Ciência da Computação pela Universidade Federal de São Carlos em 2008. Atualmente é aluno pesquisador do Grupo de Sistemas Distribuídos e Redes (GSDR) do Departamento de Computação da Universidade Federal de São Carlos. Tem experiência na área de Ciência da Computação, atuando nas áreas de Privacidade e Personalização de Serviços, Teoria da Computação e Tecnologias Adaptativas.

Reginaldo Aparecido Gotardo obteve o mestrado em Ciência da Computação pela Universidade Federal de São Carlos em 2008. Atualmente é professor das Faculdades COC. Tem experiência na área de Ciência da Computação, com ênfase em Linguagens de Programação, atuando principalmente nos seguintes temas: Computação em Grade, GT4, Globus, Web Services e Privacidade e Personalização de Serviços.

Sérgio Donizetti Zorzo obteve o doutorado em Engenharia Elétrica pela Universidade de São Paulo em 1996. Atualmente é professor associado do Departamento de Computação da Universidade Federal de São Carlos. Tem experiência na área de Ciência da Computação, com ênfase em Teleinformática, atuando principalmente em Qualidade, Privacidade e Personalização de Serviços, Multimídia, Computação em Ambientes sem Fio e Tecnologias Adaptativas.

Estudio de la mejora de la calidad de voz para un sintetizador en idioma castellano usando el método de Autómatas Adaptativos (5 Enero 2009)

R. Caya y C. Zapata, *Member IEEE*

***Resumen*— El proyecto en desarrollo, presentado en este artículo, realiza un estudio de la mejora de la calidad de voz del motor de síntesis de voz *FESTIVAL* mediante la adición de autómatas adaptativos. Dichos autómatas están diseñados para dar solución a algunos problemas fonológicos identificados para el castellano, y se basan en reglas lingüísticas dependientes del contexto las cuales también son materia de investigación del proyecto. El sistema modificado será sometido a pruebas formales de calidad de voz como MOS y frases psicoacústicas de Harvard adaptadas al castellano, para evaluar su mejora.**

Palabras clave—Síntesis del habla, Interfaces de lenguaje Natural, Autómata Adaptativo.

I. NOMENCLATURA

- AA: Autómata adaptativo.
- TTS: Texto to speech, en español texto a voz.
- Elementos supra-segmentales: estructuras mayores a la palabra léxica propias del discurso oral.
- Inteligibilidad: la facilidad para comprender la señal de voz producida.
- Naturalidad: un indicador de la semejanza de los sonidos producidos artificialmente con los naturales.
- PLN: procesamiento de lenguaje natural
- Coarticulación: fenómeno lingüístico mediante el cual un fonema se ve influenciado por su contexto.
- Alofonemas: variaciones de un fonema en un idioma.
- Utterance: unidad del discurso ubicada entre dos pausas evidentes denotadas por signos de puntuación

II. INTRODUCCIÓN

ACTUALMENTE los avances en el campo de la tecnología del habla nos permiten encontrar sistemas de síntesis de voz muy avanzados en diversos idiomas. Sin embargo éstos aún presentan problemas que impiden el logro de la naturalidad deseada por los usuarios, entendida, de acuerdo con [9], como la similitud lograda entre la voz artificial generada y el discurso humano, respecto a las características de un idioma determinado.

El presente proyecto tiene como objetivo identificar y formular las reglas dependientes del contexto que puedan dar solución a algunos de los problemas encontrados en un sintetizador de voz en particular, y que permitan la mejora en la naturalidad de la voz sintética producida utilizando técnicas adaptativas. Su realización se encuentra justificada por el impacto que significaría para la sociedad actual contar con una mejor calidad de la síntesis de voz.

III. MARCO CONCEPTUAL

La búsqueda de un medio que permita una interacción más natural con el ordenador, tal como menciona [1], constituye una de las principales razones para la investigación en la rama denominada HCI por sus siglas en inglés (Human-Computer interaction). En este contexto, la síntesis de voz parece ser un gran aliado, debido a que los sistemas de síntesis de voz permiten enriquecer la interacción humano-computador desde el punto de vista de la respuesta del ordenador.

Los sistemas de síntesis son comúnmente denominados TTS por sus siglas en inglés (Text-to-speech), sin embargo su información de entrada puede ser tanto texto como una representación lingüística simbólica. Según la flexibilidad de la cadena de entrada que reciben, se pueden clasificar, como indica [11], en: sistemas de respuesta de voz y conversores de texto a voz. A estos últimos corresponde el sistema de síntesis con el que se trabaja en este proyecto, pues presentan mayores problemas en la calidad de voz al permitir trabajar con cualquier texto de entrada.

Todo sintetizador implementa un método de síntesis. En la actualidad los principales métodos utilizados son: síntesis concatenativa y síntesis por formantes. Tal como señala [17], el primer tipo se refiere a la capacidad del sintetizador de producir un discurso fluido, en tiempo real, en base a la consecución de unidades del habla, pregrabadas y etiquetadas en una base de datos. En este tipo de síntesis la selección del tamaño de las unidades a almacenar presenta una complejidad alta, con influencia en la calidad de voz obtenida, como se menciona en [14]. La síntesis por formantes, en cambio, propone la generación de la voz sintética a partir de un modelo acústico de resonadores electrónicos que simulen las características físicas del aparato fonológico humano [17].

Con respecto a los temas de investigación en el área, la mayoría busca integrar al proceso de síntesis el concepto de prosodia, con la finalidad de analizar y representar

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

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

R. Caya es alumna de pregrado en la Especialidad de Ingeniería Informática de la Facultad de Ciencias e Ingeniería de la Pontificia Universidad Católica del Perú, Av. Universitaria 1801, San Miguel, Lima 32, Perú (correo e.: rosalia.caya@pucp.edu.pe).

formalmente las características de los elementos supra-segmentales en la expresión oral. De acuerdo con [8], es la prosodia la encargada de dotar al discurso de expresividad y actitud que permiten que el receptor comprenda el mensaje transmitido.

IV. Descripción de la solución

En este proyecto se propone el estudio de la mejora de la calidad de un sintetizador de voz para el castellano, utilizando el método de AA, para la representación de relaciones gramaticales dependientes del contexto que afecten la calidad de la voz sintética. Dicha calidad puede ser evaluada desde dos puntos: inteligibilidad y naturalidad. Sin embargo, es importante resaltar que no se ha implementado un nuevo sintetizador de voz, sino que se ha modificado uno existente.

El sintetizador con el cual se trabaja es Festival Speech Synthesis System, una plataforma de software libre dedicada por completo a la síntesis, creada por el Centro de Investigación de Tecnologías del Lenguaje de la Universidad de Edimburgo [4]. Su elección corresponde a que se trata de un software libre, cuenta con documentación oficial, presenta flexibilidad en el trabajo con varios lenguajes de programación y ha sido utilizado anteriormente por otros trabajos de investigación con resultados satisfactorios. Debido a estas razones su uso académico es bastante viable.

El proyecto se viene realizando en dos fases: investigación de las reglas fonológicas del español y desarrollo del aporte producto de la investigación. Los objetivos establecidos para la primera fase son: identificar los factores que impiden la producción de una voz sintética natural y formular las reglas dependientes del contexto usando autómatas adaptativos que solucionen algunos de los factores identificados en el punto anterior. La fase de desarrollo del aporte tiene como objetivos: implementar los autómatas adaptativos en la plataforma de síntesis elegida; diseñar y realizar experimentos con los autómatas adaptativos implementados en casos reales de síntesis de voz para el castellano. Actualmente el proyecto está finalizando la primera fase.

A. Sustento de la solución

La síntesis de voz es realizada en FESTIVAL, por medio de la aplicación de reglas que buscan modelar sus características. Así mismo, varias de estas reglas presentan dependencias del contexto propias del idioma que modelan. Bajo estas consideraciones se eligió trabajar con un AA debido a que su principal objetivo es el análisis del contexto por medio de la formulación de reglas que le permitan adaptarse al mismo. Por lo tanto, los AA encuentran un escenario idóneo en el análisis de las características supra-segmentales del castellano.

Por otra parte, la formación de palabras fonológicas y el análisis de los modelos melódicos de puntuación, constituyen en castellano conceptos que modelan características supra-segmentales que enriquecen la expresión oral, como son: las pausas, la melodía y el tono. Así mismo, el análisis para ambos conceptos es viable dentro del alcance del presente proyecto al encontrarse normado su uso en la gramática española como se demuestra en [12].

Como se menciona, actualmente se emplean diversas técnicas en los sintetizadores de voz y hasta el momento la calidad de la síntesis en términos de naturalidad va asociada con el grado de complejidad del método usado para llevarla a cabo. Algunos de estos métodos son: Modelos ocultos de Markov, redes neuronales, TD-PSOLA, MEL-CEPSTRAL, entre otros. El inconveniente al utilizar estos métodos es que el manejar los problemas dependientes del contexto aumenta rápidamente la complejidad de la solución, restringiendo los proyecto a un ámbito reducido del dominio del problema para poder acceder a una calidad de voz aceptable. En estos casos no solo se tiene que tratar con la complejidad teórica del algoritmo sino que a ello se suman la propia complejidad del problema y la adaptación del algoritmo para solucionarlo. Tal como menciona [11], actualmente los sistemas de síntesis del habla demandan la maximización de la calidad de voz, mientras minimizan su requerimiento de espacio en memoria y la complejidad del algoritmo que manejan.

Un AA, tal como se menciona en [3], provee un modo claro y simple de manejar sintácticamente las características dependientes del contexto. Esta dependencia se maneja mediante modificaciones en la estructura del propio AA y que son llevadas a cabo por reglas que se disparan al detectarse una condición predefinida en el contexto de ejecución del autómata.

Es relevante, en este punto, señalar que las aplicaciones actuales de síntesis de voz, como FESTIVAL, utilizan reglas para el manejo de las condiciones de dependencia del contexto en la síntesis de voz, por lo que el usar AA se presenta como una alternativa muy natural, pero con una ventaja: la capacidad representativa de los AA, esto es, se espera que sea más sencillo de representar e implementar reglas que manejen condiciones de dependencia del contexto usando AA que con los métodos actuales.

B. Resultados esperados

El aporte práctico del proyecto es la implementación de una mejora de la naturalidad en la voz sintética producida por FESTIVAL para el castellano, lograda mediante AA.

El resultado esperado general es: La mejora de la calidad de voz con AA disminuyendo la complejidad de la solución requerida hasta ahora en para la solución de problemas de la síntesis de voz en relación con la naturalidad y la prosodia.

En relación con los objetivos específicos del proyecto los resultados esperados son:

1. La documentación de los factores identificados que impiden la producción de voz sintética natural y su orden de importancia.
2. El diseño de AA que permitan dar solución a algunos de los problemas antes mencionados para el castellano.
3. Los resultados de la evaluación realizada al sintetizador modificado con AA, respecto a la naturalidad resultante.

V. Apoyo teórico

Diversos modelos de autómatas han sido aplicados exitosamente en diversas ramas del PLN para afrontar múltiples problemas. Así la teoría de autómatas provee

herramientas eficientes y convenientes para la representación del fenómeno lingüístico y encuentra en el PLN su campo de mayor aplicación [1]. En los TTS, la teoría de autómatas ya ha sido empleada para las tareas de transformación del texto de entrada en unidades manejables por el sistema.

En consecuencia, tanto la teoría sobre autómatas como las aplicaciones realizadas hasta el momento sostienen que existe un subconjunto de reglas de la gramática del lenguaje natural que pueden ser representadas por medio de autómatas.

Los trabajos realizados hasta el momento respecto a TTS, señalan que una posible mejora en la calidad de voz se lograría al analizar y representar las características contextuales que presenta el texto de entrada del sistema. De este modo, fenómenos naturales como la coarticulación y la selección de alofonemas en el habla puedan ser simulados. Actualmente los TTS más avanzados han realizado mejoras en su calidad adicionando modelos estadísticos altamente complejos para analizar las características prosódicas inmersas a nivel léxico. Los resultados han sido diversos, presentando un mejora significativa en lenguas en las que algunas características prosódicas poseen representación léxica, como en el caso del francés y el castellano. Para el caso del castellano la prosodia es parcialmente representada en grafemas por medio de signos de puntuación, interrogación, y admiración, además del uso de la tilde para denotar la fuerza de voz en alguna sílaba y, en general, convenciones ortográficas y gramaticales que permiten aplicar diversas características al texto que indiquen cuál debe ser su representación oral.

El contar con una representación prosódica parcial en el nivel tipográfico del castellano permite identificar, de modo evidente, la relación entre sintaxis y prosodia. Dicha relación se presenta por medio de construcciones gramaticales, que en el campo lingüístico son reconocidas desde hace ya varios años.

Los autómatas adaptativos son aplicados en la actualidad a diversos campos. Tal como se menciona en [3], la tecnología adaptativa es aplicable, en general, en aquellos entornos en los cuáles se presente dependencia del contexto. La aplicación de dicha tecnología para algunas tareas en los procesos de síntesis de voz no sólo es posible sino que además es recomendada en [18], desde el punto de vista de un análisis del contexto para lograr una mejor representación del discurso humano tal como se postula en [16].

Los autómatas adaptativos presentan además la posibilidad de trabajar desde el nivel sintáctico, con una complejidad de solución mucho menor respecto a las técnicas utilizadas hasta el momento.

En resumen, la teoría de AA nos permite representar algunas características del discurso del castellano por medio de una gramática finita, clara y exacta, en especial cuando las reglas que constituyen dicha gramática presentan una fuerte dependencia del contexto en el que se encuentran.

VI. Métodología aplicada

El proyecto de investigación se encuentra guiado por la metodología propuesta en [4] para proyectos de investigación. La adaptación de las líneas generales descritas en [4] para este proyecto dieron como resultado las siguientes etapas:

1. Identificación de los parámetros presentes en la sintaxis que afectan la naturalidad del discurso.
2. Identificación de casos en los cuales se encuentra presente la dependencia del contexto.
3. Identificación de las reglas que solucionan los casos seleccionados.
4. Construcción de los autómatas adaptativos que implementan dichas reglas.
5. Selección y realización de pruebas que evalúen la calidad del sintetizador.
6. Análisis de resultados.
7. Elaboración de conclusiones.

VII. Diseño e implementación de los Autómatas Adaptativos

Para este proyecto se diseñarán e implementarán dos AA, cada uno con una función específica. El primero de ellos se dedicará a la formación de palabras fonológicas a partir del texto ingresado. El segundo se concentrará en la asignación de pausas y patrones melódicos característicos de los signos de puntuación en el castellano. Cada uno de ellos se detalla a continuación.

A. Autómata Adaptativo para la formación de palabras fonológicas

El autómata adaptativo que se diseña a continuación tiene como función el reconocimiento y la formación de las palabras fonológicas para el texto de entrada del sintetizador. La palabra fonológica es un concepto clave en la prosodia, la acentuación y la entonación dentro de la gramática española. Su inclusión dentro del procesamiento del texto en un sintetizador de voz permitirá elevar el análisis realizado al nivel de frase dentro de la jerarquía fónica.

La formación de la palabra fonológica en el castellano sigue las siguientes reglas [10]:

1. La concatenación se realiza con palabras léxicas completas, por lo tanto una palabra fónica contendrá por lo menos una palabra léxica.

2. Se concatenan las palabras átonas presentes, desde el inicio del texto a analizar, con la primera palabra que contenga la sílaba tónica, junto con la cual constituirán la palabra fónica.

Para el siguiente ejemplo extraído de [10]:

"El tigre se sube en el árbol"

Según la regla 1 y 2 su representación sería:

"Eltigre sesube enelárbol"

Donde la primera palabra fónica está formada por las palabras: El + tigre; la segunda, por: se + sube, y la tercera por: en + el + árbol.

Siguiendo la notación formal descrita en [18], el autómata adaptativo encargado de la construcción de palabras fonológicas, nombrado como M_F, se define formalmente como una 8-tupla:

M =(Q, $\sum$, q_0, F, δ, Q,Γ,π), donde:

Q: {a_0, a_f}

$\sum$: {átona, tónica, ε}

q_0 : {a_0}

F : { a_f }

δ:

- $P_0 : a_0$, átona$\rightarrow a_0$
- $P_1 : a_0, \varepsilon \rightarrow a_f$

Q : {a1, a2, a3,...}
Γ: (-, a, ε, a_f)
(+, a, ε, a')
(+, a', átona, a')
(+, a', tónica, a'')
(+,a'', ε, a_f)
π : ((a_0, átona, a_0)) = {(-, a, ε, a_f), (+, a, ε, a'),(+, a', átona, a'), (+, a', tónica, a''), (+,a'', ε, a_f)}, para todo a, a_f que existen y para cada a' y a'' que no existe en Q.

La cadena inicial del AA en su estado inicial está denotada por ω, y corresponde a la lista de palabras léxicas, enmarcadas por los símbolos separadores indicados para el idioma (en FESTIVAL son el espacio en blanco y los símbolos de puntuación presentes en cada idioma). Dicha lista es extraíble, dentro de la arquitectura de FESTIVAL, desde la relación WORD que posee la unidad del discurso (utterance) construida desde el texto proporcionado por el usuario. Por lo tanto, tenemos:

$$\omega = \text{utt.WORD}$$

Además, en Festival se tiene acceso a las listas de palabras átonas del castellano previamente identificadas y definidas en una lista de nombre "atona", con las cuáles se trabajará en el autómata.

La operación de M_F ocurre de la siguiente forma: se parte de la configuración inicial del autómata descrita en la Fig. 1

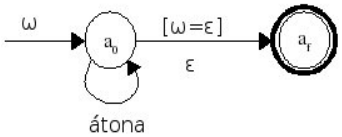


Fig. 1. Configuración Inicial del AA M_F

Luego, al recibir la cadena de entrada dicha configuración cambia a la señalada en la Fig. 2 en la cual se leen n palabras átonas antes de hallar una palabra tónica, y cada vez que se identifica dicha palabra se analiza si ya se acabó de procesar ω. Si fue así, se acaba el proceso. Si por otro lado ocurrió $\omega \neq \varepsilon$, entonces el autómata sustituirá la única transición condicionada en vacío presente por una secuencia de cuatro transiciones: La primera en vacío, la segunda consumiendo x palabras átonas, la tercera consumiendo una palabra tónica y una última transición condicionada hacia el estado final. Esta última transición garantiza la existencia de una transición condicionada en cada paso de la operación del autómata.

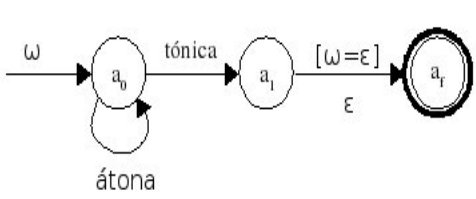


Fig. 2. Configuración del AA M_F luego de realizar la primera transición adaptativa

Luego de i transiciones adaptativas la configuración del autómata será como en la Fig. 3.

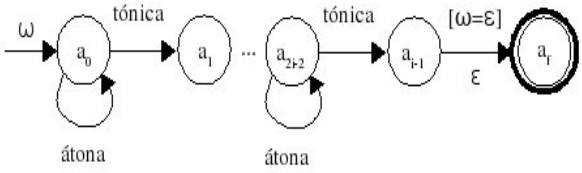


Fig. 3. Configuración del AA M_F luego de i transiciones adaptativas

De esta manera M_F se dedicará así a construir palabras fonológicas que presenten la forma correspondiente con su definición en castellano: [átona ^{n} tónica] ^{k}

B. Autómata Adaptativo para análisis de los signos de puntuación

Para el análisis de las pausas y entonaciones característicos de los signos de puntuación en el castellano se realizarán un conjunto de autómatas adaptativos, cada uno de los cuales analizará el caso particular de la utilización de un signo, y finalmente trabajarán en conjunto de acuerdo a la estructura jerárquica fónica de frases del idioma [6]. Los signos de puntuación que manifiestan pausas y entonación melódica son: "."(punto), ";"(punto y coma), ","(coma), "…"(puntos suspensivos) y ":"(dos puntos); sin embargo también se analizarán aquellos signos que se manifiestan sólo como cambios en el patrón melódico: "¡!"(signos de admiración), "¿?"(signos de interrogación) y "()"(paréntesis). Los autómatas encargados del análisis serán definidos individualmente para luego ser unificados al nivel de la oración.

El mecanismo encargado del análisis de la pausa y entonación características del punto, nombrado M_p, se define formalmente según [18]:
M =(Q, ∑, q_o, F, δ, Q,Γ,π), donde:
Q: {a_0, a_f}
∑: {l, [.], [ ], ε}
q_0 : {a_0}
F : { a_f }
δ:

[6] $P_0 : a_0, l \rightarrow a_0$
[7] $P_1 : a_0, \varepsilon \rightarrow af$
[8] $P_2 : a_0, [.] \rightarrow af$

Q : {a1, a2, a3,...}
Γ: (-, a, ε, a_f)
(+, a, [.], a')
(+, a', [ ], a'')
(+,a'', ε, a''')
(+,a''', l, a''')
(+,a''', [.], a_f)
π : ((a_i, [.], a_f)) = {(-, a, ε, a_f), (+, a, [.], a'), (+, a', [ ], a''), (+,a'', ε, a'''), (+,a''', l, a'''), (+,a''', [.], a_f)}, para todo a, a_f que existen y para cada a', a'', a''' que no existe en Q.

Donde l es cualquier símbolo diferente a un signo de puntuación: letras, números, o caracteres especiales. Así mismo es importante notar en adelante el reconocimiento del espacio en blanco que sigue a un signo de puntuación en el castellano, normado por la Real Academia Española en [12], con lo cual se elimina el problema de los separadores numéricos, como se menciona formalmente en [12] para el acápite de números.

La cadena de entrada de M_p está representada por ω y corresponde a la lista de caracteres ingresados para la síntesis, se analiza a este nivel debido a que Festival trabaja en los demás niveles sin signos de puntuación. Dicha lista es extraíble dentro de la arquitectura de FESTIVAL desde la relación TEXT que posee el utterance:

ω=utt.TEXT

Además se tiene acceso a la lista de signos de putuación del castellano previamente identificados y definidos en una lista de nombre punc, con las cuáles se trabajará en el autómata.

El funcionamiento de Mp es similar al autómata de formación de la palabra fonológica, pero en este caso además de reconocer el caracter punto [.], el segundo estado creado se encarga del reconocimiento del espacio en blanco que sigue al signo de puntuación, y de establecer una pausa normal(#) dentro de la estructura del utterance. Se establece esta pausa debido a que cada punto reconocido dentro del utterance corresponderá obligatoriamente a un punto seguido, a excepción del último el cual será un punto final o punto a parte(los cuales son fonológicamente idénticos), por ello en el estado af se establecerá una pausa larga (##) correspondiente a este tipo de puntuación.

Adicionalmente con cada pausa establecida se asignará el patrón melódico de acuerdo a las normas del castellano, para el caso del punto una inflexión descendente en cadencia absoluta(⇓),

La configuración inicial de Mp y su cambio con cada transición adaptativa se representa en la Fig. 4.

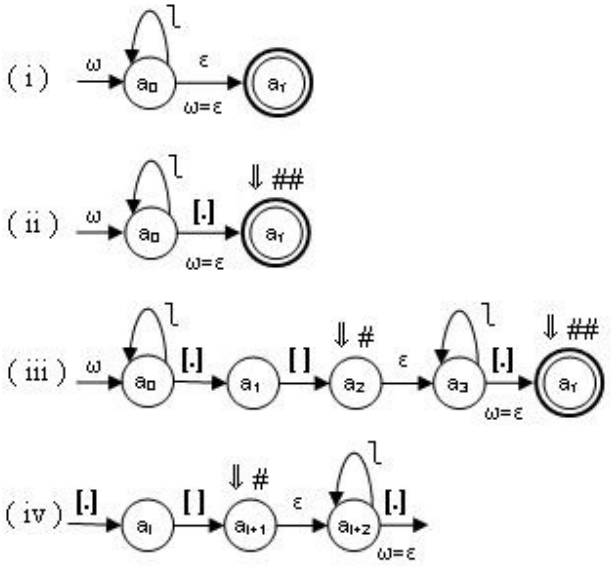


Fig.3. (i) Configuración del AA Mp. (ii) Mp luego de realizar la primera transición adaptativa. (iii) Mp luego de una segundo transición adaptativa. (iv) Cambios en Mp luego de cada transición adaptativa.

Para el caso de la coma, el mecanismo encargado de su análisis, nombrado Mc, se define formalmente:
M =(Q, ∑, q_0, F, δ, Q,Γ,π), donde:
Q: {b_0, b_f}
∑: {l, [,], [ ], ε}
q_0 : {b_0}
F : { b_f }
δ:

- P_0 : b_0, l → b_0
- P_1 : b_0, ε → bf
- P_2 : b_0, [,] → bf

Q : {b1, b2, b3,...}
Γ: (-, b, ε, b_f)
(+, b, [,], b')
(+, b', [ ], b'')
(+,b'', ε, b''')
(+,b''', l, b''')
(+,b''', ε, b_f)
π : ((b_i, [.], b_f)) = {(-, b, ε, b_f), (+, b, [,], b'), (+, b', [ ], b''), (+,b'', ε, b'''), (+,b''', l, b'''), (+,b''', ε, b_f)}, para todo b, b_f que existen y para cada b', b'', b''' que no existe en Q.

Este AA permite el reconocimiento de comas en todas las formas permitidas gramaticalmente: unitaria (como separación entre oraciones), vocativos y enumeraciones de acuerdo con [4].

Tal como en el autómata anterior, Mc reconoce las comas, asigna una pausa mínima(#) y establece la entonación correspondiente, en este caso una inflexión descendente (↓). Sin embargo este autómata no trabajará de modo aislado sino que se creará y obtendrá su entrada a partir del reconocimiento de la coma por M_p en alguno de sus estados de lectura, así su estado final retornará el control al estado correspondiente de M_p La descripción de su funcionamiento se presenta en la Fig. 5.

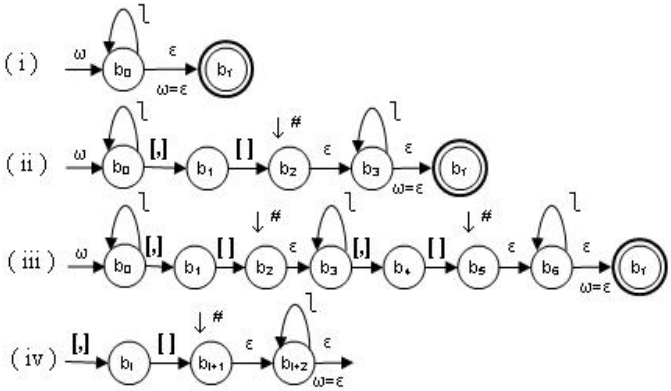


Fig.5. (i) Configuración del AA Mc. (ii) Mc luego de realizar la primera transición adaptativa. (iii) Mc luego de una segundo transición adaptativa. (iv) Cambios en Mc luego de cada transición adaptativa.

En el caso del punto y coma, el mecanismo encargado de su análisis, nombrado M_{pc}, se define formalmente:
M =(Q, ∑, q_0, F, δ, Q,Γ,π), donde:
Q: {c_0, c_f}
∑: {l, [;], [ ], ε}
q_0 : {c_0}
F : { c_f }
δ:

- P_0 : c_0, l → c_0
- P_1 : c_0, ε → cf
- P_2 : c_0, [;] → cf

Q : {c1, c2, c3,...}

Γ: (-, c, ε, c_f)
(+, c, [;], c')
(+, c', [ ], c'')
(+,c'', ε, c''')
(+,c''', l, c''')
(+,c''', ε, c_f)
π : ((c_i, [;], c_f)) = {(-, c, ε, c_f), (+, c, [;], c'), (+, c', [ ], c''), (+,c'', ε, c'''), (+,c''', l, c'''), (+,c''', ε, c_f)}, para todo c, c_f que existen y para cada c', c'', c''' que no existe en Q.

Según [10], el punto y coma indica una pausa superior a la marcada por la coma e inferior a la señalada por el punto, para el caso de este proyecto representada por (#). Así mismo, el punto y coma es usado para separar, como el punto enunciados completos y siempre representa una inflexión melódica descendente (↓), sin llegar a la cadencia con la cual se representa al punto.

Al igual que en el caso de la coma, M_{pc} trabajará en coordinación con M_p, conservando el análisis en el nivel de oración.

La representación del funcionamiento de M_{pc} se prensenta en la Fig.6.

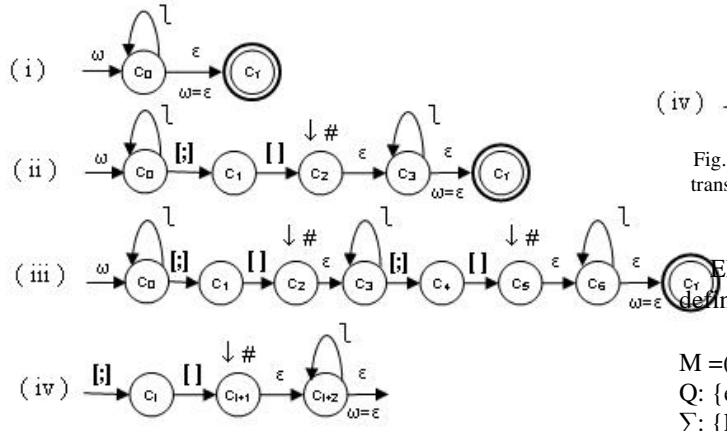


Fig. 6. (i) Configuración del AA Mpc. (ii) Mpc luego de realizar la primera transición adaptativa. (iii) Mpc luego de una segundo transición adaptativa. (iv) Cambios en Mpc luego de cada transición adaptativa.

Para el caso de los dos puntos, el mecanismo encargado de su análisis, nombrado M_{dp}, se define formalmente:

M =(Q, ∑, q_0, F, δ, Q,Γ,π), donde:
Q: {d_0, d_f}
∑: {l, [:], [ ], ε}
q_0 : {d_0}
F : { d_f }
δ:

- P_0 : d_0, l → d_0
- P_1 : d_0, ε → df
- P_2 : d_0, [:] → df

Q : {d1, d2, d3,...}
Γ: (-, d, ε, d_f)
(+, d, [:], d')
(+, d', [ ], d'')
(+, d'', ε, d''')
(+, d''', l, d''')
(+, d''', ε, d_f)
π : ((d_i, [:], d_f)) = {(-, d, ε, d_f), (+, d, [:], d'), (+, d', [ ], d''), (+, d'', ε, d'''), (+,d''', l, d'''), (+,d''', ε, d_f)}, para todo d, d_f que existen y para cada d', d'', d''' que no existe en Q.

Los dos puntos detienen el discurso para llamar la atención sobre lo que sigue, debido a ello en [10] y [4] se los representa con una inflexión descendente (↓), acompañada de una pausa pequeña ($^{\#\#}$), que puede ser similar a la correspondiente a una coma.

Al igual que en los casos anteriores éste autómata trabaja en coordinación con M_p, conservando el análisis en el nivel de oración.

En la Fig. 7 se presenta su configuración inicial y estados adaptativos.

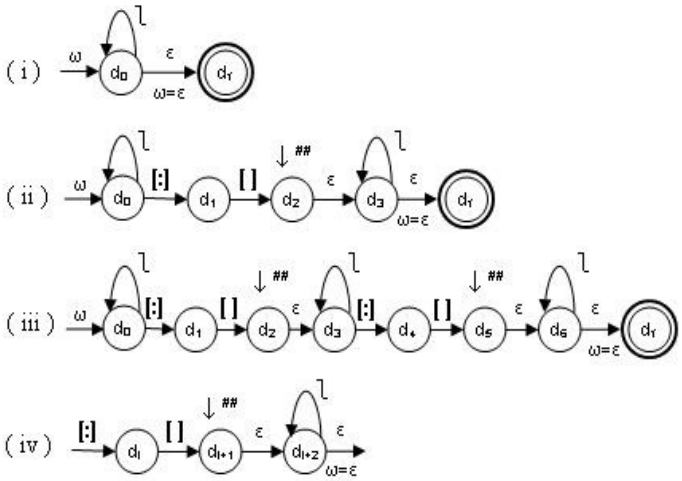


Fig. 7. (i) Configuración del AA Mpc. (ii) Mpc luego de realizar la primera transición adaptativa. (iii) Mpc luego de una segundo transición adaptativa. (iv) Cambios en Mpc luego de cada transición adaptativa.

El caso de los puntos suspensivos se nombra M_{ps} y se define formalmente como sigue:

M =(Q, ∑, q_0, F, δ, Q,Γ,π), donde:
Q: {e_0, e_f}
∑: {l, […], [ ], ε}
q_0 : {e_0}
F : { e_f }
δ:

- P_0 : e_0, l → e_0
- P_1 : e_0, ε → ef
- P_2 : e_0, [:] → ef

Q : {e1, e2, e3,...}
Γ: (-, e, ε, e_f)
(+, e, [:], e')
(+, e', [ ], e'')
(+, e'', ε, e''')
(+, e''', l, e''')
(+, e''', ε, e_f)
π : ((e_i, […], e_f)) = {(-, e, ε, e_f), (+, e, […], e'), (+, e', [ ], e''), (+, e'', ε, e'''), (+,e''', l, e'''), (+,e''', ε, e_f)}, para todo e, e_f que existen y para cada e', e'', e''' que no existe en Q.

Los puntos suspensivos tienen usos bastante estandarizados y se representan siempre como una inflexión ascendente en anticadencia (⇑), típica del final de una frase. Así mismo, suponen una interrupción de la oración o un final impreciso, lo cual se manifiesta a través de una pausa normal (#), según lo indica Navarro en [10].

Una representación de su funcionamiento se muestra en la Fig. 8.

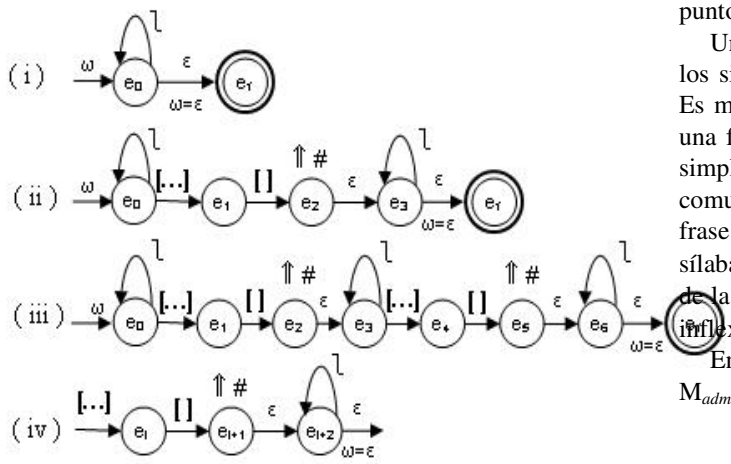


Fig. 8. (i) Configuración del AA Mps. (ii) Mps luego de realizar la primera transición adaptativa. (iii) Mps luego de una segundo transición adaptativa. (iv) Cambios en Mps luego de cada transición adaptativa.

En los casos de signos compuestos por apertura y clausura, tales como los signos de interrogación(¿ ?), admiración(!¡) y paréntesis(), se realizará un autómata por cada uno para garantizar las reglas de apertura y cierre características de los mismos.

A diferencia de los autómatas anteriores, el análisis de signos de interrogación, admiración y paréntesis debe tener en cuenta que estos signos encierran oraciones o frases con sentido completo y pueden contener otros signos de puntuación. Por lo tanto es necesario incluir un análisis a nivel de oración dentro de estos autómatas, agregando las características melódicas correspondientes de cada caso.

Para los signos de interrogación, el autómata M_{int} inicia reconociendo el signo de apertura (¿) y condicionando su llegada a un estado final con la lectura del signo de clausura (?). Luego un "estado" intermedio se encargará del análisis de la oración contenida entre los signos, este "estado" es en realidad una instancia del AA unificado de los signos de puntuación mostrados hasta ahora, nombrado como M_{uni}.

Con el reconocimiento del signo de apertura se asigna el patrón melódico correspondiente. Así las preguntas se manifiestan con un arranque en anticadencia (⇑) en la primera sílaba tónica de la frase y finalizan con una inflexión circunfleja (↑↓) desde la última sílaba tónica, según lo descrito por Alcoba en [4] y que podemos apreciar en la Fig. 9.

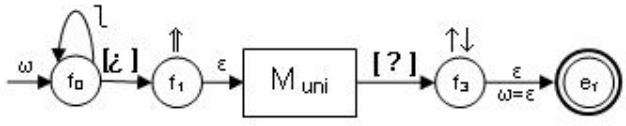


Fig. 9. Configuración general de.M *int*

M_{int} también trabaja en cooperación con M_p, asi se puede crear una jerarquía de oraciones, en la cual una instancia de M_p se encarga del análisis de la oración principal, y M_{int} puede analizar un segundo nivel de oración mediante otra instancia de M_p, la diferencia entre ambas instancias será que la responsable de la oración principal finaliza siempre con un punto (.) y la encapsulada en M_{int} no.

Un trabajo muy similar es el que se realiza con M_{adm}, para los signos de admiración, con la diferencia en la entonación. Es muy difícil tratar de sistematizar las distintas melodías de una frase contenida con signos de admiración, por ello se ha simplificado la tarea y se seleccionando aquellos rasgos comunes entre las diversas melodías. Bajo esta premisa, una frase admirativa se manifiesta con un ascenso hacia la primera sílaba tónica (⇑), un cuerpo tonal correspondiente al contenido de la frase, hasta la última sílaba tónica donde se presenta una inflexión de cadencia (⇓).

En la Fig. 10 se puede apreciar la configuración general de M_{adm}.

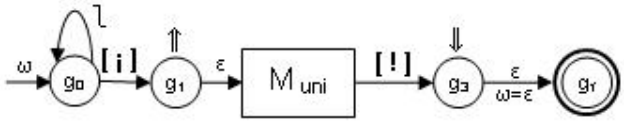


Fig. 10. Configuración general de.M *adm*.

Para el caso de los paréntesis en M_{par}, según [4] debido a que son signos que encierran elementos incidentales o aclaratorios intercalados en un enunciado, ambos se representan con una inflexión descendente (↓), tal como se puede ver en la Fig.11.

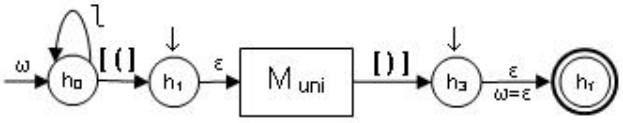


Fig. 11. Configuración general de.M *par*

El mecanismo unificado de puntuación, M_{uni}, está conformado por todos los AA mencionados anteriormente, y organizados de modo que cada uno de ellos pueda desarrollarse de manera independiente, pero manteniendo un trabajo colaborativo. En la Fig. 12 se presenta la configuración inicial de M_{uni}, la cual se limita a la lectura de caracteres del texto, y aunque una frase no contenga ningún signo de puntuación podrá ser analizada.

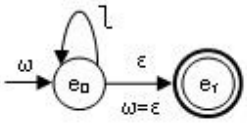


Fig. 12. Configuración inicial de.M *uni*.

La primera transición adaptativa creará la instancia del autómata correspondiente al signo de puntuación reconocido, y pasará el control sobre la cadena de entrada al mismo, el autómata respectivo asignará la pausa y patrón entonativos adecuado, luego seguirá consumiendo signos de ω hasta que encuentre un nuevo signo de puntuación, si este no le

corresponde entonces devolverá el signo a la cadena y el control regresará al estado correspondiente del autómata que realizó la transición adaptativa inicial.

La Fig. 13 muestra la configuración del autómata M_{uni} para el caso de una cadena de entrada que contenga todos los signos de puntuación analizados.

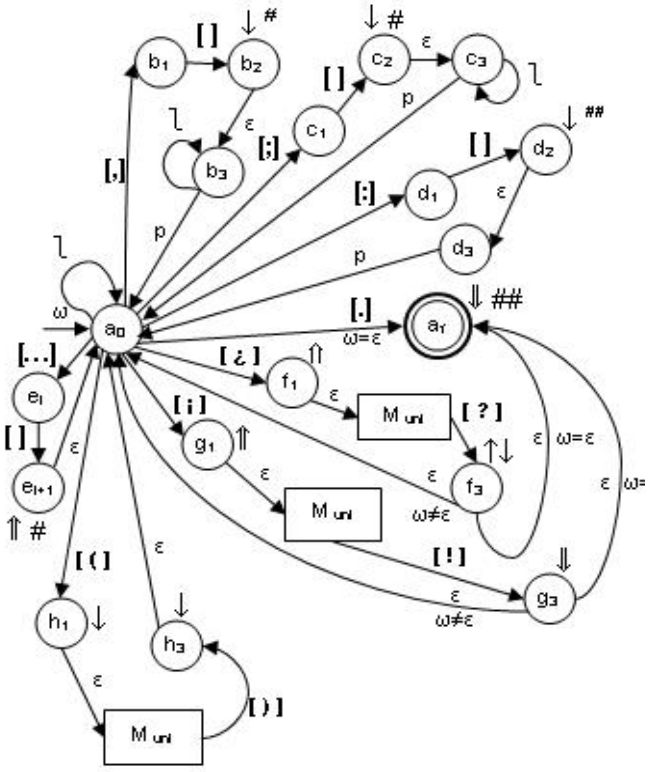


Fig. 13. Configuración de.M *uni* para un ω con todos los signos de puntuación analizados.

Las formas en las que puede ampliarse M_{uni} son variadas, por ejemplo puede crecer al interior de M_c, M_{pc}, M_{dp},o M_{ps}, tal como crecería cada uno de ellos de manera individual. Sin embargo también puede desarrollarse al nivel de la oración, en M_p, formando la estructura de análisis de un párrafo. La Fig. 14 nuestra esta última variante.

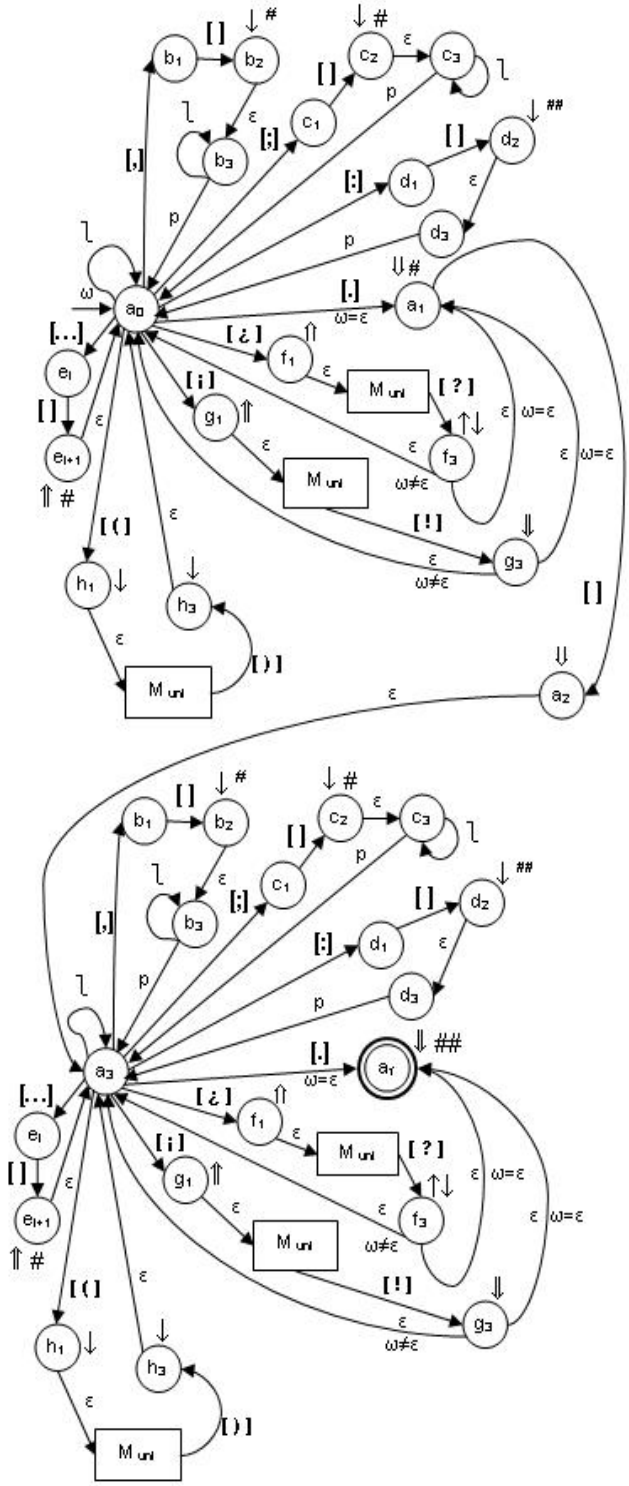


Fig. 14. Configuración de.M *uni* con crecimiento en M*p*.

VIII. IMPLEMENTACIÓN

La implementación de M_F que se presenta a continuación se realizó utilizando el entorno de desarrollo DrScheme, de uso libre para el lenguaje de programación Scheme.

```
;; Concatena los elementos de Word y los devuelve en un string
   ( define (concat_elem Word)
     (   if(zero? (length Word))
         ;;si es tónica
         ""
         ;; si es átona
         (string-append (car Word) (concat_elem (cdr Word)))
      )
   )

;; Forma las palabras fonológicas correspondientes de acuerdo
con  la lista  Word y las devuelve en forma de lista en pal_fon

   (define formar_fonologica
    ( lambda (Word pal_atona pal_fon)
      ( if(zero? (length Word))
          pal_fon
          ( if (not (member (car Word) atonas))
            (formar_fonologica (cdr Word) (list )
           ((append pal_fon (list ( string-append (concat_elem
                                          pal_atona)(car Word)))
              )
          )
        (formar_fonologica (cdr Word) (append pal_atona (list
                          (car Word))) pal_fon)
         )
       )
    )
  )
```

La implementación de M_F se encuentra insertada en el archivo spanlex.scm de FESTIVAL ubicado por defecto en la carpeta correspondiente a la voz en español

IX. Evaluación de la Calidad

Las siguientes secciones exponen la factibilidad de esta evaluación y el método elegido.

A. *Factibilidad de la evaluación de la calidad*

La calidad de un sintetizador del habla es un término multidimensional y puede ser evaluada desde tres perspectivas diferentes: su inteligibilidad, su naturalidad, y su adaptación para una aplicación (Klatt 1987). Cada una de estas perspectivas cobra diversos grados de importancia en determinados contextos, dependiendo del tipo de información que se necesita. La calidad es usada en comparaciones de elementos similares cuando no existen características cuantitativas disponibles, o estas son muy difíciles de utilizar o no deseadas.

El término "auditoría de la calidad" se refiere a la percepción humana del sonido, y sugiere que la comparación cualitativa debe ser realizada por oyentes quienes juzgarán de acuerdo con sus propias preferencias respecto a lo que mejor calidad significa. Sin embargo, algunas conclusiones pueden ser señaladas acerca de lo que es aceptado o no de manera general. Un dispositivo de medida puede ser utilizado en algunos casos, si se conoce la correspondencia entre la cantidad medida y la calidad de percepción respectiva.

La evaluación de la calidad es una tarea subjetiva, en la cual los jueces comparan lo que oyen contra lo que aprecian como perfecto. Las comparaciones entre las características de diferentes sistemas se pueden hacer, y el sistema con las características que están más cercanas a lo que la intuición del individuo califica como perfecta, tendrá la mejor calidad.

B. *Métodos de evaluación a utilizar en este trabajo*

Los métodos para la evaluación de la calidad de voz de un sintetizador, utilizados en la actualidad, se examinaron en el contexto de las necesidades del presente proyecto: la evaluación de la mejora de la calidad de voz respecto a la naturalidad, particularmente en aspectos de supra-segmentación y de puntuación.

Así mismo, se tomó en cuenta la viabilidad de aplicar las diversas pruebas respecto a la disponibilidad de los oyentes, el trabajo necesario para la preparación de las pruebas y la existencia de adaptaciones de cada prueba para el idioma castellano y la posibilidad de su obtención.

El método seleccionado fue MOS, que permite calificar la voz en sus diversas características al asignarle un valor en una escala [17]. Para el desarrollo de las pruebas se utilizarán algunas frases provenientes de diálogos cotidianos y otras, de la adaptación de las frases psicoacústicas de la Universidad de Harvard, adaptadas para el castellano, mencionadas en [14]. Debido a que se tiene especial interés en probar el desempeño de los modelos melódicos seguidos para puntuación, se efectuarán pruebas enfocadas hacia este punto por medio de la lectura de textos especialmente preparados por asesores lingüísticos.

X. Trabajos Futuros

El proyecto continuará con la implementación de los autómatas adaptativos necesarios para el tratamiento del análisis y representación de los patrones melódicos para los signos de puntuación, caso en el cual también se presenta dependencia del contexto y que podría enriquecer la calidad de la síntesis de voz.

El proyecto evaluará formalmente la calidad del sintetizador obtenido. Se realizará la comparación de la "naturalidad" entre la versión original del motor de síntesis FESTIVAL y la versión optimizada, por medio de los autómatas adaptativos implementados.

Para cada una de las pruebas, se decidió utilizar oyentes ingenuos o no preparados, con lo cual su primer contacto con la voz sintética permitirá recoger de modo más real su percepción sobre la misma. La referencia que se presentará a cada uno de los oyentes será la voz de un hispanohablante masculino nativo. Todos los participantes de las pruebas serán miembros de la comunidad universitaria de la Pontificia Universidad Católica del Perú e hispanohablantes nativos. Las pruebas se efectuarán en un ambiente cerrado, silencioso, de dimensiones medianas, y la emisión de la voz se realizará por medio de altoparlantes. Cada una de las pruebas a realizarse será guiada por una persona administradora del sistema.

XI. Conclusiones

Como resultado del trabajo realizado hasta hoy, se pueden

obtener las siguientes conclusiones:

1) La investigación en temas de síntesis de voz no es más una actividad electiva en el ámbito académico como lo eran años atrás. Dado que hoy en día su aplicación impacta el ámbito comercial alrededor de todo el mundo es necesario emplear recursos en la investigación de métodos que permitan solucionar los problemas inherentes a ella.

2) Los trabajos de síntesis para el idioma castellano, son pocos comparados con otros idiomas, como el inglés, y su naturalidad aún necesita ser mejorada en diversos aspectos.

3) La técnica de autómatas adaptativos presenta claridad, facilidad y comodidad para modelar los aspectos de la síntesis de voz dependientes del contexto en el idioma castellano tratados en el proyecto.

4) El lenguaje natural es definitivamente un tópico complejo en el cual existe mucha información a nivel lingüístico, pero menor a nivel informático. Se necesita de investigadores que analicen dicha información y la trasladen a especificaciones computables.

AGRADECIMIENTOS

Los autores reconocen las contribuciones de la profesora Beatriz Mauchi y al profesor Jorge Pérez del Departamento de Humanidades de la Pontificia Universidad Católica del Perú, por su aporte en el aspecto lingüístico del proyecto. Asimismo, al Doctor Joao José Neto de la Universidad Politécnica de Sao Paulo (Brasil), y al Doctor Joaquim Llisterri de la Universidad Autónoma de Barcelona (España) por su colaboración en el tema de métodos de evaluación de TTS.

REFERENCIAS

Periodicals (Artículos de revista):

[1] M. Mohri,, "On Some Applications of Finite-State Automata Theory to Natural Language Processing" *Journal of Natural Language Engineering*, vol. 2, n°1, p.p. 1-20, 1996. [Online] Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.36.5122

[2] B. A. Myers, "A Brief History of Human Computer Interaction Technology," *ACM interactions*, vol. 5, n°2, p.p. 44-54, March,1998. [Online].
Available:http://www.cs.cmu.edu/~amulet/papers/uihistory.tr.html

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

Books (Libros):

[4] Alcoba, S., La expresión Oral, 1ra ed., Barcelona:Ariel, 2000.

[5] A. Black, P. Taylor y R. Caley, Festival Speech Synthesis System, 1.4 ed., University of Edinburh, 2002. [Online] Available: http://www.cstr.ed.ac.uk/projects/festival/

[6] J. Cantero, M. Marti and J. Llisterri, Teoría y análisis de la entonación, Barcelona: Edicions Universitat Barcelona, 2002.

[7] E. Da Silva and E. Muszkat, Metodologia da Pesquisa e Elaboração de Dissertação, 3th ed., UFSC - Universidade Federal de Santa Catarina, Florianópolis, Santa Catarina, 2001.

[8] X. Huang, A. Acero y H. W. Hon, Spoken Language Processing: A guide to Theory, Algorithm, and System Development, 1rst ed., New Jersey: Prentice-Hall, 2001, p.p. 689-953.

[9] D. Jurafsky and J. H. Martin, Speech and language processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition, 2nd ed. New York: Prentice-Hall, 2000, p.p. 30-50.

[10]T. Navarro, Manual de la entonación Española, 2nd ed., New York, Hispanic Institute in the United States, 1954, pp. 1-60.

[11]D. O'Shaughnessy, Speech Communications: Human and Machine, IEEE Press Marketing, 2nd ed., 2000.

[12]Real Academia Española, Ortografía de la Lengua Española, Madrid: Espasa, 1999.

[13]Real Academia Española y Asociación de Academias de la Lengua Española, Diccionario Panhispánico de dudas, Madrid: Santillana, 2005.

Published Papers from Conference Proceedings (Artículos presentados en conferencias publicados):

[14]L. Aguilar, J. M. Fernández, J. M. Garrido, J. Llisterri, A. Macarrón, L. Monzón and M. A. Rodríguez, "Diseño de pruebas para la evaluación de habla sintetizada en español y su aplicación a un sistema de conversión de texto a habla" Actas del X Congreso de la Sociedad Española para el Procesamiento del Lenguaje Natural, Córdova,España, July 1994. [Online]. Available: http://liceu.uab.cat/~joaquim/publicacions/Aguilar_et_al_94_Evaluacion_Texto_Habla_Espanol.pdf

[15]J. Kominek, C. L. Bennett and A. W. Black,, "Evaluating and Correcting Phoneme Segmentation for Unit Selection Synthesis," in Proc. of the 8th European Conference on Speech Communication and Technology, pp. 313-316, Ginebra,Suiza, Sep.,2003.

[16]R. Sproat, M. Ostendorf, and A. Hunt, "The Need for Increased Speech Synthesis Research", in Report of the 1998 NSF Workshop for Discussing Research Priorities and Evaluation Strategies in Speech Synthesis , March, 1999.

Dissertations (Tesis doctorales):

[17]S. Lemmety, "Review of Speech Synthesis Technology", Master's thesis, supervised by M. Karjalainen, Helsinki University of Technology, University, Helsinki, March 1999.

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

Rosalia Edith Caya Carhuanina nació en Lima, Perú, el 27 de febrero de 1986. Se graduó en el colegio privado Beata Ana María Javouhey y estudia en la Pontificia Universidad Católica del Perú.

Entre sus campos de interés están el procesamiento de lenguaje natural y las interfaces humano-computador.

Claudia Zapata Del Río (M'2007) nació en Lima, Perú, el 03 de octubre de 1978. Se graduó en el colegio Cristo Rey, se recibió de Bachiller en Ciencias con mención en Ingeniería Informática de la Pontificia Universidad Católica del Perú y concluyó en la misma los estudios de Maestría en Ciencias de Computación.

Obtuvo el título profesional de Ingeniero en Informática mediante trabajo de tesis y es miembro del Colegio de Ingenieros del Perú.

Ejerció profesionalmente en Synopsis S.A. y actualmente es profesor auxiliar de la Pontificia Universidad Católica del Perú

Estudo comparativo entre algoritmos genéticos adaptativos e não-adaptativos aplicados à modelagem ambiental de *Peponapis* e *Cucurbita*

R. L. Stange, T. C. Giannini, F. S. Santana, J. J. Neto, A. M. Saraiva

***Resumo* — A modelagem ambiental pode ser feita combinando dados ambientais e de espécies para gerar modelos de distribuição, o que requer algoritmos não triviais. O objetivo deste trabalho é comparar os algoritmos GARP, um dos mais utilizados, e ADAPTGARP, que implementa um ferramental adaptativo. Os experimentos consideraram diversas espécies dos gêneros *Peponapis* e *Cucurbita,* respectivamente polinizadores e polinizados. Os resultados dos experimentos comparativos apresentaram mapas de distribuição das espécies com áreas potenciais de ocorrência muito semelhantes e variações de AUC, medida mais aceita em modelagem, dentro dos limites de tolerância. Foi possível concluir que os algoritmos são equivalentes e que o ferramental adaptativo está adequado, que é a principal contribuição deste trabalho. A partir dela, futuros trabalhos podem ser desenvolvidos em adaptatividade, envolvendo inserção de características mais gerais de algoritmos genéticos e inclusão de novos tipos de tratamentos, com o objetivo de gerar modelos mais precisos do que os atuais.**

***Palavras chaves* — Sistemas adaptativos, tabelas de decisão adaptativas, algoritmos adaptativos, *Peponapis*, *Cucurbita*, modelagem ambiental.**

I. Nomenclatura

AUC - Área sob a Curva
GARP – Genetic Algorithm for Rule-set Production
ROC - Gráfico do Receptor-Operador
SIG – Sistema de Informação Geográfica

II. Introdução

A MODELAGEM ambiental representa, atualmente, um dos maiores problemas para o desenvolvimento sustentável [3]. O mapeamento da distribuição geográfica das espécies é um dos seus maiores desafios, uma vez que as informações existentes são em geral raras, estão mundialmente distribuídas em diversas instituições de estudo, pesquisa ou empresas relacionadas à área ambiental, e apresentam níveis de confiabilidade e qualidade bastante variáveis [17].

A modelagem de nicho ecológico é uma das técnicas atualmente mais usadas para obter a distribuição geográfica das espécies. Os modelos obtidos são representações gráficas de uma combinação de pontos de ocorrência de espécies e de dados ambientais, tais como temperatura, altitude e precipitação. O processo de modelagem é descrito em detalhes em [20].

Para a geração de um modelo são necessários métodos específicos, capazes de calcular a função de distribuição probabilística de uma espécie, que são implementados na forma de algoritmos computacionais. Dentre os diversos algoritmos disponíveis, os mais aplicados atualmente são o GARP [http://nhm.ku.edu/desktopGARP] ou a sua principal variação, conhecida como GARP *BestSubsets* (que efetua diversas execuções do GARP e tenta selecionar os melhores modelos), e o MAXENT (baseado em entropia máxima) [http://www.cs.princeton.edu/~schapire/maxent]. Todos estão também disponíveis na solução openModeller [http:// openmodeller.sourceforge.net], que ainda integra outros algoritmos e fontes de dados.

Considerando o alto índice de utilização do GARP pelos pesquisadores em biodiversidade, foi desenvolvido um algoritmo adaptativo baseado em tabelas de decisão, denominado ADAPTGARP, inicialmente apresentado em [2]. Porém, não foi realizado um estudo detalhado a respeito dos resultados obtidos por este novo algoritmo, para determinar sua aplicação efetiva em modelagem de nicho ecológico e, com isso, dar continuidade à implementação do potencial da adaptatividade na solução deste problema.

A proposta deste trabalho é, portanto, apresentar um estudo comparativo entre os algoritmos GARP e ADAPTGARP para modelagem de nicho ecológico, usando como base para os experimentos duas espécies do gênero *Peponapis* e duas de *Cucurbita.*

O gênero *Peponapis* (Eucerini: Apidae) é constituído de abelhas solitárias com distribuição Neotropical. Esse gênero está diretamente associado às espécies de *Cucurbita*, pois coletam pólen e néctar em suas flores, e constroem os ninhos no solo, junto a essa espécie. O gênero *Cucurbita* apresenta algumas espécies que têm grande importância agrícola (abóboras) e sugere-se que a distribuição atual de *Peponapis* tenha sido influenciada pelo cultivo dessas espécies [13].

Embora atualmente ainda exista alguma controvérsia no que se refere à validação de modelos, sob a alegação de que os métodos para avaliação e comparação de algoritmos ainda não estão suficientemente bem definidos para este domínio [14],

The authors are grateful to FAPESP – Fundação de Amparo à Pesquisa do Estado de São Paulo – Brazil, for the support to the openModeller (04/11012-0) and BioAbelha (04/15801-0) projects.

R. L. Stange is a Master Student at Escola Politécnica da Universidade de São Paulo, Computation and Digital Systems Department, Av. Prof. Luciano Gualberto, travessa 3, número 158 - Cidade Universitária - São Paulo - SP - CEP: 05508-900 (rlstange@gmail.com).

T. C. Giannini is a PhD Student at Bioscience Intitute, Departament of Ecology, Rua do Matão, trav. 14, n 321 - Cidade Universitária - São Paulo - SP - CEP: 05508-900 (giannini@usp.br).

F. S. Santana is a PhD Student at Escola Politécnica da Universidade de São Paulo, Computation and Digital Systems Department, Av. Prof. Luciano Gualberto, travessa 3, número 158 - Cidade Universitária - São Paulo - SP - CEP: 05508-900 (fabiana.santana@usp.br).

J. J. Neto and A. M Saraiva are both professors at Escola Politécnica da Universidade de São Paulo, Computation and Digital Systems Department, Av. Prof. Luciano Gualberto, travessa 3, número 158 - Cidade Universitária - São Paulo - SP - CEP: 05508-900 (joao.jose@poli.usp.br, amsaraiv@usp.br).

neste estudo optou-se por utilizar a metodologia proposta em [1]. Esta é, a despeito de todas as críticas, a mais bem aceita entre os especialistas da comunidade de pesquisa em biodiversidade. Como os objetivos principais são comparar os resultados obtidos pelo ADAPTGARP e pelo GARP em relação aos seus aspectos algorítmicos e analisar a eficácia do ferramental adaptativo implementado, este trabalho não se aprofunda nos aspectos ecológicos das espécies enfocadas.

III. MATERIAIS E MÉTODOS

A. *Abordagem geral sobre tecnologias adaptativas*

O termo adaptatividade pode ser definido como a capacidade que um sistema tem para modificar o seu comportamento de forma espontânea, em resposta apenas ao seu histórico de operação e aos dados de entrada, sem interferência de agentes externos [16].

O foco principal do estudo da tecnologia adaptativa é a resolução de problemas práticos através da aplicação de modelos baseados em dispositivos adaptativos. O formalismo geral que caracteriza os dispositivos adaptativos foi apresentado pela primeira vez em [15].

Nestes dispositivos, um conjunto finito de regras pode ser representado através de uma coleção de condições que testam a situação atual do dispositivo e, quando aplicável, levam o dispositivo para uma próxima situação. Quando uma única regra é aderente à situação atual do dispositivo, a próxima situação é determinada pela regra em questão. Em situações em que mais de uma regra adere ao estado corrente do dispositivo, as possíveis situações são tratadas em paralelo e o dispositivo deve exibir uma operação não-determinística. No caso de nenhuma regra ser aderente à situação corrente, a operação do dispositivo pode ser descontinuada. Os formalismos subjacentes dos dispositivos adaptativos podem ser classificados de acordo com diversas categorias, conforme sua forma de operação. Os principais são [16]:

• Dispositivos de reconhecimento: pertencem à classe dos autômatos e são baseados na sucessão de mudanças de estados;

• Dispositivos de geração de classes das gramáticas: são baseados na aplicação sucessiva de regras de substituição;

• Dispositivos para a representação de sistemas assíncronos: incorporam mecanismos responsáveis pela representação de fenômenos de sincronização;

• Dispositivos estocásticos: incluem dispositivos semelhantes às cadeias de Markov e são capazes de representar fenômenos de caráter aleatório;

• Dispositivos de processamento: incluem dispositivos tais como as linguagens de programação adaptativas, que permitem descrever a lógica de programas com código automodificável;

• Dispositivos de auxílio à tomada de decisões: são representados principalmente pelas tabelas de decisão e pelas árvores de decisão.

A opção para o desenvolvimento deste trabalho foi o uso de tabelas de decisão adaptativas, incluídas no último item da classificação apresentada acima.

B. *Tabelas de decisão adaptativas*

Uma tabela de decisão convencional [18] é composta por colunas que representam conjuntos de regras associadas a condições e ações. A primeira coluna, partindo da primeira linha da tabela, representa um conjunto de condições e, a seguir, encontra-se um conjunto de ações. A tabela de decisão opera verificando as condições de acordo com os valores definidos nas colunas de regras. Quando a condição é satisfeita de acordo com uma determinada regra, essa regra é considerada válida e todas as ações a ela associadas são executadas. Uma tabela de decisão convencional é mostrada na Tabela 1.

TABELA 1
TABELA DE DECISÃO CONVENCIONAL [2]

		0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Condições	c_1										
	c_2										
	…										
	c_n										
Ações	a_1										
	a_2										
	…										
	z_m										

Uma versão adaptativa de uma tabela de decisão convencional pode ser obtida adicionando-se um número de linhas a ela onde são incluídas as funções adaptativas. Além disso, a cada coluna que representa uma regra simples, deve ser adicionada uma chamada para uma função adaptativa associada à execução de uma regra em particular. Com isso, sempre que uma regra é aplicada, uma função adaptativa é invocada, permitindo mudanças no conjunto de regras.

A *Tabela de Decisão Adaptativa* é definida como um dispositivo guiado por regras, onde ações adaptativas permitem mudanças no conjunto de regras da tabela de forma dinâmica [2]. A possibilidade de mudanças no conjunto de regras é, essencialmente, o que caracteriza a adaptatividade do dispositivo. A estrutura geral de uma tabela de decisão adaptativa é apresentada na Tabela 2, baseada no formato descrito em [15] e na tabela de decisão convencional descrita em [9]. Observe o acréscimo de linhas e colunas para as funções e ações adaptativas.

Na execução de uma condição em uma Tabela de Decisão Adaptativa, primeiramente são verificadas as regras não-adaptativas e, se uma única delas se aplica, as ações correspondentes são executadas. Caso mais de uma regra não-adaptativa satisfaça a condição, as ações correspondentes às mesmas devem ser aplicadas em paralelo, como previamente definido para tabelas de decisão convencionais.

Porém, se nenhuma regra satisfizer a condição, trata-se de uma condição não prevista e, no caso de uma tabela de decisão convencional, não haveria como prosseguir. Porém, na solução adaptativa deve-se verificar, nesse caso, se uma regra adaptativa se aplica, para que ações adaptativas sejam executadas e o conjunto de regras não-adaptativas seja alterado. Isto faz com que o comportamento do sistema se modifique e, uma vez aplicada à regra, utiliza-se novamente a tabela de decisão adaptativa em sua configuração resultante.

TABELA 2
TABELA DE DECISÃO ADAPTATIVA [2]

							D_1	D_2	...	D_h	R_1	R_2	...	R_h
		0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**h + 4**				
Condições	c_1													
	…													
	c_n													
Ações	a_1													
	…													
	z_m													
Funções Adaptativas	b_{a_1}													
	…													
	b_{a_f}													

C. O problema de modelagem de nicho ecológico

O problema da modelagem de nicho ecológico está relacionado às condições que permitem à sobrevivência de uma determinada espécie, o que tipicamente conhecido como nicho, tais como temperatura, geologia e vegetação, sendo desconsiderados os fatores externos, como a influência do homem, as interações bióticas e as barreiras geográficas [16].

O conceito de nicho ecológico é definido em [16] como o conjunto de condições ecológicas e ambientais sob as quais uma espécie é capaz de manter suas populações sem a necessidade de imigração e, em [8], como sendo um conjunto de condições ecológicas com as quais as populações conseguem se manter sem interferência, que pode ser representado por um espaço ecológico/ambiental multidimensional, onde as condições são as dimensões deste espaço.

Portanto, baseados no conceito de nicho ecológico, a geração de modelos de distribuição geográfica de espécies precisa combinar os dados disponíveis correspondentes aos pontos de ocorrência ou ausência das espécies com os dados ambientais pertinentes ao seu nicho.

Os pontos de ocorrência correspondem aos locais em que as espécies foram encontradas. Os pontos de ausência são menos confiáveis, mas em princípio representam os pontos estudados em que a espécie foi procurada e não foi encontrada (a confiança do ponto está diretamente relacionada à metodologia de busca). Os dados ambientais são apresentados na forma de camadas, normalmente chamadas de *layers*. Todos devem ser disponibilizados na forma de coordenadas geo-referenciadas, considerando a região específica de estudo.

Um de nicho ecológico modelo é uma função de distribuição da probabilidade de ocorrência de uma espécie em cada coordenada geográfica da região de estudo e, usualmente, é projetado em um mapa geográfico em escala de cores, onde cada cor representa uma probabilidade específica. O processo de modelagem é detalhado em [19], mas é importante destacar o papel dos algoritmos nesse processo, pois são eles os responsáveis por obter, a partir de dados usualmente escassos, o mapeamento da distribuição das espécies.

D. O algoritmo GARP

O GARP é um algoritmo genético, neste contexto adaptado para o propósito de realizar modelagem de nicho ecológico. O termo algoritmo genético foi introduzido por [7] para explicar os mecanismos da evolução natural e para transformar estes mecanismos em sistemas artificiais, com o objetivo de resolver problemas difíceis de busca e otimização.

Esse algoritmo utiliza conceitos de nicho ecológico para definir uma população e, a partir disso, trabalha no processamento das regras que definem esse nicho. O algoritmo procura encontrar o nicho de uma espécie analisando diversos *layers* ambientais. As regras a que o GARP se refere estão diretamente relacionadas a estes *layers*.

A implementação do GARP considera três tipos de regras: *ranges rules*, *negate rules* e *logit rules*. *Range rules* podem ser escritas, em linguagem natural, como "Se o valor da variável ambiental X está entre os valores A e B, onde A é menor que B, então a espécie está presente". *Negative rules* capturam os pontos de presença das espécies para valores de uma variável ambiental a partir de um determinado intervalo: "Se o valor da variável ambiental X é menor que o valor de A ou o valor da variável X é maior o valor de B, onde A é menor que B, então a espécie está presente". *Logit rules* refletem os pontos de ocorrência das espécies como uma função de distribuição de probabilidade em termos de diferentes variáveis ambientais: "Considerando que as variáveis x_1, x_2,...., x_n determinam os pontos de presença de espécies em uma determinada área geográfica, a probabilidade de ocorrência de espécies em um ponto específico p_0 desta área é dado por: Prob(a espécie está presente em p_0) = $1 - e^{x_1 + x_2 + ... + x_n}$".

A Figura 1 apresenta o GARP em pseudo-código, de acordo com a implementação atual disponível no sistema openModeller (http:// openmodeller.sourceforge.net), que é de fato a mais utilizada. A implementação considera a criação de um conjunto de regras, chamado de *_offspring*, que também armazena o conjunto de regras resultante de cada iteração do algoritmo. A quantidade de regras é limitada por *_popsize*, que é um parâmetro constante pré-definido e que armazena o tamanho da população. A operação de colonização consiste em gerar regras para *_offspring*, sendo que inicialmente este conjunto de regras deve ser preenchido com exatamente *_popsize* regras. O método *initializeGARP* é executado apenas uma vez, para gerar o conjunto inicial de regras. Depois disso, o método *interateGARP* é chamado, onde a cada iteração, o tamanho do conjunto de regras é controlado, a fim de que o tamanho estabelecido em _popsize seja mantido. Caso a quantidade de regras em *_offspring* seja menor do que *_popsize*, a operação de colonização é chamada para criar novas regras; caso contrário, algumas regras devem ser removidas de *_offspring*, sendo que a remoção deve sempre tentar manter o maior valor possível para a função de adaptação do conjunto de regras, conhecida como função de *fitness*. O método *keepFittest* realiza a substituição de uma regra no conjunto de regras se ela fornecer uma performance melhor do que a regra similar, ou insere uma regra que seja capaz de melhorar a função de *fitness* do conjunto de regras, se nenhuma regra similar for encontrada. Esta operação garante que o valor da função de *fitness*, na pior das hipóteses, se mantém. Na seqüência, os métodos tradicionais dos algoritmos genéticos são executados, que são: *Select*, *Crossover* e *Mutate*.

Após estas operações, o desempenho de cada regra no conjunto de regras é individualmente avaliado e, em seguida, um filtro remove as regras que resultam em redução do valor da função de *fitness*. Por último, o número de iterações é acrescido de um, a fim de estabelecer um limite máximo para o número de iterações. Portanto, o método *iterateGARP* é repetido até que ocorra a convergência (os valores de convergência são parâmetros fixos e pré-definidos) ou até que o número máximo de iterações seja atingido. Portanto, o número de iterações executadas de fato depende dos dois critérios, sendo que o algoritmo para sua execução no que for satisfeito primeito.

E. O algoritmo ADAPTGARP

O ADAPTGARP é uma versão adaptativa do GARP proposta em [2], considerando a versão implementada no openModeller [http://openmodeller.sourceforge.net] e baseado no uso de Tabelas de Decisão Adaptativas.

Os operadores *Crossover* e *Mutation* (na implementação, este operador aparece com o nome de *Mutate*) trabalham como manipuladores de regras através de outras regras e são chamados pelo algoritmo principal. A adaptatividade consiste na inclusão dos respectivos operadores genéticos adaptativos, chamados de *AdaptCrossover* e *AdaptMutation*, que são capazes de alterar as regras a serem manipuladas pelos operadores *Crossover* e *Mutation*. A estrutura de dados é descrita de acordo com as técnicas adaptativas presentes em [15].

O operador genético *Crossover* escolhe dois indivíduos e troca informações entre eles, produzindo um novo individuo. No caso do GARP, isto se aplica aos limites dos dados referentes ao nicho da espécie em estudo, atuando sobre os seus limites. A técnica requer que regras de dispositivos não-adaptativos sejam codificadas em tabelas de decisão. Portanto, é necessário definir funções adaptativas para manipulação das regras inseridas na tabela de decisão. Essas funções adaptativas são inseridas como parte da tabela de decisão adaptativa, o que resulta em uma representação dos operadores genéticos como parte da tabela de decisão adaptativa e essas funções permitem a mudança dinâmica no conjunto de regras.

A implementação das regras do GARP foi baseada em [15]. Por exemplo, considere *h* o intervalo associado às variáveis ambientais *k*, como mostra a Tabela 3 a seguir.

TABELA 3
ESQUEMA DE REGRAS DO GARP

	$\mathbf{R_1}$	$\mathbf{R_2}$	$\mathbf{R_3}$		$\mathbf{R_h}$
$x_1 \geq$	A_{11}	A_{21}	A_{31}	...	A_{h1}
$x_1 \leq$	B_{11}	B_{21}	B_{31}	...	B_{h1}
...	...	...	...	...	...
$x_k \geq$	A_{1k}	A_{2k}	A_{3k}	...	A_{hk}
$x_k \leq$	B_{1k}	B_{2k}	B_{3k}	...	B_{hk}

No GARP, um exemplo de tipo de regra associado a esta tabela poderia ser: "Se x1 ∈ [A11, B11] e x2 ∈ [A12, B12] e… xk ∈ [A1k, B1k] então a espécie está presente."

A regra R_j significa que, $1 \leq j \leq h$, para cada variável ambiental x_i, onde $1 \leq i \leq k$, então a espécie está presente na localização associada. Esta é a principal representação da

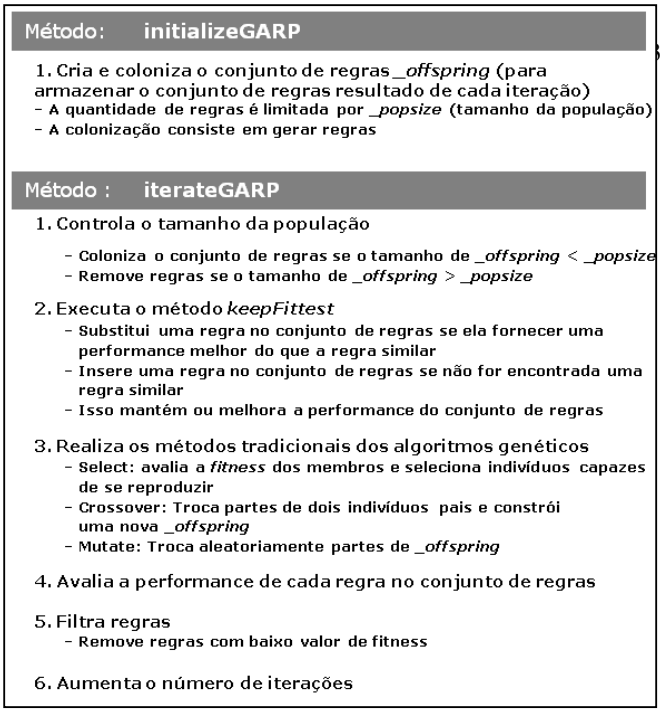


Figura 1 – Algoritmo GARP, implementação openModeller.

estrutura de dados para operadores genéticos adaptativos descritos a seguir. O número de colunas da Tabela 4 podem mudar com a execução do operador *AdaptCrossover*. Já o *AdaptMutation* atua em regras já existentes na tabela de decisão, alterando os limites dos intervalos.

Algumas decisões devem ser tomadas sobre como lidar com as regras. Qualquer regra é referida pelo índice da tabela, por exemplo, sua respectiva coluna. Dado que se trata de um dispositivo adaptativo, o número de regras pode ser alterado e o conjunto de regras a ser considerado deve ser uma estrutura matricial dinâmica, que mantém a numeração seqüencial das colunas.

Para especificar as funções adaptativas, funções auxiliares são aplicadas, que são descritas a seguir:

a) $Ux(y) = \begin{cases} 0, & \text{if } x < y \\ 1, & \text{if } \geq y \end{cases}$

b) A função *rand(n)* gera um numero inteiro aleatório entre *1* e *n*, e a função *rand(a,b)*, produz um número inteiro aleatório entre *a* e *b*, quando *a* e *b* inteiros, ou um número real aleatório entre *a* e *b*, quando *a* e *b* são números reais (isto depende da definição dos dados analisados sobre a espécie ou *layer*).

Na implementação do *AdaptCrossover*, considera-se que o operador genético *Crossover* escolhe dois indivíduos e troca informações genéticas entre eles, produzindo um novo individuo. A Tabela 4 especifica a função adaptativa do operador *AdaptCrossover* na tabela de decisão.

A célula da primeira linha na primeira coluna é usada para identificar o nome da função. As outras células desta linha ficam vazias. A segunda coluna, entre a segunda e a quinta linha, são utilizadas para especificar os parâmetros do operador *AdapCrossover*. Os parâmetros *i* e *j* são números inteiros, representando os índices das regras a serem cruzadas. Os parâmetros *p* e *q* são números inteiros que especificam seções de intervalos que são herdadas pelo novo indivíduo, resultantes do cruzamento entre as regras *i* e *j*. As linhas

restantes especificam as condições adaptativas, de acordo com as funções adaptativas.

A função adaptativa pode adicionar uma nova linha na tabela, que é expressa pelo símbolo "+" no cabeçalho da terceira coluna. Outra ação elementar é o símbolo "-", que remove regras. Este símbolo também é aplicado na versão adaptativa do operador de mutação, mostrado na tabela 5. O operador *AdaptCrossover* cria novas regras para que todos os intervalos com índice "< p" ou "≥ q" herdem a regra R_i.

TABELA 4
TABELA DE DECISÃO ADAPTATIVA PARA ADAPTCROSSOVER

		+
	AdaptCrossover	
	I	
	J	
	P	
	Q	
$x_1 \geq$		$A_{i1} + (A_{j1} - A_{i1})(U_1(p) - U_1(q))$
$x_1 \leq$		$B_{i1} + (B_{j1} - B_{i1})(U_1(p) - U_1(q))$
...		...
$x_k \geq$		$A_{ik} + (A_{jk} - A_{ik})(U_k(p) - U_k(q))$
$x_k \leq$		$B_{ik} + (B_{jk} - B_{ik})(U_k(p) - U_k(q))$

As regras do *AdaptCrossover* podem ser matematicamente resumidas por:

$$x_h \geq \begin{cases} A_{ih}, & \text{if } h < p \\ A_{jh}, & \text{if } p \leq h < q \\ A_{ih}, & \text{if } h \geq q \end{cases}$$

e

$$x_h \leq \begin{cases} B_{ih}, & \text{if } h < p \\ B_{jh}, & \text{if } p \leq h < q \\ B_{ih}, & \text{if } h \geq q \end{cases}$$

para todo $1 \leq h \leq k$.

O operador genético de mutação, o *Mutation*, seleciona um indivíduo e pode modificar as informações a ele pertinentes, produzindo a mutação do indivíduo, no caso das regras do GARP. A Tabela 5 apresenta a função adaptativa *AdaptMutation* como uma tabela de decisão [2]. A célula da primeira linha e primeira coluna é usada somente para identificar o nome da função. Na segunda coluna, as linhas entre dois e cinco especificam os parâmetros do *AdaptMutation*. O parâmetro *j* é um número inteiro que representa o índice associado a uma regra; os valores de *k* e *g* são números inteiros aleatórios; *k* especifica um gene a ser modificado; *g* especifica se o *k*-ésimo gene será modificado ou não; e os parâmetros *a* e *b* são números inteiros aleatórios que representam os novos limites para o intervalo, se ele for alterado.

TABELA 5
TABELA DE DECISÃO ADAPTATIVA PARA ADAPTMUTATION

		-	+
	AdaptMutation		
	j		
	r		
	g		
	a		
	b		
$x_1 \geq$		A_{j1}	$A_{j1}U_r(g) - (a - A_{j1})U_r(g + 1)$
$x_1 \leq$		B_{j1}	$B_{j1}U_r(g) - (b - B_{j1})U_r(g + 1)$
...			
$x_k \geq$		A_{jk}	$A_{jk}U_r(g) - (a - A_{jk})U_r(g + 1)$
$x_k \leq$		B_{jk}	$B_{jk}U_r(g) - (b - B_{jk})U_r(g + 1)$

As regras do *AdaptMutation* podem ser matematicamente resumidas por:

$$x_h \geq \begin{cases} A_{jh}, & \text{if } r < g \\ a, & \text{if } r = g \\ A_{jh}, & \text{if } r > g \end{cases}$$

e

$$x_j \leq \begin{cases} B_{jh}, & \text{if } r < g \\ b, & \text{if } r = g \\ B_{jh}, & \text{if } r > g \end{cases}$$

para todo $1 \leq h \leq k$.

IV. ESTUDO COMPARATIVO ENTRE O GARP E O ADAPTGARP

Esta seção apresenta os resultados do estudo comparativo entre os algoritmos GARP e ADAPTGARP dentro do contexto do problema de modelagem de nicho ecológico, a fim de validar o ferramental adaptativo proposto pelo uso das Tabelas de Decisão Adaptativas.

Para comparar os dois algoritmos, foram utilizados os pontos de ocorrência de duas espécies de *Peponapis* e duas de *Cucurbita*, obtidas a partir de sítios na Internet que apresentam dados de herbários e coleções. São eles: rede *speciesLink* [http://splink.cria.org.br/], CONABIO (Comisión Nacional para el Conocimiento y uso de la Biodiversidad – [http://www.conabio.gob.mx/] e GBIF (Global Biodiversity Information Facility [http://www.gbif.org/]. Foram também consideradas as informações apresentadas em [10][11][12], onde essas espécies foram extensamente estudadas, do ponto de vista biológico. A Tabela 6 resume os dados utilizados no experimento.

TABELA 6
ESPÉCIES E NÚMERO DE PONTOS DE OCORRÊNCIA ANALISADOS

Nome Científico	**N. de pontos de ocorrência**	**Países que apresentam registro da espécie**
P. timberlakei Hurd & Linsley, 1964	31	EUA e México
P. azteca Hurd & Linsley, 1966	60	México
C. palmata S. Watson	90	EUA e México
C. radicans Naudin	39	México

As coordenadas geográficas, organizadas por municípios, foram obtidas a partir de dois principais sítios da Internet. Para o Brasil, foi utilizada uma ferramenta oferecida pela própria rede *speciesLink*, denominada *geoloc*, e para os demais países utilizou-se o sítio da Internet denominado Tageo [http://www.tageo.com]. A conversão para graus decimais, a limpeza e a correção dos dados foram feitas também através das ferramentas presentes no sítio do *speciesLink* e denominadas, respectivamente, de *conversor*, *datacleaning* e *speciesmapper*.

A modelagem de nicho ecológico das quatro espécies foi feita utilizando os algoritmos GARP e ADAPTGARP. Porém,

como a análise aqui proposta sempre teve o objetivo e o compromisso de estar o mais próxima possível da realidade, e a maior parte da comunidade de pesquisa em biodiversidade utiliza a versão do GARP conhecida como "GARP with best subsets", o ADAPTGARP também foi implementado seguindo os mesmos padrões. Portanto, ao invés de utilizar diretamente o algoritmo proposto em [2], foi implementada uma versão do ADAPTGARP usando os mesmos conceitos de "best subsets", resultando na implementação do "ADAPTGARP with best subsets".

Na modelagem feita pelo GARP, os pontos de ocorrência são divididos em dois conjuntos de dados para cada iteração do algoritmo, sendo que um deles é utilizado para testar a qualidade do modelo gerado (dados para teste) e o outro é usado para desenvolver efetivamente o modelo (dados para treino). O modelo é composto por regras que são desenvolvidas através de refinamento, teste e seleção de subconjuntos dos dados de teste. O desempenho do modelo foi medido através do método conhecido como "área sob a curva" (AUC - *area under curve*) do gráfico do receptor-operador (ROC - *receiver operating characteristic*), que é calculada a partir dos valores da matriz de confusão [4]. Como discutido anteriormente, embora não haja consenso sobre os métodos de validação de modelos, este método é o mais adotado pela comunidade de pesquisa em biodiversidade atualmente e, portanto, foi considerado como a melhor alternativa de comparação também para este estudo.

Foram feitos dois tipos de testes: um deles, denominado *teste interno*, usa a totalidade dos pontos de ocorrência obtidos e o outro, denominado *teste externo* (ou *teste independente*), que efetua divisões dos dados em partições, de forma aleatória e sem reposição, sendo que cada partição, por sua vez, será dividida em dados de teste e dados de treino (Tabela 7) no momento da execução dos algoritmos. Este teste é bastante importante, pois associa uma espécie de análise de sensibilidade nos modelos gerados, em relação aos dados de entrada.

TABELA 7
NÚMERO DE PARTIÇÕES DE CADA ESPÉCIE PARA O TESTE EXTERNO UTILIZANDO-SE DADOS DE TESTE (30%) E DADOS DE TREINO (70%).

Nome científico	N. de partições	N. de pontos para teste (30%)	N. de pontos para treino (70%)
C. palmata	3	9	21
P. azteca	2	9	21
P. timberlakei	1	9	21
C. radicans	1	9	21

Foi usado o limite proposto por [22], que estipula que os dados de treino devem conter no mínimo 20 pontos de ocorrência. A proporção adotada entre número de pontos de treino e número de pontos de teste foi de 70% e 30%, respectivamente.

Foram calculadas a média e o desvio padrão dos valores de AUC obtidos para cada partição de dados de cada espécie. [1] sugere que os resultados do AUC médio sejam interpretados da seguinte forma: excelente, quando acima de 0,90; bom, entre 0,90-0,80; razoável, entre 0,80-0,70; pobre, entre 0,70-0,60; e falho, entre 0,60-0,50.

Foi utilizada uma máscara com grade de 9km^2 de resolução, abrangendo o continente americano (aproximadamente 765 mil células) e 37 camadas ambientais, com mesma resolução, obtidas no sítio do Worldclim [http://www.worldclim.org]. As camadas utilizadas na modelagem referem-se à: 1) Altitude; 2) Precipitação para 12 meses; 3) Temperatura máxima para 12 meses; e 4) Temperatura mínima para 12 meses.

Para as análises das informações geográficas obtidas foi utilizado o programa ArcGIS, versão 9.2 [http://www.esri.com/software/arcgis/index.html], que também é uma das ferramenta de SIG mais adotadas pela comunidade de pesquisa em biodiversidade.

V. RESULTADOS

A Figura 2 mostra os mapas resultantes da modelagem de cada espécie a partir de cada algoritmo.

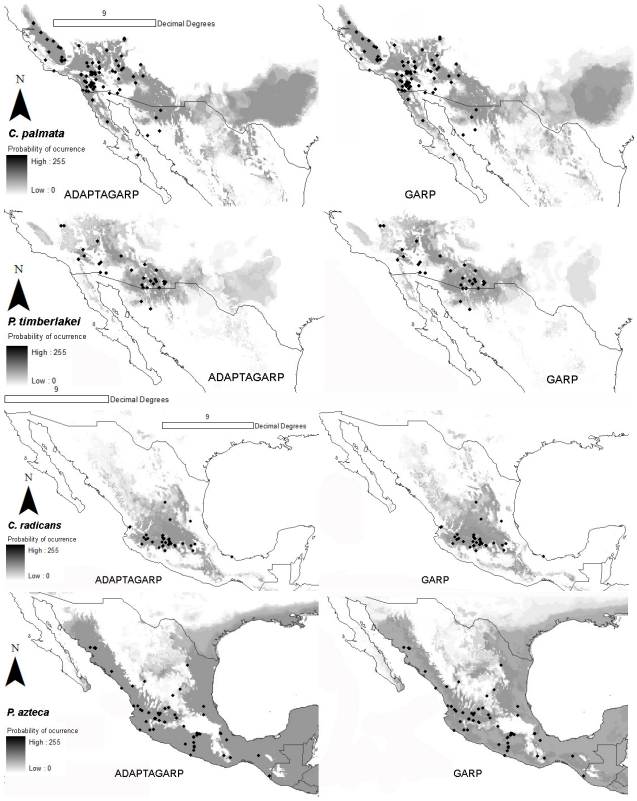


Figura 2 – Mapas de distribuição evidenciando as áreas potenciais da espécie *Cucurbita palmata*, *Peponapis timberlakei, C. radicans* e *P. azteca*, obtidos através da modelagem utilizando-se GARP e ADAPTGARP "with best subsets".

A Tabela 8 apresenta os resultados de AUC obtidos a partir do teste interno executado com a totalidade de pontos de ocorrência para cada espécie e a Figura 3 mostra o gráfico comparativo entre estes valores.

TABELA 8
RESULTADOS DE AUC DO TESTE INTERNO

Nome Científico	GARP AUC	ADAPTGARP AUC
C. palmata	0.90	0.90
P. timberlakei	0.86	0.89

C. radicans	0.97	0.97
P. azteca	0.97	0.96

A tabela 9 apresenta as médias e desvios padrão dos AUC obtidos através do teste externo, realizado com as partições dos dados para a execução do GARP e do ADAPTGARP "with best subsets". Da mesma forma, a Figura 4 apresenta o resumo gráfico dos resultados obtidos.

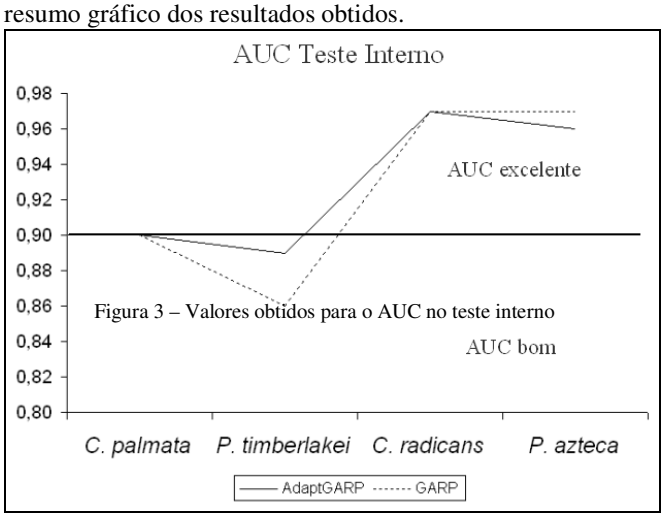


Figura 3 – Valores obtidos para o AUC no teste interno

Figura 3 - Valores obtidos para o AUC no teste interno

TABELA 9
RESULTADOS DE AUC DO TESTE EXTERNO

Nome Científico	**GARP**			
	Teste (30%)		**Treino (70%)**	
	Média	**DP**	**Média**	**DP**
C. palmata	0.87	0.12	0.96	0.03
P. timberlakei	0.89	-	0.94	-
C. radicans	0.93	-	0.95	.
P. azteca	0.95	0.01	0.96	0.04
	ADAPTGARP			
	Média	**DP**	**Média**	**DP**
C. palmata	0.83	0.09	0.91	0.02
P. timberlakei	0.89	-	0.90	-
C. radicans	1.00	-	0.95	-
P. azteca	0.97	0.05	0.96	0.05

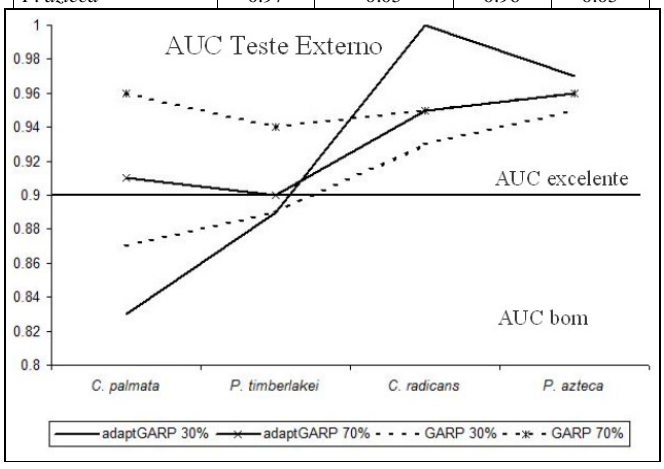


Figura 4 – Valores obtidos para o AUC no teste externo

VI. ANÁLISE DOS RESULTADOS, INOVAÇÕES E DISCUSSÕES SOBRE TRABALHOS FUTUROS

Observe que, mesmo visualmente, é possível notar a semelhança entre os modelos gerados, através da observação dos mapas obtidos para cada espécie, utilizando-se os dois algoritmos, conforme apresentado na Figura 2. Observe que as áreas potenciais de ocorrência são de fato muito semelhantes. Da mesma forma, os resultados de AUC do teste interno apresentados na Tabela 8 mostram valores semelhantes para os dois algoritmos.

No entanto, as duas espécies de *Peponapis* apresentaram resultados diferentes para o teste interno (Tabela 8): o ADAPTGARP resultou um modelo com valor mais alto de AUC para *P. timberlakei* e com um valor mais baixo para *P. azteca.* Em relação ao teste externo, Tabela 9, é possível observar a variação dos resultados de AUC para os dois algoritmos, especialmente nas espécies *C. palmata* e *P. timberlakei*. Apenas para essas duas espécies, os valores estiveram abaixo de 0.90. Porém, uma flutuação como esta já era esperada, devido às diferentes amostras selecionadas aleatoriamente e que, portanto, podem variar de uma execução para outra. De qualquer forma, essas diferenças não representam uma variação estatisticamente significativa para algoritmos genéticos.

É importante notar, para uma correta análise dos resultados, que a implementação atual do ADAPTGARP provê um ferramental adaptativo baseado em tabelas de decisão adaptativas e normalização de resultados. Considerando que o ADAPTGARP utiliza mecanismos adaptativos apenas para implementar a simulação do algoritmo genético, através de uma tabela de decisão adaptativa, é de se esperar que macroscopicamente os resultados apresentados sejam similares. Mais especificamente, nesta implementação, os eventuais desvios apresentados pelo GARP nas operações de *Crossover* e *Mutation* são normalizados nas respectivas implementações do *AdaptCrossover* e *AdaptMutation* do ADAPTGARP. Embora não se possa afirmar sem um estudo mais aprofundado, esta normalização provavelmente é a justificativa para a melhoria dos resultados medidos do AUC, nos casos em que isso ocorreu, embora os valores obtidos ainda continuem bastante próximos.

Embora possa se argumentar que estes resultados ainda tenham que ser submetidos à avaliação da comunidade de pesquisa em geral, através da livre disponibilização do algoritmo para experimentos com diferentes espécies com características e áreas de ocorrência também diversas, o ponto principal deste estudo é que os resultados comprovam que esta abordagem é uma opção válida para a modelagem de nicho ecológico, como era o objetivo inicial. Portanto, uma vez implementado o ferramental adaptativo, este pode e deve continuar a ser explorado e evoluído, o que deve ser feito através da inserção de "conhecimento" ao ADAPTGARP.

Além da própria implementação do "ADAPTGARP with best subsets", que vai incentivar seu uso pela comunidade de pesquisa em biodiversidade, este trabalho foi fundamental no sentido de estabelecer uma validação mais precisa da simulação deste algoritmo genético usando tabelas de decisão adaptativas.

Com relação ao desempenho, como já havia sido avaliado em [2], constatou-se mais uma vez que os algoritmos se

comportam de maneira similar. Este fato também era previsto, pela própria estrutura do algoritmo que realiza uma simulação do GARP usando Tabelas de Decisão Adaptativas. Isto também acontece nos resultados de geração de mapas e valores de AUC. As variações observadas nos resultados, também neste caso, ocorrem devido à flutuação estatística e são perfeitamente aceitáveis.

A comprovação de que o algoritmo implementado, utilizando Tabelas de Decisão Adaptativas como dispositivo adaptativo, efetivamente simula o algoritmo genético original permite evoluções significativas no algoritmo, em termos computacionais. Portanto, o ferramental adotado pode ser considerado válido e pronto para ser trabalhado.

Uma delas é que, na implementação do próprio GARP, por sua estrutura, algumas características importantes dos algoritmos genéticos não estão devidamente cobertas, como considerar populações e probabilidades relacionadas aos seus operadores (além de crossover e mutação, por exemplo, podemos citar o caso da mortalidade). Essas características podem ser consideradas na tabela de decisão definindo-se novas condições a serem testadas, novas condições a serem incluídas e novas ações adaptativas a serem convenientemente executadas, de tal modo que os novos elementos passem a ser considerados na tomada de decisão por ela representada.

Outra contribuição é a inclusão de um mecanismo de implementação de máscaras para que determinados dados se tornem visíveis ou invisíveis aos algoritmos, conforme a conveniência. Por exemplo, isso pode ser aplicado para a seleção automática de layers ambientais, possibilitando incluir na tabela de decisão adaptativa ações para filtrar layers relevantes para cada determinado experimento. Essa técnica pode ser aplicada para considerar fatos que sejam do conhecimento do pesquisador, alterando as tabelas com o objetivo de reduzir a quantidade de dados a serem levados em conta para as tomadas de decisão. Ilustra essa situação o conhecimento prévio de valores mínimos e máximos de variáves ambientais (altitude, temperatura, precipitação), que estão fortemente correlacionadas com ausência ou presença de determinadas espécies. Por exemplo, uma espécie que seja bem conhecida e que só ocorra em regiões desérticas, apresentará restrições relativas ao clima (nesse caso, seco), índice pluviométrico (baixo) e temperatura (baixa). O que neste exemplo foi classificado como "seco" ou "baixo(a)" pode, através do conhecimento do pesquisador (no caso, biólogo, ecólogo ou especialista de áreas afins) sobre a espécie, ser transformado em valores numéricos de mínimos e máximos, restringindo os extremos da tabela de decisão adaptativa para valores expressos em números reais.

Portanto, é possível evoluir a solução e inserir o conhecimento do pesquisador sobre a espécie que está em estudo para a construção das funções adaptativas. Portanto, ao invés de se normalizar os resultados para evitar grandes desvios, a tabela de decisão adaptativa pode utilizar valores reais baseados em fatos reais para limitar o intervalo. Se a necessidade de normalização surgir, ela ainda pode ser aplicada, mas a tendência é minimizar ainda mais o erro, por estar agora associada a valores baseados em conhecimento pré-estabelecido.

Outra inovação é acrescentar ao ferramental adaptativo fatores de correlação, para o caso de estudos de espécies co-dependentes. Por exemplo, no caso de espécies que dependem de certos polinizadores específicos, é necessário considerar o estudo da distribuição do polinizador na distribuição da espécie vegetal enfocada. Analogamente, o estudo da distribuição do polinizador deve considerar a distribuição da espécie vegetal, pois esta é essencial como fonte de recursos para sua sobrevivência.

Além destes aspectos, diversos itens relacionados à seleção de layers, algoritmos e parâmetros também podem ser acrescentados ao ADAPTGARP, de forma que o modelo fique sempre o mais próximo possível daquilo que já é conhecido sobre a distribuição geográfica da espécie, aproximando ao máximo o modelo da realidade, que é o objetivo principal da modelagem de nicho ecológico. Com relação à seleção automática de layers, por exemplo, é possível automatizar a inclusão de novas regras e ações adaptativas que permitam ao algoritmo deduzir quais layers são essenciais e quais podem ser ignoradas em cada experimento, baseado na dedução das correlações existentes entre grupos de layers.

Cada uma dessas contribuições, brevemente descritas nesta seção, serão alvos de trabalhos a serem desenvolvidos em um futuro próximo, agora que já é sabido que o ferramental baseado em Tabelas de Decisão Adaptativas é perfeitamente viável, em termos computacionais e em termos de modelagem de nicho ecológico.

VII. Conclusão

Os dois algoritmos, GARP e ADAPTGARP, se mostraram equivalentes em relação aos resultados de AUC e das áreas potenciais evidenciadas nos mapas. Portanto, pode-se considerar que os resultados obtidos na modelagem de nicho ecológico de ambos são equivalentes. Em termos biológicos, essas diferenças até poderiam ser mais bem analisadas através de outros experimentos com outras metodologias ou organismos, mas o que foi feito neste trabalho, considerando a variedade e a quantidade de testes realizada, já é suficiente para estabelecer que a solução adotada parece ser uma boa base para o desenvolvimento de novas pesquisas em adaptatividade.

Em trabalhos futuros, a solução aqui validada será evoluída de forma a permitir inserir o conhecimento prévio do pesquisador nas Tabelas de Decisão Adaptativas sobre a espécie que está em estudo. Desta forma, será possível controlar melhor o comportamento do algoritmo em determinadas situações, inserir novas funcionalidades relacionadas aos algoritmos genéticos e, como resultado, melhorar a qualidade dos modelos de nicho ecológico através da exploração da tecnologia adaptativa.

Agradecimentos

Os autores agradecem à FAPESP – Fundação de Amparo à Pesquisa do Estado de São Paulo – Brazil, pelo apoio financeiro aos projetos openModeller (04/11012-0) e BioAbelha (04/15801-0).

REFERÊNCIAS

[1] ARAÚJO, M. B., PEARSON, R. G., THUILLER, W. & Erhard, M. 2005. *Validation of species-climate impact models under climate change*. Global Change Biology v. 11, s/n, p.1504-1513.

[2] BRAVO, C., NETO, J.J., SANTANA, F.S., SARAIVA, A.M., 2007. *Towards and adaptive implementation of genetic algorithms*. Latin American Workshop on Biodiversity Informatics – INBI 2007 / CLEI 2007 – XXXIII Conferência Latino Americana de Informática. 9–12 Outubro. San José, Costa Rica.

[3] BRUNDTLAND,G., 1987. Our CommonFuture: TheWorld Commission on Environment and Development. Oxford University Press, Oxford.

[4] ELITH, J., et. al. 2006. Novel methods improve prediction of species' distributions from occurrence data. Ecography v. 29, n. 2, p. 129-151.

[6] GILDERSLEEVE, T. R. Decision Tables and Their Practical Application in Data Processing. Englewood Cliffs, N. J., Prentice Hall, 1970.

[7] HOLLAND, J. H. 1975. Adaptation in Natural and Artificial Systems. University of Michigan Press, Ann Arbor.

[8] HUTCHINSON, G.E. (1981). Introducción a la ecologia de poblaciones. Barcelona, Editorial Blume. 492 p.

[9] HUGHES, M. L., SHANK, R. M., STEIN, E. S. Decision Tables. Midi Publications, Management Development Institute, Divisions of Information, Industries, Inc., Wayne, Pennsylvania, 1968.

[10] HURD, P.D. LINSLEY, E.G. 1964. The squash and gourd bees - genera Peponapis and Xenoglossa - inhabiting America North of Mexico Hilgardia v. 35, p. 375-477

[11] HURD, P.D. LINSLEY, E.G. 1966. The Mexican squash and gourd bees of the genus Peponapis Ann. of Entomol. Soc. Of America, v. 59, p.835-51

[12] HURD, P.D. LINSLEY, E.G. 1967. South American squash and gourd bees of the genus Peponapis Ann. of Entomol. Soc. Of America, v. 60, p. 647-61

[13] HURD, P. D. Jr., LINSLEY, E. G. and WHITAKER, T. W. 1971. Squash and gourd bees (Peponapis, Xenoglossa) and the origin of the cultivated Cucurbita. Evolution 25, 218-234.

[14] LOBO, J. M., JIMÉNEZ-VALVERDE, A. & REAL, R. 2008. AUC: a misleading measure of the performance of predictive distribution models. Global Ecology and Biogeography v. 17, n. 2, p. 145-151.

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

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

[17] PETERSON, A. T., PAPES, M. & SOBERON, J. 2008. Rethinking receiver operating characteristic analysis applications in ecological niche modeling. Ecological Modelling v. 213, n. 1, p. 63-72.

[18] PHILLIPS, S. J., ANDERSON, R. P. & SCHAPIRE, R. E. 2006. Maximum entropy modeling of species geographic distributions. Ecological Modelling 190, 231–259.

[19] POLLACK, S.L., HICKS, H.T., and HARRISON, W.J., Decision Tables: Theory and Practice, Wiley, New York, 1971.

[20] SANTANA, F. S, SIQUEIRA M. F., SARAIVA, A. M. & CORREA, P. L. P. 2008. A reference business process for ecological niche modeling. Ecological Informatics v. 3, n. 1, p. 75-86.

[21] SOBERON, J. and PETERSON, A. T., 2005. Interpretation of models of fundamental ecological niches and species' distributional areas. 1-10.

[22] STOCKWELL, D. R. B., BEACH, J. H., STEWART, A., VORONSTSOV, G., VIEGLAIS, D. and PEREIRA, R. S., 2006. The use of the GARP genetic algorithm and internet grid in the Lifemapper world atlas os species biodiversity. Ecological Modelling.

[23] STOCKWELL, D. R. B & PETERSON, A. T. 2002. Effects of sample size on accuracy of species distribution models. Ecological Modelling 148, 1–13.

[24] SWETS, K. A. 1988. Measuring the accuracy of diagnostic systems. Science 240, 1285–1293. WCED (World Commission on Environment and Development) 1987. Our common future. Oxford Univ. Press.

Autores

Renata Luiza Stange é mestranda em Engenharia Elétrica (Área de concentração: Computação e Sistemas Digitais) pela Escola Politécnica da Universidade de São Paulo. Especialista em Administração de Sistemas de Informação pela Universidade Federal de Lavras em Ciências e Bacharel em Análise de Sistemas pela Universidade Estadual do Centro-Oeste do Paraná.

Fabiana Soares Santana é doutoranda em Engenharia Elétrica (Área de concentração: Computação e Sistemas Digitais) pela Escola Politécnica da Universidade de São Paulo. Mestre em Ciências (Matemática Aplicada - área de concentração: Ciência da Computação) e Bacharel em Ciência da Computação pelo Instituto de Matemática e Estatística da Universidade de São Paulo. Professora do Centro Universitário da FEI e professora do Centro Universitário UNIFIEO. 17 anos de experiência docente, em instituições de renome nacional e internacional. 10 anos de experiência profissional em TI, com ocupação de cargos de liderança, incluindo a gerência de tecnologia e desenvolvimento de sistemas em instituições financeiras de médio e grande porte e a coordenação de projetos de caráter internacional.

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.

Antonio M. Saraiva é professor titular no Depto. De Engenharia de Computação e Sistemas Digitais da Escola Politécnica da Universidade de São Paulo (EPUSP), Brasil. Graduou-se em Engenharia Elétrica pela EPUSP (1980) e em Agronomia pela ESALQ-USP (1987). É Mestre (1993), Doutor (1998) e Livre-Docente (2003) pela EPUSP. Fundador (1989) e coordenador do Laboratório de Automação Agrícola (LAA) da EPUSP. Suas áreas de interesse abrangem as diversas aplicações das tecnologias da informação e comunicação ao agronegócio e ambiente. É membro da Associação Brasileira de Agroinformática (SBIAgro), da qual foi presidente (2004-2007), da Associação Brasileira de Engenharia Agrícola (SBEA) e da American Society of Agricultural and Biological Engineers (ASABE).

Um estudo sobre a Ação Elementar de Consulta no Formalismo Adaptativo

I. Chaer, R. L. A. Rocha

***Resumo* — Este trabalho discute o mecanismo de consulta especificado no Formalismo Adaptativo, levantando algumas questões que ainda não foram bem estudadas nas aplicações existentes. É proposta uma nova definição do seu comportamento, abrangente o suficiente para servir a qualquer aplicação do Formalismo sem restringir a sua capacidade e, acredita-se, sem impor novas dificuldades ao desenvolvimento. Também é delineado um método, restrito a situações nas quais o dispositivo subjacente à camada adaptativa tem poder equivalente às gramáticas regulares, capaz de satisfazer os requisitos dessa definição.**

***Palavras-chave* — dispositivo adaptativo (*adaptive device*), busca (*search*), consulta (*query*), autômato adaptativo (*adaptive automaton*).**

I. Introdução

Desde a sua definição [8], o Formalismo Adaptativo foi muito usado em tarefas de análise de linguagens como uma extensão natural dos modelos de Autômatos de Estados Finitos [2][8][9] e, mais tarde, como uma camada independente do mecanismo de processamento subjacente [10][11], consistindo a partir de então em um método de extensão do poder computacional de qualquer Dispositivo Guiado por Regras. No entanto, devido à falta de métodos de construção de modelos, as aplicações existentes foram desenvolvidas de maneira *ad-hoc*, ou seja, específica para atender os problemas individuais de cada implementação. A definição original do Formalismo tem potencial para utilização em um universo de problemas muito mais amplo do que aquele que foi explorado, e alguns pontos ainda não foram suficientemente estudados. Um dos pontos que apresentam mais evidência é o da operação de consulta – das três ações adaptativas elementares, é a que se aplica no maior número de situações, e, ao mesmo tempo, aquela com maior potencial para aumentar a complexidade computacional de uma aplicação.

Para tratar a questão adequadamente é necessário, inicialmente, expor a definição do Formalismo Adaptativo, o que é feito na seção II. A seção III traz uma exposição sobre linguagens formais, discutindo pontos relevantes para a questão da ação elementar de consulta no Formalismo Adaptativo. Na seção IV fala-se da operação de consulta no Formalismo, analisando como o problema tem sido tratado, o que foi deixado de lado – por não se ter feito necessário até então – e é proposta uma definição mais detalhada. Na Seção V é delineado um método para realizar a consulta de acordo com a definição proposta. Finalmente, a seção VI traz as considerações finais do trabalho.

Este trabalho recebeu apoio do Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) através do programa de Bolsas de Mestrado.

I. Chaer é estudante de Mestrado junto ao Laboratório de Linguagens e Técnicas Adaptativas, Escola Politécnica da USP, São Paulo/SP, Brasil; e-mail: iuri.chaer@poli.usp.br.

R. L. A. Rocha é pesquisador do Laboratório de Linguagens e Técnicas Adaptativas, Escola Politécnica da USP, São Paulo/SP, Brasil; fone: 55 11-3091-5583; e-mail: luis.rocha@poli.usp.br.

II. O Formalismo Adaptativo

O Formalismo Adaptativo consiste fundamentalmente em um mecanismo de auto-modificação que pode ser aplicado a qualquer dispositivo guiado por regras [5][10]. Esse mecanismo, definido ele mesmo como um dispositivo guiado por regras, age como uma camada superior que dá, mesmo aplicado sobre um dispositivo com capacidade equivalente a uma gramática regular, poder computacional para simular a Máquina de Turing [13].

Nas subseções seguintes será formalizado o conceito de dispositivo guiado por regra e de dispositivos adaptativos, foco de estudo deste trabalho. Elas são uma simplificação do exposto em [5] e [10] – recomenda-se aos interessados em uma exposição mais detalhada que leiam esses trabalhos.

A. Dispositivos guiados por regras

A definição de *dispositivo guiado por regras* engloba alguns dos principais formalismos usados como referência no estudo da computação, tais como a Máquina de Turing, os Autômatos de Estados Finitos e as Árvores de Decisão. São dispositivos determinados por conjuntos finitos de configurações e de regras, capazes de, quando alimentados com uma cadeia de símbolos, emitir (ou não) uma cadeia de saída e um julgamento de aceitação ou rejeição.

Formalmente, o dispositivo guiado por regras D pode ser definido como uma ênupla $D = (C, \Sigma, \Phi, c_0, C_a, R)$, onde:

- C é o conjunto de todas as configurações que D pode assumir,
- Σ é o alfabeto de entrada – o conjunto de todos os símbolos que podem ser consumidos por D (incluindo ε, o símbolo nulo),
- Φ é o alfabeto de saída de D (também incluindo ε),
- c_0 é a configuração inicial do dispositivo ($c_0 \in C$),
- C_a é o conjunto das configurações de aceitação de D ($C_a \subseteq C$)
- e R é o conjunto de regras que define as operações em D ($R \subseteq C \times \Sigma \times C \times \Phi$).

Cada regra $r_i \in R$ é, assim, dada por uma ênupla $r_i = (c_i, \sigma, c_j, \phi)$, de maneira que a cada passo o dispositivo, estando em uma configuração c_i, consome um símbolo $\sigma \in \Sigma$, passa a um estado c_j e emite um símbolo $\phi \in \Phi$. Partindo da configuração c_0 e sendo alimentado com uma cadeia de símbolos s, o dispositivo D vai consumindo os símbolos de s e mudando de configuração até chegar a um ponto onde não haja nenhuma regra de R que possa ser aplicada ou que s seja esgotada. Se

nesse momento o dispositivo estiver em uma configuração $c_i \in C_a$ e todos os símbolos de *s* houverem sido consumidos, diz-se que *s* foi *aceita* por *D*. Se o dispositivo houver terminado o processamento sem que as condições de aceitação tenham sido satisfeitas, diz-se que *s* foi *rejeitada*.

Note que a definição não limita, em nenhum momento, a aplicação de regras dependendo do símbolo nulo ε nem a existência de múltiplas regras para uma mesma configuração e símbolo consumido. Essas situações são chamadas *não-determinismos* porque, ao chegar a um ponto onde ocorra algum desses eventos, é possível prosseguir por mais de um caminho (ou seja, o progresso não está univocamente determinado). Nessas situações todas as alternativas serão processadas. Se alguma delas terminar em alguma situação de aceitação, conforme descrito anteriormente, a cadeia é considerada *aceita*.

B. Dispositivos Adaptativos

Conforme exposto no início desta seção, um dispositivo adaptativo consiste em um dispositivo guiado por regras *D* encapsulado por um mecanismo adaptativo *A*. O mecanismo *A* associa a cada regra de transição do dispositivo subjacente *D* duas operações de alteração estrutural sobre *D*, uma anterior e outra posterior à aplicação da regra, cada uma delas (ou ambas) podendo ser nula. Assim, o dispositivo adaptativo D_a pode ser definido formalmente como a dupla $D_a = (D, A)$, com *D* conforme a descrição na subseção anterior e $A \subseteq O_a \times R \times O_a$. As regras adaptativas em *A* podem agir sobre o dispositivo *D* alterando: o seu conjunto de possíveis configurações *C*, de configurações de aceitação C_a, o conjunto de regras *R*.

O conjunto de operações adaptativas O_a contém, além da operação nula, combinações das três ações adaptativas elementares:

- consulta, designada pelo símbolo ?;
- remoção, representada como -;
- e inserção, com o símbolo +.

Essas operações têm de ser aplicadas na ordem especificada acima: primeiro consultas, depois remoções e, ao final, inserções. Em [5] é proposto que a toda operação de remoção e inserção esteja associada uma consulta – o atual trabalho adota esse procedimento, centralizando nessa ação elementar os percursos pelo caminho do dispositivo. As operações de inserção e remoção têm como objeto os conjuntos *C*, *R* e *A*. Com isso, o que se tem ao exercitar um dispositivo adaptativo são, além das transições de configuração características dos dispositivos guiados por regras, transições entre diferentes dispositivos guiados por regras. A definição exata da operação de consulta está sendo propositalmente adiada – a sua caracterização em trabalhos anteriores não é suficientemente precisa para as necessidades desta pesquisa, e a discussão sobre o assunto requer uma seção exclusiva.

III. Tratamento de Linguagens Formais

Em [12] Chomsky estabeleceu as bases de uma hierarquia de linguagens definida a partir do tipo de restrições impostas pelo conjunto de regras necessárias para gerá-las. Essa hierarquia acaba por definir, como corolário, uma ordem entre os dispositivos computacionais, já que a capacidade reconhecimento de cada nível da hierarquia implica na capacidade de reconhecimento de todas aquelas que estão abaixo. Um dispositivo capaz de reconhecer linguagens do tipo 3 só é capaz de reconhecer esse tipo de linguagem. Um que reconheça linguagens do tipo 2 é também capaz de reconhecer todas as do tipo 3; e assim sucessivamente até o tipo 0, cujos aceitadores são capazes de reconhecer todos os demais tipos de linguagem.

Sendo α e β símbolos do alfabeto de uma linguagem, e **A**, **B** e **C** símbolos não-terminais usados na gramática que gera essa linguagem, pode-se representar as alternativas de regras de produção de cada gramática da seguinte maneira [1]:

- Linguagens do tipo 0 (recursivamente enumeráveis): $\alpha \rightarrow \beta$
- Linguagens do tipo 1 (sensíveis a contexto): $\alpha\mathbf{A}\beta \rightarrow \alpha\mathbf{B}\beta$
- Linguagens do tipo 2 (livres de contexto): $\mathbf{A} \rightarrow \mathbf{BC}$
- Linguagens do tipo 3 (regulares): $\mathbf{A} \rightarrow \alpha\mathbf{B}$

De maneira que linguagens recursivamente enumeráveis são geradas por gramáticas que permitem qualquer regra de produção, inclusive a troca de símbolos terminais. Linguagens sensíveis a contexto são mais restritas, permitindo, no máximo, a troca de um símbolo não-terminal dado um contexto local de símbolos terminais. Linguagens livres de contexto não permitem nenhuma produção onde seja analisado um símbolo terminal, mas permitem produções gerando mais de um símbolo não-terminal, e linguagens regulares permitem somente derivações de não-terminais a terminais e, no máximo, um não-terminal, sendo que todas as derivações da gramática resultando em algum não-terminal tem de ter os não-terminais na mesma posição em relação aos terminais gerados (não é aceitável que aconteça $\mathbf{A} \rightarrow \alpha\mathbf{B}$ e $\mathbf{B} \rightarrow \mathbf{A}\alpha$ na mesma gramática).

Dispositivos adaptativos, conforme descrito em seções anteriores, têm poder computacional equivalente à Máquina de Turing (i.e. são capazes de aceitar uma linguagem do tipo 0) mesmo quando o mecanismo adaptativo é sobreposto a um formalismo capaz apenas de lidar com linguagens do tipo 3 [13].

Tomando como referência um autômato de estados finitos como dispositivo subjacente, é fácil ver que as operações adaptativas necessárias para reconhecer uma linguagem do tipo 2 ficam restritas às transições e estados vizinhos à transição sendo ativada.

Para reconhecer a maioria das linguagens de programação – um subconjunto particularmente restrito das linguagens do tipo 1 – lida-se com situações onde é possível estabelecer um número pequeno de estruturas onde há de fato dependência de contexto (e.g. declaração de parâmetros, escopo de estruturas condicionais), e as alterações no dispositivo subjacente ficam restritas aos módulos responsáveis por elas. No entanto, para linguagens com dependências de contexto seguindo em fluxos distintos (como, por exemplo, as anáforas nas linguagens naturais, que podem ser relativas a especificadores que as precedem ou as sucedem), ou onde a própria estrutura de dependências de contexto varia. Nessas condições, é provável que a estrutura do dispositivo subjacente seja alterada radicalmente e de maneiras imprevisíveis *a priori*, inclusive de

maneira que fique impossível reconhecer (ou recompor) a estrutura original. Esse tipo de situação não foi estudada em nenhum trabalho anterior com o Formalismo Adaptativo, mas somente perseguindo os casos mais extremos é possível estabelecer soluções que valham em qualquer situação, e é certo que, dentro do conjunto infinito de linguagens dos tipos 0 e 1, o subconjunto que gera esse tipo de comportamento em dispositivos adaptativos não é nada desprezível.

As implicações do tipo da linguagem sendo processada sobre os passos computacionais de um dispositivo adaptativo têm impacto especialmente no controle do dispositivo subjacente. No modelo seguido neste trabalho, onde as operações de caminhamento no dispositivo ficam concentradas na ação adaptativa elementar de consulta, é para ela que será transferida a complexidade adicional.

IV. A Operação de Consulta

Na exposição da seção anterior, observa-se que os dispositivos adaptativos são capazes de reconhecer linguagens dos tipos 1 e 0, exceto em casos particulares, podem sofrer alterações extensas nos seus dispositivos subjacentes. As abordagens ao problema da consulta que podem ser encontradas nas aplicações existentes do formalismo adaptativo [2][5][9] seguem a descrição original em [8], onde a operação de busca recebe uma ênupla que representa a relação que define uma regra de transição adaptativa:

$$(o_{pré}, c_0, \sigma, c_1, \phi, o_{pós})$$

Sendo:

- $o_{pré} \in O_a$ a operação adaptativa anterior à transição;
- $c_0 \in C$ o estado de origem da transição;
- $\sigma \in \Sigma$ o símbolo consumido na transição;
- $c_1 \in C$ o estado de destino da transição;
- $\phi \in \Phi$ o símbolo de saída na transição e
- $o_{pós} \in O_a$ a operação adaptativa posterior à transição.

Esse modelo de consulta é bastante restrito por ser baseado nos rótulos dos estados e em relações envolvendo somente um símbolo de entrada ou saída. Com isso, ou se define rótulos com um valor semântico *a priori* que não vai poder ser alterado ao longo da execução, ou restringe-se o dispositivo suficientemente para que buscas baseadas em somente um símbolo não resultem em ambigüidades. Claramente nenhuma dessas restrições pode ser assumida em um mecanismo genérico para a implementação da ação adaptativa elementar de consulta, exceto em soluções pontuais, *ad-hoc*. Pistori toca esse problema em [5], onde propõe que a operação de consulta seja abordada como um problema de satisfação de relações entre as variáveis, e levanta a questão de refinamento dos resultados de uma busca.

A questão central para conceber um mecanismo genérico de consulta para um dispositivo guiado por regras é decidir quais parâmetros são suficientes para gerar resultados precisos, e a inclusão da camada adaptativa adiciona o requisito de que não se recorra a informações relativas à forma original do dispositivo. A resposta natural a essa questão é adicionar aos parâmetros a própria funcionalidade do setor do dispositivo que se deseja encontrar, e fazer com que as consultas funcionem recursivamente, para que os seus resultados possam ser refinados buscando sub-módulos específicos de cada módulo funcional. Por causa desse sistema de refinamento da busca, será abandonada, nessa proposta, a idéia defendida em [4][5][14] de, havendo co-referência entre as variáveis passadas nas operações de consulta, localizar somente os resultados que satisfaçam a essas relações: esse procedimento adiciona muita complexidade ao sistema e só permite a localização através de detalhes estruturais do dispositivo subjacente. A proposta é, assim, que uma chamada à operação de consulta consistiria em uma seqüência contendo no mínimo uma instância da seguinte ênupla:

$$(o_{pré}, c_0, \sigma, c_1, \phi, o_{pós}, f)$$

Sendo o último elemento, f, a função semântica que caracteriza o sub-módulo funcional a ser localizado, e os elementos anteriores e posteriores à transição passam a referenciar, então, às regiões anteriores e posteriores ao sub-módulo f. A passagem de informações na operação de consulta é por valor, e não referência, de maneira que as incógnitas devem ser caracterizadas pela passagem de um valor inválido, equivalente à constante indefinida encontrada na maioria das linguagens de programação (mas não igual ao símbolo vazio ε). Se f ficar indefinido em alguma ênupla da consulta, assume-se, naturalmente, que a intenção é localizar uma única transição, e a operação na recursão originada por essa ênupla reverte ao modo da definição original. Quando f estiver definido, o símbolo consumido σ também assume uma interpretação mais geral – sendo o interesse na consulta localizar o sub-módulo dentro do dispositivo, são retornados os símbolos consumidos em todas as transições externas ao sub-módulo que levam a ele; no caso de não haver nenhuma transição desse tipo (e.g. sub-módulos que compartilham o estado inicial do dispositivo), o símbolo assume um valor inválido. Cada ênupla da seqüência passada na invocação da operação define uma recursão, onde será feita uma nova busca restrita às sub-máquinas localizadas na recursão anterior. O algoritmo de consulta mais simples poderia ser implementado, em pseudocódigo, da seguinte maneira:

```
consulta(padrões, dispositivoSubjacente, índice)
       dispositivosLocais = [];
       foreach sub-módulo in dispositivoSubjacente
              if satisfaz(sub-módulo,
padrões[índice])
                     push(dispositivosLocais, sub-
módulo)
       índice = índice + 1
       if índice >= length(padrões)
              return dispositivosLocais
       else
              dispositivos = []
    foreach sub-módulo in dispositivosLocais
                     push(dispositivos,
           consulta(padrões, sub-módulo, índice))
       return dispositivos
```

```
satisfaz(padrão, disp)
        resultados = []
        if              ( (not defined padrão.f and
          not defined disp.f) or
          disp.f == padrão.f) and
       indefOuIgual(padrão.o_pré, disp.o_pré) and
       indefOuIgual(padrão.c0, disp.c0) and
       indefOuIgual(padrão.σ, disp.σ) and
       indefOuIgual(padrão.c1, disp.c1) and
       indefOuIgual(padrão.□, disp.□) and
       indefOuIgual(padrão.o_pós, disp.o_pós)
               return true
        else
               return false


indefOuIgual(a, b)
        if not defined(a) or a==b
               return true
        else
               return false
```

Dessa maneira, recorrendo a vetores para tratar as multiplicidades de resultados e parâmetros na recursão, o que se faz é verificar cada sub-módulo do dispositivo em busca daqueles capazes de satisfazer à seqüência de padrões definida. O algoritmo dado testa exaustivamente todas as possibilidades, e sem dúvida pode ser melhorado. A obtenção dos sub-módulos que compõem um dispositivo e a determinação da função semântica deles dependem do formalismo subjacente sendo usado no dispositivo adaptativo. Note que as próprias regras do dispositivo são tratadas como sub-máquinas atômicas sem função semântica marcada – elas são definidas univocamente pelos seus outros parâmetros –, e que a reversão para a definição original ocorre naturalmente no algoritmo.

É importante, entretanto, manter em mente que essa proposta, apesar de ser capaz de aumentar muito o poder da ação de consulta, implica em um grande aumento na complexidade computacional da sua execução. Isso porque, tomando como referência de dispositivo subjacente o autômato de estados finitos, mesmo sendo definida uma forma normal biunívoca para definir a relação entre autômatos e funções sintáticas (i.e., somente um autômato implementando cada função e somente uma função sendo implementada por cada autômato), no pior caso haverá recursões equivalentes ao problema de isomorfismo de sub-grafos: um problema conhecidamente NP Completo.

Uma vez proposta a interface de um mecanismo de consulta que pode ser aplicado a um método genérico de construção de dispositivos adaptativos, dada a observação já feita quanto à complexidade da sua computação, é importante discutir o seu tratamento. É condição *sine qua non*, uma vez que respostas em tempo infinito não são suficientes para o modelo adaptativo, que seja estabelecida uma forma normalizada para a descrição da semântica subjacente às possíveis sub-módulos do dispositivo. Tendo essa forma normalizada, é possível em um número de passos $O(2^n)$ realizar qualquer consulta por comparação exaustiva; $O(2^{n \cdot n})$ considerando as recursões do procedimento proposto. No entanto, restringindo o dispositivo guiado por regras subjacente do dispositivo adaptativo, há métodos de busca com custo em tempo muito melhor na maioria dos casos. Na seção seguinte será proposto um método para executar consultas em autômatos de estados finitos adaptativos, método esse que certamente pode ser estendido a qualquer dispositivo adaptativo cujo formalismo subjacente tenha poder de computação equivalente ao das gramáticas regulares.

V. Um Método de Consulta para Autômatos de Estados Finitos Adaptativos

O autômato de estados finitos (AF) é um formalismo extremamente atraente como camada subjacente de um dispositivo adaptativo devido à sua simplicidade: sendo capaz de reconhecer somente linguagens regulares, é normalmente bastante simples projetar e compreender o funcionamento de um autômato desse tipo. O motivo pelo qual ele foi escolhido para este trabalho, no entanto, é o ferramental disponível para o seu tratamento. Em [5], Hopcroft apresenta um algoritmo com complexidade assintótica $O(n \cdot log(n))$ para minimizar um autômato de estados finitos determinístico (AFD). O AFD mínimo é uma descrição única para uma linguagem regular [1][6], e, por isso, um ótimo candidato para o elemento ***f*** requerido para os parâmetros de consulta definidos na seção anterior.

Árvores e AFDs são muito semelhantes estruturalmente. Estabelecendo uma relação de ordem para o alfabeto Σ reconhecido por um AFD é possível usar os algoritmos de caminhamento pré-ordem e pós-ordem exigindo somente uma pequena modificação para tratamento de ciclos (o caminhamento precisa ser interrompido sempre que um for encontrado). Isso é um fato particularmente interessante porque, uma vez estabelecida a relação de ordem e a forma de caminhamento, pode-se então caracterizar o autômato através de uma cadeia de comprimento $O(n+m)$ símbolos, sendo n o número de estados e m o número de regras de transição do AFD.

Sendo o AFD mínimo uma descrição única para um AF e havendo a possibilidade de descrevê-lo com uma simples cadeia de caracteres, existe a possibilidade de indexar a semântica inerente a cada sub-módulo de um AF em estruturas onde a busca pode ser executada com custo extremamente baixo, tais como *hashes* e árvores binárias, reduzindo o custo de cada recursão em uma consulta do pior caso mencionado anteriormente, $O(2^n)$, a $O(1)$ ou $O(log(n))$. No entanto, há um custo colateral: é necessário gerar, para cada sub-módulo possível do dispositivo subjacente, as classes de equivalência da linguagem que ele reconhece, a um custo $O(2^n) \cdot (O(n \cdot log(n)) + O(n))$ – ou seja, $O(2^n)$. Incorre-se nesse custo no momento da indexação, e, a cada atualização do índice, em um custo de $\Omega(n \cdot log(n))$ e $O(2^n)$, dependendo da influência da alteração sobre o dispositivo como um todo. Para suprir a questão da recursão é necessário que o índice construído relacione, para cada classe de equivalência, todos os sub-módulos funcionais contidos no módulo responsável pela classe em questão. Isso não é particularmente custoso ou complicado uma vez que todos esses módulos têm mesmo que

ser gerados *bottom-up*, mas essa indexação resulta também em um custo em espaço polinomial em relação ao número de regras no dispositivo subjacente. Caso não seja possível garantir que o AF subjacente é determinístico, haverá necessidade de converter cada sub-módulo no seu análogo determinístico mínimo, uma tarefa executada em $O(2^n)$ passos com o algoritmo apresentado em [3] e [7] que terá de ser repetida para cada sub-módulo indexado (inclusive podendo aumentar o número e de estados de cada sub-módulo a 2^e), motivo pelo qual é extremamente recomendável usar somente AFDs.

A complexidade apresentada pelo método proposto decai a um custo inferior ao do pior caso apresentado na seção IV. No entanto, isso não invalida o método como uma maneira de reduzir o custo real do procedimento. Note que a geração de um novo índice de sub-módulos caracterizados pelos seus AFDs mínimos só ocorre quando há mudanças no dispositivo subjacente, e, mais que isso, as atualizações na maioria das situações possíveis não chegará ao pior caso. O custo, em uma aplicação que execute mais consultas do que inserções e remoções, será certamente menor do que se fosse usado o algoritmo exaustivo, já que o custo de cada re-indexação será amortizado entre todas as consultas. Considerando que, no modelo assumido neste trabalho, é necessária uma consulta para a execução de cada inserção ou remoção, dada uma aplicação genérica do formalismo é extremamente provável que o procedimento descrito resulte em ganhos no tempo de processamento. Além disso, se o conceito de consultas pela semântica for adotado completamente (i.e., os rótulos dos estados forem ignorados em todas as operações de consulta), é possível abstrair a estrutura real do dispositivo subjacente, criando a possibilidade de realizar nele operações de otimização (e.g. remoção de redundâncias) sem prejuízo; potencialmente melhorando o desempenho tanto do sistema de indexação para consulta quanto do uso do autômato para reconhecer cadeias.

Para exemplificar o funcionamento do método proposto, será usado o AF não-determinístico da figura 1:

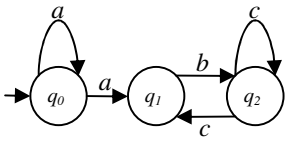


Fig. 1. Exemplo de autômato de estados finitos não-determinístico

Usando o método proposto nesta seção, é necessário, em primeiro lugar, determinar o AFD mínimo de cada sub-módulo do AF apresentado. Na figura 2 são mostradas as versões determinísticas mínimas dos sub-módulos contendo estados ligados por transições iniciados em estados com transições de saída:

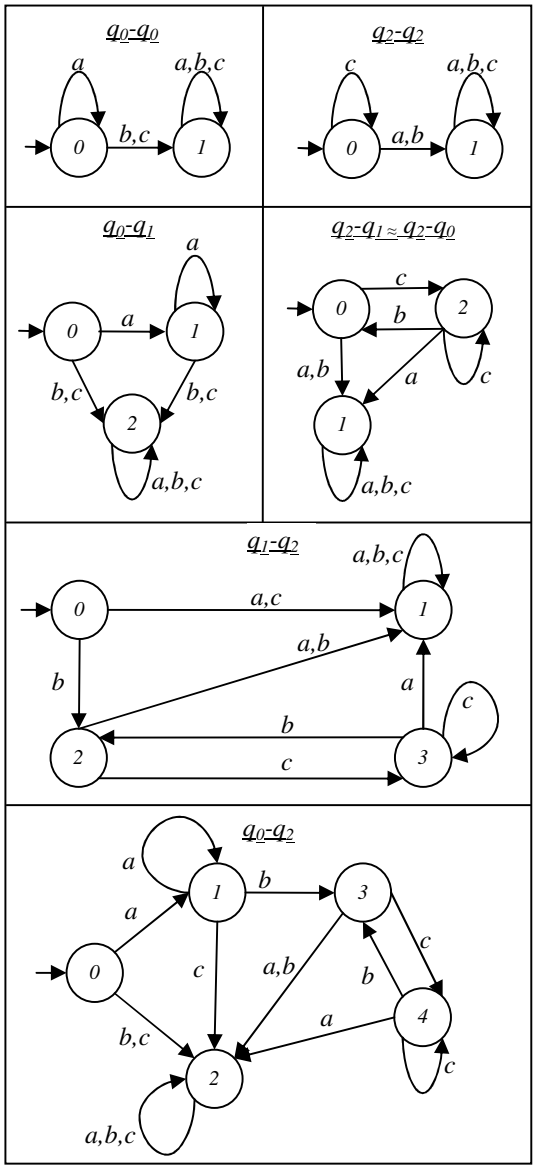


Fig. 2. Sub-módulos minimizados do AF não-determinístico da figura 1

Nesse exemplo, os estados já foram rotulados de acordo com a ordem de visitação seguindo o algoritmo pré-ordem assumindo que $a<b<c$. Dessa maneira, o índice para consultas teria a seguinte aparência (omitindo as operações adaptativas e o símbolo de saída, dado que não foram definidos estados de aceitação ou mecanismo adaptativo):

TABELA I
ÍNDICE PARA BUSCAS NO AF DA FIGURA 1

c_0	σ	c_1	f
q_0	*a*	q_0	*indef*
q_0	*a*	q_1	*indef*
q_1	*b*	q_2	*indef*
q_2	*c*	q_1	*indef*
q_2	*c*	q_2	*indef*
q_0	-	q_0	*011111*
q_2	*b*	q_2	*110111*
q_0	-	q_1	*122122222*
q_2	-	q_1	*112102222*
q_2	-	q_0	*112102222*
q_1	*a*	q_2	*121111113123*
q_0	-	q_2	*122132222114234*

Cada número mostrado na coluna f representa uma transição consumindo o símbolo que viria naquela ordem – por exemplo, na sexta linha, *011111* significa que, do estado *0*, alcança-se o estado *0* consumindo *a*, *1* consumindo *b* e *1* consumindo *c*; do estado *1*, atinge-se *1* consumindo *a*, *b* ou *c*.

VI. CONCLUSÃO

Neste trabalho foi discutida a ação adaptativa elementar de consulta. Inicialmente foi exposto o seu papel dentro do Formalismo Adaptativo, ressaltando sua importância para o funcionamento de um dispositivo adaptativo e o universo restrito de aplicações que lhe foram destinadas até o momento.

oi mostrado que existem situações para as quais a definição da ação adaptativa elementar de consulta que foi usada nos trabalhos existentes é insuficiente, e propostos novos parâmetros para que essa ação possa ser executada em aplicações genéricas do Formalismo. Foram discutidas algumas questões sobre as dificuldades de implementar a ação da consulta satisfazendo os requisitos da nova proposta, em especial a complexidade da sua computação, exponencial no tempo em relação ao número de configurações do dispositivo subjacente.

Finalmente, foi delineado um algoritmo para mitigar o custo de uma operação genérica como a proposta, e discutido, *grosso modo*, o seu custo computacional. Acredita-se que a proposta feita tem uma relação custo-benefício favorável, mas não foram desenvolvidas aplicações reais para demonstrar a validade dessa conjectura. Isso é deixado como tarefa para trabalhos futuros – tanto em experimentos baseados na reprodução de aplicações já existentes quanto verificando o comportamento da proposta em linguagens cujas gramáticas exijam alterações extensas no dispositivo adaptativo que as reconhece.

REFERÊNCIAS

[1] A. Salomaa. *Formal Languages.* Academic Press, 1973.

[2] D. P. Shibata. "Tradução Grafema-Fonema para a Língua Portuguesa Baseada em Autômatos Adaptativos", Dissertação de Mestrado, USP, São Paulo, 2008.

[3] H. Lewis e C. Papadimitriu, *Elements of the Theory of Computation.* Prentice-Hall, 1998.

[4] H. Pistori e J. J. Neto, "AdapTools: Aspectos de Implementação e Utilização," *Boletim Técnico PCS*, Escola Politécnica, São Paulo, 2003.

[5] H. Pistori, "Tecnologia Adaptativa em Engenharia de Computacão: Estado da Arte e Aplicações," Tese de Doutorado, USP, São Paulo, 2003.

[6] J. E. Hopcroft. "An nlogn algorithm for minimizing the states in a finite automaton." in The Theory of Machines and Computations, ed. Z Kohave. New York, Academic Press, 1971.

[7] J. E. Hopcroft, R. Motwani, J. D. Ullman. *Introduction to automata theory, languages, and computation*, 2nd. ed. Ed. Addison-Wesley, 2001.

[8] J. J. Neto, "Contribuições à metodologia de construção de compiladores", Tese de Livre Docência, USP, São Paulo, 1993.

[9] J. J. Neto, "Adaptive Automata for Context-Sensitive Languages," *ACM SIGPLAN NOTICES*, Vol. 29, n. 9, pp. 115-124, September, 1994.

[10] J. J. Neto e M. Moraes, "Formalismo adaptativo aplicado ao reconhecimento de linguagem natural," em *Anais da Conferencia Iberoamericana en Sistemas, Cibernética e Informática*, 19-21 de Julho, 2002, Orlando, Florida, EUA.

[11] M. K. Iwai, "Um formalismo gramatical adaptativo para linguagens dependentes de contexto," Tese de Doutorado, USP, São Paulo, 2000.

[12] N. Chomsky, "Three models for the description of language," IEEE Transactions on Information Theory, Vol. 2, n. 3, pp. 113-124, 1956.

[13] R. L. Rocha, "Um método de escolha automática de soluções usando tecnologia adaptativa.", Tese de Doutorado, USP, São Paulo, 2000.

[14] R. L. Rocha, "A Structural Operational Semantics Approach to Adaptive Devices", em *Anais do Congresso de Lógica Aplicada à Tecnologia*, 21-23 de Novembro, 2007, São Paulo.

Iúri Chaer nasceu na capital de São Paulo, Brasil, a três de Março de 1981. Formou-se Engenheiro Eletricista pela Escola Politécnica da USP ao final de 2003.

Trabalhou por cinco anos no setor privado em sistemas computacionais para tratamento de grandes massas de dados, e, posteriormente, participou do primeiro Programa de Residência em Tecnologia oferecido na Universidade de Minas Gerais, em 2007. Atualmente pesquisa Análise Semântica de Linguagens Naturais e Mecanismos de Inferência.

Ricardo Luis A. Rocha é natural do Rio de Janeiro-RJ e nasceu em 29/05/1960. Graduou-se em Engenharia Elétrica modalidade Eletrônica na PUC-RJ, em 1982. É Mestre e Doutor em Engenharia de Computação pela EPUSP (1995 e 2000, respectivamente). Suas áreas de atuação incluem Tecnologias Adaptativas, Fundamentos de Computação e Modelos Computacionais.

Dr. Rocha é membro da ACM (Association for Computing Machinery) e da SBC (Sociedade Brasileira de Computação).

Mesa Redonda

***Resumo* — Esta Mesa Redonda, que encerrou os trabalhos do WTA 2008, foi presidida pelo Professor Dr. Ricardo Luis de Azevedo da Rocha (EPUSP) e contou com a participação de quatro convidados: Professora Claudia Zapata del Río (PUC-Peru), Professor Sérgio Donizetti Zorzo (UFSCar), Professor Ítalo Santiago Vega (PUC-SP), e Professor Marcus Vinícius Midena Ramos (UVSF). Em uma primeira etapa, os membros da Mesa Redonda expuseram suas visões e experiências com foco na teoria da computação e sua vinculação à tecnologia adaptativa, e na segunda, apresentaram sugestões, para maior aproximação entre os interessados na área e para a motivação e viabilização de iniciativas em Tecnologia Adaptativa.**

I. INSTALAÇÃO DA MESA REDONDA

Na segunda metade da tarde do último dia do evento, ocorreu a mesa redonda, cujas falas estão transcritas a seguir.

Visto que parte das falas foi em espanhol, foi feita para estas falas uma tradução livre que procurou respeitar ao máximo o estilo original.

Levando em conta também que a linguagem falada e a escrita diferem em diversos pontos, foram feitas as adaptações prosódicas adequadas, com o cuidado de manter fielmente o conteúdo, garantindo dessa maneira a obtenção de uma legibilidade melhor, e maior naturalidade para o texto escrito.

Presidiu a mesa o professor Ricardo Luis de Azevedo da Rocha (identificado abaixo como Ricardo).

Participaram como convidados quatro professores universitários:

o professor Sérgio Donizetti Zorzo, da Universidade Federal de São Carlos - SP (identificado abaixo como Zorzo);

a professora Claudia Zapata del Río, da Pontificia Universidad Católica del Perú - Lima (identificada abaixo como Claudia);

o professor Ítalo Santiago Vega, da Pontifícia Universidade Católica de São Paulo - SP (identificado abaixo como Ítalo);

o professor Marcus Vinicius Midena Ramos, da Universidade do Vale do São Francisco - PE (identificado abaixo como Marcus);

Encerrou a sessão o prof. João José Neto, da Universidade de São Paulo - SP (identificado abaixo como João);

A seguir, apresenta-se uma transcrição livre do pronunciamento dos membros da Mesa Redonda, bem como dos questionamentos apresentados pela platéia e das respostas dadas pelos membros da Mesa.

II. TRANSCRIÇÃO DA MESA REDONDA

Ricardo:

O tema desta mesa é "**Importância dos conhecimentos científicos na tecnologia adaptativa**".

A Tecnologia Adaptativa é fruto da teoria da computação. Portanto como em todas as aplicações que vimos ontem e hoje, ela repassa para as diversas áreas do conhecimento não apenas a computação em si.

Vimos aplicações inclusive em áreas como gerenciamento (tabelas de decisão adaptativas, árvores de decisão adaptativas)

Obviamente vamos ter de abordar alguns aspectos como, por exemplo, a questão educacional, de formação desses profissionais que irão trabalhar com Tecnologia Adaptativa.

Chamarei então para compor essa mesa redonda

O professor Sérgio Donizetti Zorzo da Universidade Federal de São Carlos

A professora Cláudia Zapata da Pontifícia Universidad Católica del Perú

O prof. Ítalo Santiago Vega, da Pontificia Universidade Católica de São Paulo e

O prof. Marcus Vinicius Midena Ramos, da Universidade do Vale do São Francisco.

Vamos começar e cada um dos componentes da mesa terá 5 minutos para expor o seu ponto de vista, sua visão do problema e depois cada um deles terá mais alguns minutos para responder a algumas perguntas.

Nessa primeira rodada, apenas a título de guia, pediria que cada componente da mesa falasse um pouco sobre a formação desse profissional, colocando sua opinião, sua visão e experiência como educador e profissional. Também o que cada um considera como pontos importantes que devem fazer parte da formação profissional, e que deveriam constar na estruturação curricular do seu curso de formação.

Vamos ouvir em primeiro lugar o prof. Sérgio Donizetti Zorzo, da UFSCar.

Zorzo:

Obrigado, Ricardo.

Há algumas considerações que gostaria de fazer, mas vou tentar colocar as coisas conforme foram solicitadas.

Tenho 30 anos de carreira de professor, e sempre fui responsável em minha universidade por uma disciplina chamada linguagens formais e autômatos, ou aspectos formais da computação, mas que trata da concepção dos aspectos de linguagens formais e sua relação com autômatos.

O que eu tenho para falar desses 30 anos? Para situar vocês, na nossa universidade temos dois cursos de graduação, um de engenharia de computação e outro de bacharelado em ciências da computação. Essa disciplina é oferecida para os dois cursos. da mesma maneira, os alunos são misturados.

Uma concorrência acontece no vestibular, quando de 13 alunos que entram para ciência da computação e por volta de 30 alunos de engenharia de computação. Dando a entender que é um grupo selecionado de alunos que entram.

Nesses 30 anos, vou falar como seguiu esta disciplina. A primeira vez que eu lecionei em 1979, adotei o livro do Hopcroft/Ullman de 1969.

Esse livro era o livro de referência do currículo da ACM, e seu texto era em inglês, que depois foi substituído em 1979 por outro texto mais novo dos mesmos autores, e seguimos com este livro a partir de então. Usei o livro do Papadimitriou,

e outros livros, mas acabei voltando para este. Em 2001 surgiu uma nova versão desse mesmo livro, e essa versão merece destaque. É uma versão que foi traduzida para o português, mas a gente percebe que ela está mais empobrecida dos aspectos teóricos, mais empobrecida nos aspectos de ciência da computação.

Demorou a perceber que na versão antiga aparecia um teorema seguido de demonstração. O segundo livro é mais a técnica.

Deu para notar o seguinte: enquanto eu exigia dos alunos que aprendessem a demonstração dos teoremas, eles tinham uma dificuldade imensa -- até um determinado ponto -- quando eu pude detectar que no ensino médio não se dava mais a formação que eu tive, e com a qual eu estava acostumado.

Tenho outra experiência com um livro texto escrito em português, por um colega nosso, da UFRGS e que é adotado por uma significativa parcela de faculdades, na maior parte particulares.

No semestre passado, tive duas turmas dessa disciplina de linguagens formais e autômatos, praticamente 95 alunos ao todo, e eu adotei pela primeira vez esse livro. Foram comprados diversos exemplares para a biblioteca, e eu usei o material desses professores. O que constatei? Sofri muito para seguir transparências e material didático de outros, para adaptar tudo aquilo. Constatei que o livro trata dos assuntos como o livro do Hopcroft trata, só que de uma forma bem mais superficial que os demais: à medida que você tenta cobrar dos alunos algo mais profundo, o livro deixa a desejar, e aí surge uma insatisfação dos alunos. Parece que em outras faculdades, só são apresentados os aspectos técnicos da tecnologia. Vamos construir um autômato? Autômatos se constroem dessa forma. Exemplifica-se dessa forma. Não tem aquela concepção teórica.

Resolvi que não vou mais voltar a adotar esse livro como livro texto.

Nesses 30 anos, percebi um empobrecimento do material didático, e um desequilíbrio da matéria, pela priorização dos aspectos técnicos apenas do assunto em detrimento dos aspectos de ciência.

Quando chega a um trabalho científico, em que se exige a ciência, aí é que surgem os grandes problemas, e vou relatar outro lado:

Sou formado em um curso de ciência de computação e tive uma formação nos aspectos científicos da matéria com o professor Paulo Velloso da PUC-RJ e aí a gente percebe qual é a visão do matemático em relação à computação, e qual é a do engenheiro. E nós estamos nesse meio de caminho.

Sou professor e orientador de mestrado. (vou colocar em público para vocês terem uma idéia). Submetemos um trabalho cientifico com certa qualidade, a um evento internacional "A", e que foi recusado por uma série de razões.

Um dos elementos do comitê de programa me escreveu convidando para submeter a outro evento Qualis "A" que iria acontecer algum tempo depois. Submetemos esse trabalho, e aí veio uma série de críticas, e foi recusado. Aí eu escrevi para essa pessoa, que se chama Massami Ito, do Japão, perguntando o que havia acontecido: se ele estava me convidando para submeter o trabalho a esse evento (é claro que não era um convite sem avaliação), e agora vinham essas avaliações dessas pessoas. Aí ele se desculpou, e disse que era a visão particular de cada um. Ele tinha razão: a resposta dele foi clara. O que a gente viu nesse evento é muito do uso da tecnologia. A gente viu também fundamentação teórica. Muita tecnologia que está aqui, a forma como ela é apresentada para quem ela é apresentada, tem um retorno para a gente. O retorno que a gente vai receber vai ser para quem a gente apresenta.

Outra coisa que eu queria falar é que este workshop é o terceiro que acontece, eu participei do segundo, não estive presente, mas tive a participação de alunos meus aqui no primeiro, e o que eu percebi foi uma melhora significativa de qualidade dos trabalhos, pelo que eu parabenizo o prof. João José como *chair* deste evento, e também a todos os participantes, só que isso não é ainda suficiente para a gente.

Como profissionais de universidade, somos cobrados por órgãos governamentais quanto aos locais onde nós publicamos nossos artigos. Tenho certeza de que este evento, na minha pessoa e nos alunos que estão subordinados a mim, tem muito mais valor que a participação em um evento científico em que ninguém conversa com você, não tem participação do público, aí você tem gasto financeiro de viajar para o exterior, gasto de tempo, e aprende pouco.

Nesses dois dias que estive aqui, e nos dias que estive no ano passado eu ganhei muito, e foi com satisfação que a gente participa nesses dois dias. Só que isso tem que ser refletido nos órgãos que nos avaliam.

A capes tem que saber que isso existe, e que os trabalhos aqui publicados têm que ser reconhecidos como trabalhos publicados num simpósio IFIP, ACM ou IEEE.

Acho que deve ser feito um esforço, isso não pode ficar só com o João José, mas devemos todos tomar para nós a necessidade de fazermos como que esse evento tenha o mesmo valor que uma publicação Qualis A em outro evento.

Ricardo:
Obrigado professor Zorzo. Peço agora à professora Cláudia para que se pronuncie.

Cláudia:
Obrigada.
Eu não falo português.
Por isso vou tratar de falar pausadamente para que possam me compreender.

Venho há cinco anos lecionando na Universidade Católica do Peru, basicamente no primeiro semestre, e nos últimos semestres oriento trabalhos de formatura.

Tenho contato, no primeiro semestre, com alunos de diversas modalidades de engenharia, e de diversos níveis, pois dou aulas para todas as modalidades.

Temos um modelo pedagógico no qual, nos dois primeiros anos, são lecionadas disciplinas gerais, abrangendo matemática, física, química, e nos três anos seguintes entramos nas disciplinas específicas das diversas modalidades.

Há uns dez anos, os cursos de caráter científicos eram muito exigentes e difíceis em nossa universidade.

Muitas pessoas que estudavam engenharia reclamavam, alegando que era um exagero forçar um engenheiro a estudar uma matemática tão avançada, e em resposta, foi feita uma mudança no plano.

Isso foi péssimo para nós que nos interessamos por computação, porque a computação se baseia muitíssimo na matemática, e uma matemática muito profunda e analítica.

Dessa maneira, alunos de engenharia que atualmente elaboram alguma pesquisa um pouco mais profunda têm sofrido com seus muitos vazios conceituais, os quais lhes dificultam, por exemplo, entender teoremas ou projetar regras básicas de uma linguagem.

Procuramos complementar isso com os alunos já engajados em seus trabalhos de formatura, levando-os, por exemplo, a ler um pouco sobre assuntos teóricos, mas é difícil mudar esses dois primeiros anos, porque se trata de uma formação básica ministrada a todas as engenharias: civil, eletrônica, e informática (correspondente ao que vocês chamam aqui de engenharia de computação).

Nas universidades peruanas, os currículos estão fortemente marcados pelas necessidades do mercado. No Peru há muito desenvolvimento de software. Não o desenvolvimento de novas tecnologias, e sim o desenvolvimento de sistemas de informação.

Isso faz com que nossos alunos comecem a trabalhar basicamente no quarto ano de estudos, e temos poucos alunos interessados em se aprofundar em temas de ciências da computação.

Em decorrência, nós temos muitos alunos dedicados a sistemas de informação, à produção industrial, e muita dificuldade de conseguir pessoas que se interessem mais por assuntos científicos. Talvez porque sejam assuntos pouco atraentes em matéria de dinheiro.

E se vocês dizem que aqui se investe pouco em pesquisa, no Peru se investe menos ainda.

É muito crítico esse investimento. Ele é particularmente maior nas universidades privadas, onde se pode conseguir um pouco mais de apoio para a pesquisa.

O forte de nossa carreira, as linhas básicas estão mais voltadas a sistemas de informação e a tecnologias de informação.

No caso das ciências da computação, há os primeiros anos especializantes, basicamente com disciplinas de algoritmos e linguagens de programação.

Reconhecemos atualmente muito mais a importância dos assuntos teóricos, cuja insuficiência, como mencionava o professor que me precedeu, impede-nos de moldar melhores pesquisas, ou de aplicar os temas de uma forma mais apropriada.

Muitos pensam que o engenheiro não precisa de muita formação teórica, quando, na realidade, ele necessita compreender como toda a teoria fundamental está construída para que possa criar inovações que sejam realmente importantes.

Há muita teoria básica que não tem sido utilizada, mas que pode facilitar muito a aplicação da tecnologia.

Um exemplo claro: a tecnologia adaptativa. É muito trabalhoso convencer as pessoas em nossa universidade das facilidades que nos proporciona a tecnologia adaptativa. Quando tentamos atacar alguns problemas, muito embora o uso da tecnologia adaptativa possa tornar as coisas bem mais simples, a despeito disso, as pessoas continuam usando algoritmos muito mais complexos em seu lugar. Porque, bem,... por nunca tê-la conhecido.

Há pouco investimento em pesquisa e poucos alunos, mas às vezes surgem bons alunos, interessados em pesquisar por conta própria.

Recentemente, o governo esteve concedendo bolsas este ano para a área de ciências da computação.

Este ano, por fim, houve interesse em fomentar o estudo das ciências da computação.

O programa de mestrado que oferecíamos anteriormente acabou sendo dividido em dois: em engenharia de software e em ciências da computação.

E aqui entra o ponto forte: a universidade optou por se alinhar a esse convite do governo para que fortaleçamos a área das ciências da computação.

Ricardo:

Obrigado professora Cláudia. Passo agora a palavra ao professor Marcus.

Marcus:

Eu comecei a dar aulas em ensino superior em 1992, e foi justamente na área de linguagens formais e autômatos. De lá para cá virou assim uma paixão, uma área que eu abracei, e ao longo desses 16 anos, não contínuos, com algumas interrupções, mas 16 anos de experiência, eu compartilho muito as opiniões do prof. Zorzo. Inclusive sobre o livro texto. Para esta mesa redonda eu achei interessante trazer uma perspectiva diferente para o debate, e eu preparei alguns slides que eu queria apresentar.

Quando eu conversei com o João sobre o que seria a mesa redonda, e ele me antecipou a idéia da importância dos fundamentos teóricos, da computação e falar dos aspectos científicos da adaptatividade nada mais é que falar de necessidade de se ter uma atividade regular e permanente em pesquisa na teoria da computação. Adaptatividade vista sob a ótica da teoria da computação é o que foi apresentado aqui.

Então fiz o seguinte, eu disse: no Brasil a gente percebe um movimento não generalizado, mas claro, de se minimizar a importância do ensino da teoria da computação em cursos superiores, em favor de se ministrar disciplinas de caráter mais prático, tecnológico, e assim por diante.

Então procurei rastrear a origem disso e se a gente vive esse problema, de onde será que ele veio? Fiz uma pesquisa sobre o assunto e descobri uns relatórios que foram publicados nos Estados Unidos na década de 1990. Descobri (isto é uma hipótese minha, não sei se verdadeira ou não), e me ficou a impressão de que o que eu encontrei nesse material pode justificar o que a gente sente e constata no Brasil nos dias de hoje.

O material que eu pesquisei foram esses artigos aqui...

São alguns artigos falando sobre a importância de você ter uma pesquisa ativa e continuada em teoria da computação.

Aqueles artigos em verde ali são de um grupo de pesquisa americano e os mais importantes são o Aho e o Papadimitriou. Eles fizeram um relatório no qual defendem uma posição. Os outros 3 artigos sublinhados já foram uma resposta aos dois originais: já houve uma polemização. A partir dessa polêmica que se instalou em 1996 nos Estados Unidos, portanto faz 12 anos (mas para nós é muito atual, e talvez coincida em termos de data, com este movimento que a gente começou a perceber aqui no Brasil). O que trazem esses artigos, de uma forma

geral? Todos eles, de uma forma geral, consideram significativos os resultados obtidos até agora, mas acham que os desafios do presente são ainda maiores.

Especialmente, eles propõem uma aproximação maior entre os teóricos da computação, ou seja, os pesquisadores da teoria da computação e a pesquisa aplicada, voltada ao desenvolvimento de produtos, de tecnologias e assim por diante. Tanto o Aho, Papadimitriou e outros, propõem que haja um isolamento menor das pessoas que compõem a comunidade da teoria da computação. E foi justamente esse o ponto da polêmica.

Mas antes de chegar à polêmica, eles relacionam umas conclusões importantes. Não sobrou nenhuma dúvida quanto à importância de se ter uma pesquisa profunda em teoria da computação. Eu achei interessante nesse relatório Pitac, que é de 1998, que é um comitê de TI que assessora o presidente americano.

Há um parágrafo lá que diz assim: que a agenda federal para a pesquisa e desenvolvimento de TI caminhou longe demais na direção de desenvolvimentos e aplicações de curto prazo, e que eles precisariam de alguma forma comprometer pesquisas de longo prazo e de alto risco, de importância fundamental. O próprio comitê que assessora o presidente americano percebeu a necessidade de se trabalhar com pesquisa de longo prazo e alto risco.

Qual foi o foco da polêmica? O próprio Aho e Papadimitriou dizem assim: havia uma saturação naquela época do mercado de trabalho nos Estados Unidos na área acadêmica para pesquisadores em teoria da computação. Havia também limites de orçamento, daquelas agências de fomento e de patrocínio de pesquisa para a área de teoria da computação. Segundo eles havia um senso de desconexão entre a área de teoria da computação e outras áreas da computação, como tecnologia, aplicações e assim por diante.

Então eles propõem um direcionamento da área no sentido de manterem o país deles líder na área, e esse redirecionamento que eles propõem é justamente no sentido de se ter uma maior aproximação dos acadêmicos da teoria da computação com o restante da comunidade, para aumentar o impacto dos resultados teóricos na parte prática. E uma maior prioridade para as pesquisas, estabelecendo conexões entre teoria e prática.

Disposição de alunos de pós graduação, a parte de pesquisa aplicada e assim por diante. Autocrítica dos pesquisadores, etc., todo o movimento para aproximar a teoria da prática.

Qual foi então a reação? Esses outros dois pesquisadores de Israel argumentaram justamente o contrário. Se há problemas na área, a solução não é fazer com que a área teórica deixe de ser eminentemente teórica e passe a pender mais para o lado das aplicações e práticas, produtos e serviços.

Se a teoria da computação teve e tem até hoje um tão grande impacto nos produtos e serviços, é justamente porque esta área conseguiu até agora se manter independente, e, portanto imune às influências de momento, as influências tecnológicas. Rebatem, portanto os argumentos dos autores anteriores.

Dizem eles que esse movimento existe por pressões externas, frustrações até da própria comunidade da teoria da computação, principalmente porque não conseguiram produzir os resultados que se imaginava seriam possíveis de obter ali atrás, e também uma questão de liderança, que tange a esfera política.

Qual é a conclusão que eu tirei disso tudo?

Em momento algum nesse debate todo se viu qualquer questionamento quanto à importância de se ter uma pesquisa profunda, continuada e ampliada dos aspectos teóricos da computação. Isso nunca foi colocada em questão. O que veio à tona foi a questão da aproximação da comunidade teórica dos problemas tecnológicos e da prática. Só. Minha tese é que a gente constata uma pressão em algumas escolas, em alguns departamentos, por uma redução de conteúdos nas disciplinas na parte teórica, e aí os prejuízos são evidentes. É até chover no molhado discutir o quanto é ruim se admitir isso como um caminho a ser seguido.

Mas aí fiquei pensando: como normalmente muito daquilo que se experimenta por aqui tem suas origens lá fora, especialmente nos Estados Unidos, será que este movimento que aconteceu aqui não foi simplesmente um erro de interpretação dessa polêmica que houve por lá em torno desses artigos? Será que quando se debateu por lá a aproximação entre as comunidades teórica e de aplicação (experimental) será que isso não foi importado de uma maneira errônea para cá?

Será que a interpretação que se deu não foi "Puxa vida, temos que abandonar, ou amenizar a carga da teoria, e enfatizar as disciplinas de caráter mais prático", e então minha hipótese - só uma hipótese - é de que talvez tenha sido esta a origem deste movimento que a gente vive. Se isso é verdade, estamos agora colocando os pingos nos i's e corrigindo os erros, Este suposto desvio de um ensino sério, profundo e consistente da teoria da computação jamais foi questionado em lugar algum. Portanto, muito menos deveria estar sendo questionado aqui. Enfim, foi só uma luz que eu quis trazer, para dar uma perspectiva diferente para o debate.

Ricardo:

Obrigado professor Marcus. Vamos continuar com o professor Ítalo.

Ítalo:

Boa tarde a todos. Sou o Ítalo, e estou na PUC-SP iniciei minha carreira docente aqui no PCS em meados de 1988, Gostaria de falar um pouquinho sobre minha trajetória na área de ensino superior.

Lecionei ao longo de todos esses anos diversas disciplinas em programas de graduação e de pós-graduação. A gente tem sempre a disciplina-trote, quando começamos a trabalhar. Aqui na Poli comecei dando aulas de sistemas digitais, laboratório de sistemas digitais, etc. Mas eu acho que o que me marcou muito no inicio da minha carreira foi participar da implantação do curso de engenharia de computação, implantado em Cubatão, onde ministrei disciplina de linguagens formais e autômatos para o curso de engenharia de computação. Senti que tendo participado dos dois cursos de engenharia da poli, engenharia elétrica com ênfase em sistemas digitais e também engenharia de computação, eu tive a felicidade de perceber duas perspectivas distintas:

No curso de engenharia que também chamamos de tradicional, de engenharia elétrica, ele tinha naquela época (mas se transformou muito nessas mudanças curriculares) de

meados de 90, a gente percebia claramente que havia uma forte ênfase para o lado da chamada engenharia, a necessidade de você poder produzir e realizar coisas, em detrimento de uma formação mais teórica na área da computação, enquanto no curso de engenharia de computação a gente trouxe umas disciplinas que reforçavam um pouco mais essa base teórica.

Eu sentia uma grande diferença no perfil do aluno formado: o formado em engenharia de computação, com essa base teórica mais forte, tinha uma desenvoltura e se constatava uma grande maturidade (talvez também por conta da diferença de formato do seu curso, com quadrimestres intercalados acadêmicos e de estágio) do aluno já na metade do curso, enquanto nosso aluno tradicional de engenharia elétrica chegava por vezes no quinto ano (quando tinham a disciplina de compiladores que cheguei a ministrar para eles) em outra condição. O aluno tinha uma grande dificuldade de entender conceitos básicos de compilação, talvez em decorrência dessa ênfase tão forte em tecnologia.

Na PUC-SP eu tive a felicidade de construir um currículo novo. Fizemos uma reforma curricular, e eu insisti para que tivéssemos uma base teórica mais forte. Aí me dou o direito de contar o porquê disso. Eu na minha vida acadêmica acabei me distanciando um pouco da linha teórica, e pendi para o lado da engenharia de software, e a gente ao conceber modelos de software a gente precisa mexer com arquitetura e no próprio desenvolvimento do sistema de software em si. e uma das minhas questões filosóficas gira em torno desse ponto.

Desenvolvimento de um sistema de software é uma atividade realizada individualmente ou em equipe? Dependendo da resposta dada a essa pergunta, o currículo de formação muda. Aproximando-me mais da engenharia de software, percebi a importância de comunicar para os alunos o lado teórico. E aí, sim: durante os momentos iniciais da formação dele, a exposição à teoria ajuda a entender diversas práticas que a gente acaba adotando durante o desenvolvimento de sistemas de software.

A preocupação com a modelagem, por exemplo, de um sistema, na minha visão é fundamental.

Existem linhas de pensamento que acreditam ser possível produzir-se o código diretamente pulando ou até não conduzindo uma etapa de modelagem mais formalizada.

Eu não vou muito nessa direção - no tutorial que apresentei aqui no ano passado acredito ter deixado isso bem claro: o *modeling* bem feito, suportado por técnicas de teorização, ajuda na produção de software de maior qualidade.

O que eu gostaria de acrescentar em relação a esse lado, de formação, é atrelado ao currículo atual. Como está o currículo atual dos cursos de formação na área computacional? A gente observa que existe uma ênfase bastante grande (inclusive a professora. Cláudia reforçou isso), a influência que o mercado tem na decisão da composição curricular.

Acho isso perigoso. É importante que a gente traga essa preocupação de mercado, para dentro do currículo, mas eu vejo que o aluno na graduação tem que ter uma formação mais completa, mais abrangente. Isso envolve não apenas a área de mercado, mas também a formação pessoal, do aluno como ser político também, capaz de refletir, de pensar, e não apenas de andar na maré de mercado.

Nesse sentido, a gente acompanha - eu participo nas equipes de avaliação de instituições de ensino superior do MEC, e a gente acaba observando que os currículos são fortemente influenciados pelo mercado, currículos que tenho observado nos últimos dois anos em que tenho participado dessas comissões.

E até que ponto - e agora eu entro num terceiro momento - o currículo tão influenciado pelo mercado, e o ponto que levanto e trago para a discussão em relação ao corpo docente.

Será que o modelo didático e pedagógico que o corpo docente adota também não tem sua direta influência na maneira com que se percebe a teoria da computação? Se a gente pegar uma aula típica de um curso de computação, que envolve umas 20 semanas de aulas expositivas, em que o aluno ouve e o professor fala, este modelo pedagógico é apropriado para que se traga o compromisso com o entendimento de uma teoria? Eu acredito que não.

Sou um pouco mais revolucionário. O próprio modo com que a gente apresenta nas aulas o currículo, que já tem sua influência direta para o mercado, a forma como a gente apresenta as aulas tem também sua parcela de culpa. A questão que tem me perseguido já há algum tempo, e que consegui finalmente materializar em dezembro do ano passado: desenvolvimento de software é uma tarefa individual ou uma tarefa de equipe? Dependendo da resposta a essa pergunta, a parcela teórica que a gente tem que absorver para fazer bem essa tarefa na verdade tem qual participação? Como é que nós professores, corpo docente, respondemos por essa outra parte da formação acadêmica?

Para finalizar minha fala, lembrando que estamos numa mesa redonda, que, portanto envolve debates, e eu acho que já trouxe algumas questões contundentes, gostaria de colocar mais uma questão: supondo que peguemos uma disciplina típica de programação, ou linguagens de programação, como é que essa disciplina é apresentada ao aluno? Tipicamente a gente apresenta um problema, normalmente essa disciplina é apresentada inicialmente para o aluno, a gente mostra, põe no algoritmo e a gente fala assim: agora codifique o algoritmo em uma linguagem 'xyz' qualquer. Essa dinâmica de aula incentiva qual posição? Está amenizando o que? Tenho uma posição, e não sei se isso vai trazer alguma questão adicional, mas gostaria de reservar minha posição em relação exatamente a essa outra colocação. Se pegarmos uma disciplina típica de programação, na qual se apresenta um problema -- algo mais ou menos assim: dados os três lados de um triângulo, calcular sua área -- e a gente dá um algoritmo, que é trivial, e pede para o aluno como exercício implementar esse algoritmo, estamos incentivando o que com isso? Estamos contribuindo de que maneira para a formação desse aluno?

Ricardo:

Nessa primeira rodada, tivemos aqui algumas questões que se mostraram em primeiro lugar muito semelhantes nas diversas instituições. De certa forma a visão dominante é essa, o professor Marcus trouxe a visão dominante no primeiro mundo (Estados Unidos) e aparentemente o que fica aqui para nós são esses resquícios de decisões que foram tomadas no passado, e cujo reflexo hoje são esses que o professor Zorzo comentou. Por exemplo, o financiamento de pesquisa, a avaliação de pesquisadores, a avaliação de congressos, que a professora Claudia estava comentando também. Sobre a

formação do aluno, toda essa dúvida que nós temos, essas pressões, externas ou políticas, ou seja, o que for, que podem ter surgido inclusive de uma compreensão equivocada, conforme a interpretação do professor Marcus, das decisões tomadas nos Estados Unidos. De certa forma essas decisões vão tornando, especialmente aqui no Brasil que nós podemos comentar, a forma de ministrar as aulas, passando pela questão didática que o professor Ítalo havia comentado, em uma espécie de dança de entretenimento. Então nós apresentamos algo bem simples para entreter o aluno, não é isso? Não se comenta de forma aprofundada os fundamentos, porque afinal de contas isso não tem muito interesse. Talvez porque o mercado não queira, ou porque o aluno não quer, e aí nós ficamos naquela dúvida, se nós pegarmos tanto as escolas públicas quanto as escolas particulares nós vamos ver esse tipo de problema também. Então essa visão que a mesa passou de certa maneira converge, mas ainda há muitas perguntas, não é? Eu gostaria de recomeçar por aqui mesmo. Alguém gostaria de se manifestar sobre essas questões?

Zorzo:

Eu acho que o que o Ítalo falou me instigou a falar mais uma coisa. O nosso currículo de formação na área de computação é dividido em engenharia de computação, sistemas de informação e ciência da computação.

E há uma preocupação da SBC em como é dada essa formação. Só que como o Ítalo citou, eu posso falar de dentro dessa área de teoria, quando você tem uma disciplina que é obrigatória da área para o curso se chamar ciência da computação. Cumpre os objetivos e a ementa como estabelece e preconiza o currículo de referência da SBC, mas como é dada essa aula? Como é o plano de ensino? Como é que esse professor cobra esse aluno? Ele cobra a fundamentação teórica do ponto de vista da ciência ou ele está cobrando os aspectos tecnológicos – mesmo daquilo que pode ser dado de um lado e do outro –? Então é aí que existe a diferença. E nós vemos diferença, e isso de uma escola para outra. E alguns livros texto estão evidenciando e nos levando a crer que mesmo que a disciplina seja de ciência o que se tem dado são apenas os aspectos tecnológicos.

Cláudia:

Bem, concordo com as opiniões que foram apresentadas, e que incluem uma grande variedade de posições sobre o que se faz em ciências da computação, e engenharia de computadores. É preciso dar-lhe uma importância maior.

Certamente gostaria de traçar um paralelo com a situação em que uma criança aprende matemática.

Quando o ser humano se vê diante duma situação em que pode aprender alguma coisa, ele geralmente se mostra muito refratário a encarar algo que desconhece.

E de que forma aprendemos a matemática? Brincando.

Como aprendemos a teoria da computação? Talvez, ao representar o mundo, a partir de suas aplicações práticas.

Normalmente quando as pessoas não são capazes de entender as ciências, acabam pensando que estas não têm utilidade, ou então, que não existem aplicações para os resultados que elas nos oferecem.

É importantíssimo o que foi mencionado acerca da aproximação entre a tecnologia e a ciência, porém deve-se tomar cuidado para que isso seja feito de tal maneira que não sejam deixados de lado os temas teóricos, aqueles que formam as próprias origens das quais a tecnologia emerge.

É preciso dar-lhe mais importância, pois vem muito marcado pelo mercado, ou pela necessidade de colocar os alunos nos postos de trabalho. As necessidades latino-americanas muitas vezes obrigam as pessoas a trabalhar precocemente, e talvez abdicar do aprofundamento de sua formação científica.

Participação da platéia:

Daniel:

Boa tarde, eu sou o Daniel, sou aluno de doutorado do LSI, e tive a oportunidade de ser membro do corpo docente de duas instituições, lecionar essas disciplinas de linguagens formais e autômatos, teoria da computação, e numa situação emergencial participar de uma comissão de coordenação de curso.

Eu como recém saído da graduação, recém terminado o mestrado, concordo bastante com a opinião do prof. Ítalo, e acho que inclusive tenho outra justificativa para essa questão que você coloca de se o desenvolvimento é para ser feito por uma pessoa ou por uma equipe, onde o conhecimento a base e o modelo têm de ser entendido.

Em minha opinião a questão profissional o argumento profissional fácil de convencer os alunos é no sentido de você perguntar a ele qual a diferença que você vai ter no mercado entre uma pessoa que seja autodidata ou que pagou por um curso de uma linguagem de programação qualquer? A diferença de um profissional que gastou um tempo se graduando é que ele consegue entender o modelo e não só uma linguagem para desenvolver. E aí, junto com essa coisa sobre a importância de linguagens formais, da importância dos aspectos teóricos, me parece que muito disso é também da formação do professor e do jeito como ele conduz a disciplina, mas também da forma como o aluno vê esses aspectos soltos. Ele não conseguir amarrar esses aspectos, entender que um curso que começa com questões lógicas, passa por formalismos e depois vai entrar em UML, vai entrar em engenharia de software, alguma técnica de modelar, que é o que vai dar consistência e vai dar reconhecimento para ele no mercado. Eu acho que é essa a diferença do ponto de vista de uma pessoa também nesse meio. Eu queria dar essa contribuição.

Ítalo:

Bem, Daniel, também vou nessa mesma conclusão sua, de que eu acredito na importância do modelo, precedendo ao código. E o modelo, esse artefato, essa representação intelectual, acaba sendo uma mídia mais apropriada para uma discussão da equipe do que o próprio código. E nós sabemos disso. Como é que se faz manutenção de código? Bom, código por si só é código, ninguém entende. E se nós fizermos um esforço maior então ninguém vai entender, nem o próprio. Talvez o modelo seja mesmo a mídia mais apropriada para a discussão e isso leva a outra conclusão importante. Eu gostaria ainda de reservar minha opinião para não influenciar as opiniões dos demais sobre esse assunto.

Marcus:

Eu queria aproveitar para fazer alguns comentários, e é o seguinte: acho que existe uma pressão de mercado e também acho que os próprios alunos, quando se deparam com disciplinas mais áridas e complicadas têm uma tendência de afastamento natural, mas aí cabe também ao professor motivar e trazer esses alunos para a disciplina. Então nesse sentido, o que a gente observa nos livros texto clássicos que a gente vê por aí todos eles entram muito diretamente na teoria, nos conceitos, nos resultados, e são muito poucos aqueles que procuram contextualizar as aplicações e também falar sobre o que essa teoria já contribuiu do ponto de vista tecnológico, de produtos e de serviços. Então acho que assim, fazendo uma mea culpa, acho que ao menos no meu caso e em outros que conheço acho que falta para mim e para esses professores trazer mais material informativo para a aula, para que os alunos entendam a importância daquela disciplina, para que ela seja vista não apenas como uma carga obrigatória, mas como uma disciplina que vai fazer toda a diferença no futuro profissional daquele indivíduo: porque ele vai estar apto a expandir suas fronteiras do conhecimento da computação, de um lado, se ele tiver uma boa formação teórica, ou então, for um mero usuário de ferramentas e de tecnologias que outros vão desenvolver para ele. Nesse sentido, esses artigos que eu citei aqui no começo, todos eles trazem essa preocupação, e, portanto a leitura deles vale inclusive por isso, porque eles são ricos em exemplo de como marcos na história da teoria da computação influenciaram definitivamente e delinearam o futuro da computação. Como muitos produtos. muitas tecnologias, muitos serviços que a gente usa hoje, derivam de muitas pesquisas de base que foram feitas há 10, 20, 30, 40 anos atrás. Então está aí um material que eu acho que pode e deve ser explorado, juntamente com a teoria. E não ficar simplesmente naquela insistência às vezes um pouco teimosa de que a teoria é importante, e que vocês um dia também vão ver para que vá servir.

Ricardo:

Deixem-me inserir então mais uma variável: essa mudança de geração. Nós percebemos que há nos alunos que vêm ingressando nas escolas atualmente, certa impaciência, e uma dificuldade grande de se concentrar em aula, porque nesse ambiente ele não consegue observar uma resposta imediata. Considerando, além disso, essas condições curriculares, eu gostaria que vocês agora refletissem um pouco sobre isso, e se a platéia quiser se manifestar, por favor, faça-o.

Jairo Galindo:

Sou Jairo, sou professor também, e escutei os colegas aqui nessa reunião, e parece magnífica essa abordagem, quanto à questão dos projetos submetidos, e não aprovados, e assim por diante.

Com referência a isso, a universidade pública e a universidade particular, e seus currículos estão orientados do ponto de vista do mercado, e o professor deve ver isso também, e o aluno já está incentivado.

Agora a pergunta que eu faria é para vocês que supostamente agora estão esclarecidos, abertos a essa parte teórica, que estão fazendo vocês para que mude este quadro?

Porque na realidade se fala muita coisa, mas vocês como professores vocês têm de cumprir essa função e, aliás, eu não sou nem formado, e de pronto eu não cumpri essa função, um telegrama chegou, e fui fora. certo? Agora, em referência a isso, eu também tenho uma preocupação. Estamos aqui com o professor Ricardo, que é coordenador da mesa, e falou sobre a importância do conhecimento científico na tecnologia adaptativa. Enumera várias áreas que fazem parte do evento, então trago essa preocupação: eu não vi nenhum trabalho exposto sobre a dinâmica da fala, a dinâmica da linguagem.

É um trabalho pioneiro, inovador, que vocês têm conhecimento, mas eu não vi nenhum trabalho, e isso ajudaria muito no esclarecimento em modelagem, sistemas de síntese de voz, na área de reconhecimento de voz, que tanto nossos colegas sofrem para adaptá-lo, é porque se trabalha muito com estatística, e a língua, e não percebemos que o que a população fala é um movimento, um dinamismo, e ela deve ser abordada porque ela faz parte da ciência, e ela deve ser abordada, porque o mundo é diversificado, e o mundo ocidental, e já sabemos que o mundo também pertence há o mundo oriental, e que o mundo oriental também dirige e governa, e o mundo ocidental também dirige e governa, e vem trazer o mundo ocidental, temos uma infinidade de diversidades e vemos também que há pessoas que se pronunciam sobre as questões. E esse pronunciamento vai e força dentro do poder, e se tem essas questões, certo? Então meu refletir vai assim encaminhado sobre essas questões.

Zorzo:

Eu peço a palavra, para falar algumas coisas para você.

Primeiro, que a formação do aluno que é voltado para o mercado e que tem uma pressão do mercado para dar uma formação mais próxima, isso está presente em todo lugar. Eu leciono em uma universidade pública, e nós vemos o assédio de empresas sobre os alunos desde o primeiro ano. Já os cursos de treinamento de ferramentas, e já querendo levar esse aluno para o mercado, focando na carga desse aluno.

Como nós fazemos para mudar isso? O docente não está atento a isso. Nós viemos fazer nosso papel aqui. Expondo, falando, ficando preocupado com isso, e que cada um de nós seja um replicador. Na verdade não é uma pessoa falando. Parece que é uma voz única de todos os que estão aqui. Então isso tem que ser replicado e expandido.

Vou pedir desculpas a vocês e para a mesa, muito obrigado pela minha participação, mas tenho um compromisso e preciso me retirar.

Muito obrigado a todos vocês.

Ricardo:

Você quer fazer uma pergunta, Amaury?

Amaury:

Apenas queria fazer um comentário para complementar, e colocar mais alguns pontos em discussão.

Eu atuo atualmente na universidade federal de Mato Grosso do Sul, que é um estado do interior do Brasil, onde a gente tem uma concorrência média, entre cursos do interior, em torno de dois a três candidatos por vaga, e eu sou um defensor da teoria, gosto de teoria, trabalho com teoria, desde a graduação até hoje, e eu acho que foram citados alguns pontos e eu gostaria de destacar também num primeiro momento essa questão das humanidades, da formação do aluno político e

cidadão. Eu acho que isto é um ponto crucial também porque eu acho que o professor principalmente quando ele trabalha com uma disciplina teórica, tem que ser um bom comunicador.

Mas a gente tem uma dificuldade muito grande porque na nossa realidade distante do centro comercial do país, a gente acaba recebendo alunos com uma base matemática mais fraca, e também com muita dificuldade de escrita, e quando a gente fala em teoria, a escrita, principalmente a formal, principalmente a formalização e a capacidade de abstração, que às vezes é trazida pela própria base matemática, e esses são pontos a meu ver fundamentais para que se possa trabalhar melhor a teoria.

Então, no outro extremo, a gente tem aquela questão do mercado.

O que acontece às vezes, eu escutei, ah, você vai trabalhar com o professor tal? Ah, não, então você vai fazer um mestrado! Ah, você vai trabalhar com o professor cicrano? Ah, não, então você vai para uma empresa! Então essa visão e essa separação existe. O que é um mestrado? É um vínculo que você estabelece com a academia, e você vai acabar em uma universidade como pesquisador. Em hipótese alguma se cogita a possibilidade de ter aquela empresa que tenha a necessidade de pessoas com aqueles conhecimentos teóricos, para trabalhar em projetos que precisam dessas informações teóricas. Hoje a gente tem lá em Mato Grosso do Sul uma empresa (a Tecsinapse) que é de ex-alunos do curso de computação, alguns já fizeram mestrado outros doutorado, e eles trabalham essa parte de modelos, de teoria, e eles também aplicam no desenvolvimento de aplicações e ferramentas. Então quer dizer hoje, a gente está começando a mudar a cabeça dos empresários, mas acho que a gente tem um longo caminho pela frente. Porque essa é uma dificuldade que a gente tem de falar não? Vemos que o papel da universidade é realmente dar uma boa formação de base. Agora também a gente tem outro argumento mais forte, que é a questão do investimento nos centros federais de educação, tecnológica, que também tem um foco um pouco diferente, então a gente tem aí uma graduação, universidade. Qual o papel da universidade, e qual é o papel dos centros de formação tecnológica? Eu acho que são coisas que precisam ficar claras, e como o professor Zorzo disse, a SBC tem lá os currículos de referência e nos cursos de sistemas de informação, que eles colocam a computação como meio, enquanto engenharia e ciência colocam a computação como fim.

Outro ponto interessante e foi discutido até no congresso da SBC no ano passado, perguntaram na SBC há as comissões especiais que são as comissões temáticas, de diversas áreas, redes, IA, linguagens de programação, mas não existe uma comissão especial de teoria da computação, e o engraçado é que são as pessoas assim mais fortes e eles justificam dizendo nós estamos permeando todas essas outras comissões. No mais, eu acho que há mesmo essa necessidade de coesão. Era isso.

Cláudia:

Bem, gostaria de mencionar que estamos quase todos bastante de acordo, e, certamente, o mercado precisa de um profissional que faça e aplique a tecnologia, mas precisa igualmente, precisamos todos, como sociedade, da produção de tecnologias novas. Não apenas usar a tecnologia existente, mas sim criar uma tecnologia nova, que ainda não existe.

Para tanto, creio que o que temos de fazer seja diferenciado, dando opções e possibilidades diversas para os estudantes.

Se for válido querer dedicar-se ao mercado, ao uso puro e simples da tecnologia, também o é que se deseje ter mais conhecimento teórico, aplicando-o em atividades de pesquisa.

Como na USP a universidade onde trabalho é uma universidade relativamente grande com muitas especialidades de humanidades, de economia, de engenharia, e temos também especialidades, de matemática, física, química, história geral, somos uma universidade multidisciplinar.

Mas, dentro da universidade, custa-nos muito fazer com que essas disciplinas conversem umas com as outras. Cada área, a psicologia, a economia, tem as suas necessidades de aplicação da computação. Mas é muito difícil o diálogo entre nós.

Uma das dificuldades que encontramos recentemente foi o de nos comunicarmos com lingüistas acerca do que desejamos aplicar. De um lado, eles não nos compreendiam, e outras vezes nós é que não os conseguíamos entender muito.

Precisamos talvez de uma tradução das regras lingüísticas. Nós vemos muito as coisas práticas, diretas, tudo regulado.

Os lingüistas, por um lado, tinham mais a postura de que as palavras se formam como uma fala, como um som. Nós, do outro, achávamos que para explicar a mesma coisa, deveria existir alguma regra...

Bem, isso ocorre em todas as áreas. É muito importante que conversemos entre nós, profissionais, sobretudo com as vantagens que exibe uma universidade como esta e como aquela em que eu trabalho, nas quais há profissionais de todas as áreas, e nos permite isso: apresentar talvez esse mundo das aplicações da ciência, resolver em conjunto problemas emanados das diferentes áreas. Algo que eu consegui observar nos trabalhos que vi aqui, e que muitíssimo me agradou.

E uma coisa que nos falta muito no Peru, e que é muito necessária, é a aplicação da computação nos problemas ambientais.

Atualmente é muito importante para a Amazônia, e tanto como no Brasil, temos os mesmos problemas, representados em temas tais como o desmatamento, o problema da água e o da contaminação do ambiente.

E temos também no Peru, assim como vocês têm aqui no Brasil uma riqueza muito importante representada pela Amazônia, que é um dos poucos pulmões da humanidade. No Peru não se investe muito aí, porque também não se tem diálogo interdisciplinar entre geógrafos e ambientalistas.

Isso esperamos que mude. O governo brasileiro com o SIVAM (não sei se todos ouviram falar disso), grupo que vê os temas ambientais na Amazônia, e que está tentando trabalhar com o Peru, mas como tudo o que acontece entre governos o processo é lento, é muito lento, e esperamos que avance, porque é vital, e creio que retorno daqui com excelentes idéias que pretendo pesquisar para canalizar esforços em direção ao tema ambiental. Eu os felicito por isso, pois reconheço que é algo muito, muito importante.

Marcus:

Eu só queria voltar ao assunto da motivação do aluno de teoria da computação. especialmente de linguagens formais e autômatos. Eu acho que uma coisa que pode ser feita e inclusive isso é parte de um trabalho desenvolvido aqui com o

Ítalo e com o João se a gente for dar uma abordagem tradicional a essa disciplina, ela é eminentemente expositiva. essa abordagem, evidentemente, tem mil problemas, porque o aluno não se concentra, dispersa facilmente, principalmente aquele aluno das gerações mais novas, que não consegue ficar concentrado por mais de cinco minutos; O que se nota é que começaram a surgir aquelas ferramentas gráficas para você ensinar linguagens formais mas em um ambiente interativo, onde o aluno desenha o autômato, aí ele aplica transformações no autômato, e na gramática, na expressão regular. Surgiu por quê? Surgiu porque é uma ferramenta gráfica, interativa, muito próxima daquele ambiente que o aluno conhece, usa no computador, gráfica, animada e assim por diante. Foi uma tentativa de ganhar esse aluno para o estudo da teoria da computação. Bom, a partir desse momento, nós desenvolvemos uma proposta alternativa para o ensino de linguagens formais, que é baseada não no uso de ferramentas fechadas, como é o caso de JFLAP e várias outras, que se conhecem por aí, mas uma idéia de aproximar a disciplina de caráter teórico abstrato das disciplinas de caráter mais prático, para as quais os alunos normalmente têm maior motivação. Então a gente encontra no curso de engenharia e de ciência o aluno geralmente motivado, para fazer os modelos UML, para fazer código em Java, em C ou no que quer que seja. Pensamos então em como capitalizar essa motivação, e fazer com que a partir dela o aluno tenha interesse também em estudar a teoria (linguagens formais e autômatos). Então fizemos um trabalho, especialmente aqui o Ítalo, que foi um trabalho no sentido de trazer, de propor modelos de engenharia de software para esses dispositivos da teoria (os vários tipos de reconhecedores, os diversos tipos de gramáticas, etc.) com que vantagem? O que a gente espera dessa proposta didática? primeiro, são problemas abertos, o aluno vai ter a chance de discutir o modelo, criticá-lo, desenvolver o modelo, e a partir desse modelo denotado em uma linguagem de programação, estender sua funcionalidade, entrar no mérito do seu desempenho, criar outras implementações a partir de modelos de outros dispositivos por extensão ou por analogia, por similaridade, mas tudo isso de forma a motivar o aluno a estudar teoria. Então a gente está escapando um pouco daquela abordagem da interface gráfica por quê? Primeiro porque ela é fechada, então o aluno abre o JFLAP lá, manda minimizar o autômato, clica um botão e o autômato aparece minimizado, em outra janela, E agora, como é que aconteceu isso? É claro que a ferramenta é útil, para ele conferir o resultado de um exercício, que ele fez manualmente, para ver se está certo ou não, mas ele não pode interferir na ferramenta, ele não pode estender a ferramenta, ele não pode fazer outra parecida, ele não pode ver como é que ela funciona, como é que ele implementa um algoritmo qualquer, então nós procuramos escapar dessa linha e fomos trazendo aí uma proposta diferente. Então talvez esta seja também uma maneira de valorizar perante os olhos do aluno a questão da teoria da computação, mostrando que ela primeiro, está contextualizada em um ambiente de engenharia de software, e segundo, que ela pode criar muitas possibilidades de aprendizado, que de outra forma ele não teria. De qualquer maneira, a pessoa mais adequada para falar sobre esse projeto, está aqui do meu lado (Ítalo), eu só quis realmente introduzir o assunto e mostrar como é que no ensino é possível criar perspectivas diferentes de motivação do aluno, que eu acho que é importante, para a gente manter a área valorizada e enfim, em alto conceito perante não só os colegas, mas também os alunos que são nosso público-alvo aqui.

Ítalo:

Eu pedi a palavra para atacar alguns pontos: o prof. Jairo havia comentado, e o prof. Amaury também, e eu gostaria de retomar essas questões. E agradecer aqui ao prof. Marcus, por me colocar na roda novamente, O professor Jairo nos trouxe a pergunta: o que a gente está fazendo? O que estamos fazendo a respeito? E eu poderia interpretar da seguinte maneira: a respeito do distanciamento da teoria e do aluno, aqui a gente está falando a respeito desse distanciamento. E o professor Amaury retomou e aí eu me permito acrescentar à palavra Professor, vírgula, a palavra Educador. Ou seja, nós no papel de professores ou nós no papel de educadores.

Em relação ao que estamos fazendo, eu tenho uma visão do que foi relatado anteriormente e vejo que diretamente temos de atacar dois pontos. o currículo que a gente está realizando nos cursos de computação. Veja que este currículo é influenciado por mercado, entretanto, eu tenho alguns senões em relação a simplesmente aceitar isso. E aí eu retomo as considerações do professor Amaury. Nosso papel é pegar as necessidades do mercado e trazer para dentro do currículo para formar o aluno que atenda o mercado? -- essa é uma questão -- ou uma das nossas missões tem por prioridade formar um profissional que eventualmente pode se colocar imediatamente no mercado, mercado de curto prazo ou mercado de longo prazo. Essa formação desse profissional exige um currículo. E que currículo é esse? Não vejo que a influência direta do mercado no currículo seja uma resposta boa. Claro que temos que considerar esse fator, mas eu não vejo este como sendo o fator primordial, eu vejo que o aluno de graduação tem que ter uma formação mais ampla, em relação a pelo menos quatro aspectos: o conhecimento, óbvio, e mais três outros importantes aspectos relacionados com a afetividade. às vezes a gente forma um profissional que é fechado, um profissional, usando uma expressão um pouco mais popular, um profissional meio "reto". Principalmente na área computacional: ele liga um computador, faz o que tem que fazer, oito horas depois ele desliga. Falta aí a componente afetividade, que não foi incorporada por nós docentes àquele profissional em formação. Um terceiro ponto relacionado com valores, que valores essa pessoa tem, que valores nós professores comunicamos a esse aluno? E agora deixamos nesse momento de ser professores para ser educadores. Que valores nós passamos para eles? Valores éticos? Valores comportamentais? E habilidades. Temos de trabalhar o lado das habilidades, agora como trabalhar habilidades para comunicar o conhecimento teórico? E aí eu retomo o segundo ponto que é que não é apenas um modelo curricular influenciado pelo mercado, e nós temos de tomar cuidado com isso, mas que modelo didático e pedagógico nós temos que trabalhar – repensar. Qual é o modelo mais apropriado para nós, e agora voltamos à pergunta original - o que estamos fazendo? - estamos pensando no modelo de um novo educador para a área computacional? A gente chega lá e apresenta: o código está aqui, entenda você depois em casa? Ou a gente leva um laptop para a aula e mostra o programa rodando -

entendeu? - Será que esse modelo didático pedagógico é apropriado? Não só para ciência da computação, mas também para reforçar a importância da teoria na sua formação profissional eu estou convencido de que o modelo didático pedagógico tem que ser diferente, agora isso requer não só uma mudança da nossa parte como corpo docente, mas também uma mudança curricular. Em função disso, com o professor Marcus e com o professor João estamos ativamente trabalhando pelo menos da minha parte neste sentido. Engenharia de Software a gente sabe que é uma área atraente. Por que não usar o atrativo da Engenharia de Software de tal maneira que ela atraia também outros elementos como a teoria da computação? Esse ensaio que fizemos é um ensaio que não vejo ainda ensaio similar a este, mas é um ensaio que estamos tentando no sentido de atrair através da Engenharia de Software o aluno para que ele se aproxime do lado teórico que é de suma importância. Apenas para completar, essa minha colocação, a gente sabe que o modelo de máquinas de estados é o modelo central, em qualquer peça de software que a gente vá produzir. Agora, como é que a gente comunica que o modelo de máquinas de estados é tão central assim quando um aluno pega um ambiente em movimento como o Eclipse, por exemplo, e brinca de arrasta-clica-cola-roda cadê o modelo de estados? Como é que a gente está ajudando o aluno a enxergar as transições dos vários estados do sistema que ele está desenvolvendo, num ambiente como esse? Seria esta uma maneira de a gente dar aulas no laboratório? Não sei. Não me sinto confortável em ministrar aulas desta maneira. E daí essa nossa preocupação: Engenharia de Software como ponto de entrada para apresentarmos teoria. E certamente eu acredito que haverá o *feedback*. Uma vez que o aluno consegue absorver a importância da teoria, ele fará uma engenharia melhor, e não a engenharia que muitas vezes a gente acaba vendo por aí, que é uma engenharia em que você acaba não podendo dar manutenção de software. Por que às vezes você não documenta as coisas? A resposta primária é: não deu tempo. Mas não é isso, não. Em geral é porque eu não sei o que eu fiz. Como então eu vou documentar isso? Como é que você testa? Bom, se não foi planejado, ... Logo, a gente não vê testabilidade, a capacidade de você olhar para um código e entendê-lo. Reflexo da qualidade com que o software acaba sendo passado para os alunos. Talvez a revisão teórica propiciada por essa maneira diferente de a gente mostrar aquilo nos leve também a gerar uma engenharia de software de melhor qualidade.

Ricardo:
Parece que nós temos mais uma pergunta?

Reginaldo:
Eu acredito que a teoria da computação é o exato limite do que você pode ou não pode fazer em termos de computabilidade. Com certeza vão surgir problemas no mundo que só a tecnologia não vai resolver. Vai precisar de uma evolução na teoria. Então eu penso que um país em que o currículo de engenharia e de ciência da computação se afastam da teoria da computação é um país que vai ficar cada vez mais ameaçado de se tornar dependente tecnologicamente. Então minha pergunta é bem dentro desse enfoque: será que não se perdeu a visão estratégica da importância da teoria da computação nos currículos? Porque eu acho que o assédio das empresas existe em todo lugar do mundo. Mas por exemplo um país como os Estados Unidos não se priva de investir em pesquisa nessa área. E mesmo aqui no Brasil existe demanda para pessoas que querem pesquisar. Se esse aspecto estratégico for perdido, talvez essa não tenha sido a causa de os nossos currículos estarem se afastando tanto?

Ricardo:
Vamos ver se essas respostas são válidas apenas no Brasil.

Cláudia:
Bem, decorre a questão se o mercado deve ou não influenciar-nos tanto. É importante ouvi-lo. Creio que os profissionais vão ter necessidade de trabalhar em algum lugar. Por outro lado, é certo que a ciência deva ajudar-nos a produzir tecnologia, pois não devemos depender tecnologicamente de outros, como ocorre em muitos países.

Como foi mencionado, a maneira de atrair o interesse pela ciência, o interesse pela teoria, atualmente tem origem na prática, e parte de mostrar aquilo que se pode fazer.

Testamos idéias mais de uma vez na nossa universidade. Tenho basicamente atribuído aos meus alunos o desenvolvimento de um jogo eletrônico.

Mas eu lhes deixo muito clara a exigência de que usem algoritmos de heurísticas, técnicas de inteligência artificial, alguma coisa adaptativa, que eu lhes recomendo que utilizem.

Em muitos deles se vê o nascimento da curiosidade, e daí vem o interesse do aprofundamento teórico.

E consigo depois ter gente interessada no aprofundamento da parte teórica, com conclusões muito interessantes, como por exemplo, aquele trabalho de Rosalía, que mostramos esta manhã.

Também a universidade em que trabalho faz reuniões periódicas com pessoas que trabalham em empresas, com diretores de empresas, e com ex-alunos que já têm uma experiência acumulada de vários anos de trabalho em empresas.

No entanto, sempre temos pensado ser importante que a universidade esteja também na vanguarda das empresas, e temos uma pergunta cuja resposta ainda permanece no ar há muito tempo, e é um comentário que fazemos muito em nossa universidade:

Quem é que deve traçar o caminho, a empresa ou a universidade? Mas não a universidade como questão de capricho, mas como uma questão de visão do futuro, e, bem... resta-nos sempre essa reflexão.

Ítalo:
Deixe-me apenas fazer um complemento. Desculpe-me, seu nome é? Reginaldo.

Você levanta uma questão da eventual perda da visão estratégica da função da teoria no currículo.

Deixe-me fazer só um complemento em relação ao seguinte: O papel da instituição de nível superior hoje continua sendo o papel de ser o centro, a fonte, a origem do conhecimento? Eu não mais acredito nisso. Acho que antigamente sim, e acho que a resposta hoje é um pouco diferente, pois não é a instituição de nível superior a única origem do conhecimento. Alguém que quer adquirir

conhecimento se dirige à instituição de nível superior. Um exemplo clássico é um ponto de Internet. Você não precisa ir até a universidade para adquirir conhecimento. Abra o Google e pronto. O conhecimento está à sua disposição. Logo, eu vejo que a instituição de nível superior começa ter também o seu papel se alterando nos tempos modernos. Não mais como um centro de conhecimento porque você não vai à universidade para buscar conhecimento, porque não é só lá que você vai encontrá-lo. Então eu não sei se essa perda de visão estratégica seria uma possível explicação para a gente ter uma fraqueza no relacionamento da teoria da computação nos currículos. Eu enxergo que o conhecimento relacionado com a teoria da computação também está disponível para nós não apenas nas universidades. Agora, como atrair alguém para esse conhecimento? Como fazer com que esse alguém se deslumbre, se sinta apaixonado por teoria da computação? Nós responsáveis por formar profissionais dessa área sabemos dessa importância, então nós temos também a responsabilidade de motivá-lo, para que ele se atraia se sinta atraído por esse conhecimento, de forma que ele possa conquistá-lo não só aqui na universidade, mas fora dela também, que é nossa realidade. Então eu continuo colocando o ponto da responsabilidade nas nossas costas --- da importância que ela tem no currículo ---. Deixe-me colocar algo no rodapé: dificilmente alguém tem atração pela máquina de Turing, um aluno de graduação. Você fala assim: está aqui uma TM (Turing machine). Ele fala prazer em te conhecer, não te conheço. Quanto mais fazer com que ele se sinta emocionado por uma tecnologia adaptativa. Se ele não consegue, digamos assim, chegar a uma teoria de máquina de Turing, e se apaixonar por ela, dificilmente outras teorias deverão atraí-lo. Agora, como é que nós atraímos esse potencial estudante para esse tipo de conhecimento? O que nós estamos fazendo a respeito disso? Eu vejo que esta perda em relação à visão estratégica eu coloco muito dela em nossas costas. Mesmo com currículos que são currículos engessados, e a gente tem de responder a órgãos superiores em relação a isso, ainda vejo no corpo docente uma responsabilidade em relação ao decaimento do interesse em relação a essa área teórica. E dá a impressão que o mercado "ganha" (é claro que essa palavra não é apropriada, mas é a que me ocorre agora). Puxa, vou ensinar o lado prático para o aluno rapidamente conseguir ingressar no mercado, mas nossa responsabilidade como docente dessa área computacional tem de ser repensada. Nós estamos deixando as portas abertas para que isso aconteça. Temos de fazer um esforço para retomar a importância do lado teórico, principalmente das bases da computação.

Ricardo:
Há mais uma pergunta?

(não identificado):
Creio que seja mais um complemento. Sou um dos poucos acadêmicos do terceiro ano de graduação em sistemas, com o prof. Amaury, e todos nós sabemos da facilidade que todos nós temos hoje em dia à informação. Para ter conhecimentos, não é preciso freqüentar uma faculdade: pegando um livro na Internet eu aprendo o que quiser. "Sozinho", entre aspas. A questão principal, acho que falta um pouco, é o corpo docente, os professores atuarem não como um, mas como ao menos um grupo mais coeso. Ter um professor de teoria e um professor de uma matéria não tão teórica que se complementem. Ou que indique o conhecimento do teórico como sendo algo de pouca importância na minha formação, ou algo do tipo. Porque eu acabo tendo um ponto de vista do mercado que quer que eu saiba uma linguagem nova - um Java, ou coisa do tipo. Eu teria um profissional mais antigo que poderia me indicar: não, o que dá dinheiro é COBOL, um Adabas da vida, um Dbase. Então a gente fica pendendo cada hora para um lado, o mercado de um lado, técnicos antigos do outro, técnicos novos em outra ponta, e até a família acaba influenciando em tudo isso nesse tipo de situação. Então fica parecendo aquela situação do filho que pede uma coisa para o pai, o pai fala não e a mãe fala sim. E aí entra a parte que o professor Ítalo citou do educador mesmo, ser esse contraponto, não ser só um professor, teórico que tem que puxar o aluno para o lado da teoria, professor que não seja de uma matéria não tão teórica também contribui com isso. E mais outra coisinha: a questão do flerte com as outras matérias: biológicas, humanas: meu curso tem certa porção de humanas. porque antes de flertar tanto com as outras áreas a gente deveria namorar a própria área da computação mesmo.

Cláudia:
Na Universidade Católica do Peru temos basicamente dois tipos de professores: o professor de teoria, e o professor prático, que normalmente trabalha por horas de trabalho, e vem do mercado de trabalho.

Essas pessoas lecionam em classes de níveis superiores dos cursos de tecnologia mais aplicada. As disciplinas básicas são normalmente da responsabilidade dos professores mais teóricos.

E é de fato uma fragilidade o fato de que não conversamos muito e que não integramos muito que vemos... sempre ocorre o problema de um aluno perguntando: e para que serve isto? E não estuda profundamente porque o assunto não lhe interessa, e então acaba descartando-o.

Acaba sendo aprovado porque precisa da aprovação na disciplina para poder continuar os estudos, e quando o aluno chega às disciplinas de nível mais avançado, e reconhece a necessidade daquele conhecimento anterior que não adquiriu, cai em si e diz: mas isso tem tudo a ver com as disciplinas anteriores, que eu não estudei da forma correta.

Há esforços significativos de alguns professores, sobretudo dos mais jovens, e com eles temos conseguido reduzir um pouco essa rejeição quanto à idéia de integrar trabalhos teóricos e práticos.

Conseguimos resultados muito bons, e se é nossa a responsabilidade de interessar um pouco o aluno em trabalhos de cunho teórico, devemos entender que como docentes devemos formar pessoas de gerações muito diferentes das nossas, com uma realidade muito diferente da nossa, com pensamentos e necessidades totalmente diversos.

Talvez isso seja bom, pensar como pensam eles para poder orientá-los e formar pessoas integrais, como também se mencionou aqui anteriormente.

Profissionais que possam desenvolver-se não apenas de forma efêmera, pelos próximos cinco anos, mas pessoas que possam entrar para o mercado, e, mais que para o mercado, para a sociedade, e que nessa sociedade tenham a capacidade

de marcar com competência a sua presença não pelos próximos cinco, e sim pelos próximos vinte, trinta anos.

Nossa formação é a base que acaba moldando certamente o mestrado, o doutorado, ou as especializações que cada profissional decida ter. Mas somos nós quem temos a responsabilidade de colocar o cimento em tudo isso.

E é importante que esse cimento certamente seja dado, para que o aluno possa engajar-se imediatamente no mercado de trabalho, e também para que possa manter-se, a despeito das mudanças na sociedade.

Porque nosso tema da computação está em mudança, e varia muito depressa. Não podemos produzir profissionais para o hoje, mas devemos fazê-lo visando ao longo prazo.

Ítalo:

A professora Claudia, quando fez o seu comentário, trouxe o perfil do professor teórico, do professor prático, e me fez com isso lembrar outro aspecto que eu gostaria também de trazer aqui para a mesa, não apenas... (isto está ligado diretamente ao modelo didático-pedagógico) que é o elemento de como se faz a avaliação. Como é que a gente avalia o aluno? Temos o aspecto de apresentar para ele uma teoria da computação. Ótimo. Existe a dicotomia em si da aprendizagem, e a avaliação cobra para que possamos dizer: sim, este aluno aprendeu. Ou não. Como é o nosso modelo de avaliação? Como é que ele é feito hoje? Como é feita a avaliação? Prova. É isso. Prova é o estímulo ao aprendizado, que é como eu gostaria de enxergá-lo, do aluno motivado porque ele quer mostrar que aprendeu, ou a prova é uma penalização? "Puxa tirei cinco, passei e vou embora". Como é o nosso modelo de avaliação? Isso está integrado ao modelo didático-pedagógico, que nós corpo docente hoje temos responsabilidade de utilizar para formar o nosso aluno.

Ricardo:

Uma última pergunta, devido ao avançado da hora.

Danilo Bellini:

Eu tenho algumas coisas para falar, pode ser que demore. Sou Danilo, sou formado aqui na Poli, atualmente sou aluno de graduação na música da ECA, eu me formei em engenharia de computação aqui, e faço mestrado em ciência da computação no IME. Tenho algumas colocações aqui:

Primeira, acho que todos vão concordar com uma analogia que eu imaginei: de que adianta formarmos um jornalista que não sabe ler? Seria uma analogia para "Do que adianta um profissional da área de computação que não sabe teoria da computação?" Essa foi a idéia: não adianta tentarmos fugir do entendimento básico do que a gente vai fazer.

Uma segunda coisa é sobre o incentivo: mencionaram utilizando motivação mercadológica como incentivo. Sempre isso funciona? Eu me lembro de uma aula de Cálculo III, Não me lembro exatamente o que aconteceu o professor Hosami do IME passou um exemplo da utilidade do teorema de Stokes, mas a utilidade apresentada foi tão irrelevante, tão... : e daí, que serve para isso? Meu interesse maior estava em adquirir aquele conhecimento, que era muito mais que aquela simples aplicação prática. Então aquilo que foi feito estava denegrindo para mim a imagem do conhecimento que eu queria adquirir. Nem sempre a demonstração de uma aplicação prática vai servir como incentivo. Isso seria uma falácia dizer: certamente será. Não estou dizendo que não é. estou apenas dizendo que para certas pessoas isso não vai servir como incentivo. Talvez sirva como o oposto.

Aliás, como incentivar algo que não conhecemos? Suponhamos que vou ensinar para uma pessoa alguma coisa. Quero incentivá-la a querer aprender aquilo. Não adianta eu dizer: Ah, isto serve para isso ou para aquilo. Ela não sabe o que é. Você pode usar isso como método para incentivar, mas dependendo das crenças que essa pessoa já tem isso não vai adiantar. Então o que a gente está discutindo aqui é sobre qual é o interesse do aluno no curso. Será que todos os alunos que entraram no curso de computação sabem o que é computação, e sabem quais são os objetivos da computação? Sabem qual é o conjunto de tarefas o conjunto de conhecimentos enfim que nós utilizamos e desenvolvemos? Adicionalmente, (isso seria uma pergunta meio vaga, mas acho que deu para compreender) qual é a função da universidade? Foi dito que está mudando. Tudo bem, pode ser que esteja mudando, mas se a função da universidade não é o ensino, em si, de quem é essa função?

Próxima: você deu bastante protecionismo sobre o construtivismo. ou pelo menos eu fiz analogia a isso (desculpe se eu estiver errado) colocando um contraste com as aulas expositivas. Uma pergunta que nem os pedagogos conseguem responder é como avaliar quem é o profissional capaz de fazer um ensino construtivista com adequação. Como fazer para ensinar dessa forma sem ter problemas? Não é uma forma que eu possa dizer que é inferior, mas também não gostaria de aceitá-la como superior, inquestionavelmente. Um dos problemas que vou citar, sobre aula expositiva, é que se não for passar sobre alguma coisa que o aluno não entendeu talvez a gente fique com o maior conhecimento saturado pelo pior aluno. Isso não me parece agradável. É melhor fazer o pior aluno repetir a matéria que fazer aquele conhecimento, que é o pior, adquirir-se no todo.

Você também falou sobre a avaliação por prova e criticou, mas será que a prova realmente não avalia direito? Porque fazer um trabalho você pode muito bem copiar da Internet, ou mandar outra pessoa fazer. Talvez exista outra forma de avaliação. Se houver, eu peço que diga, porque estou muito interessado em conhecê-la.

A próxima coisa é Ítalo falou sobre ciência área de humanas, e eu queria só citar que não pode ser ciência exata se é ciência humana se é ciência biológica. Todo esse caminho é em direção à objetividade. Eu posso virar aqui e fazer um discurso imenso sobre a palavra amor. É um sentimento, isso é um conhecimento objetivo. Eu sei que é um sentimento. Qual? Vamos investigar filosoficamente o que é, mas não estamos perdendo a objetividade. A partir do momento que a gente perde a objetividade a gente vai fugir da ciência. Estamos partindo para um pseudo-ceticismo, e isso não é bom dentro da universidade. Em instante algum alguém disse que está fazendo isso, porém essa aparentemente é uma tendência, que eu vejo principalmente nos alunos. Que nós estamos direcionando a eles a intuição deles, ao mercado, o que fazer? porque nesse instante a gente foge do principal da ciência. O conhecimento, filosoficamente, básico, nós não temos, a maioria. Talvez nem eu saiba direito o que seja. Está certo, estudei o que é a *priori*, a *posteriori*, Kant, Aristóteles e Cia.,

mas nem todos estudaram tudo isso, nem todos sabem a diferença do a *priori* do a *posteriori*. Nem todos sabem que as três ciências formais são lógica, matemática e computação. Por que computação está aqui como ciência formal e não como uma ciência natural como a física? Todas essas questões deveriam ser de conhecimento pleno dos nossos alunos.

Bem, acredito que é isso. já temos muitas perguntas. obrigado.

Ítalo:

Eu gostaria de fazer alguns comentários. Seu nome é mesmo? Danilo. Danilo, muito bacana a sua reflexão, seu ponto de vista, sua perspectiva das coisas. Revela mesmo uma diferenciação. Em relação às diversas colocações que você fez, e que eu consegui anotar algumas delas aqui, eu iniciaria por uma relacionada com o papel da instituição de nível superior. Se não é ensino, qual é? No meu entendimento, era exclusivamente o ensino, era.

Danilo:

Desculpe, minha pergunta era: se esse papel não é da universidade, de quem é? Eu não duvido que haja outros papéis para a universidade, o que eu queria saber é de quem seria o papel do ensino em si.

Ítalo:

Não, perfeito. Estou dizendo apenas o seguinte: o que eu vejo é que a universidade deixou de ser apenas um centro de ensino. Ela acaba sendo, na minha visão, o local no qual a gente encontra educadores e pessoas que estão sendo educadas, e isso envolve ensino inclusive. Então o papel que eu vejo, a responsabilidade, continua sendo do corpo docente. mas não exclusivamente ensinar.

Eu vejo que o professor tem como papel complementar os dias de hoje, e isso tem uma implicação direta em relação à outra pergunta: em relação à aula expositiva. O modelo de aula expositiva supõe que o papel preponderante do professor é ensinar. Eu percebo hoje que não se reduz apenas a essa técnica - aula expositiva - a técnica que você tem que empregar. Se ele está assumindo o papel de educador, algumas aulas poderão ser expositivas, mas não todas. Eu vejo que algumas aulas tem que ser aulas que sejam encontros, um dos quais seria do tipo aula, e outros, do tipo conversa: vamos conversar, "O que você pensa sobre tal coisa?" sem entrar obrigatoriamente em uma rota filosófica, subjetiva, mas o educador se comprometendo a conduzir um diálogo que em algum momento leve a uma conclusão objetiva. Então eu vejo assim: não apenas aula expositiva, como a gente acaba encontrando na maior parte dos casos. Eu vejo a necessidade de haver encontros como este aqui, por exemplo. O encontro seria neste estilo. Um bate-papo, uma conversa de pontos de vista, eu vejo que este tipo de encontro com os alunos de graduação faz com que o elemento afetividade, o elemento valor humano, ele acaba também se sobressaindo junto com o conhecimento que é comunicado.

Em relação ao modelo de avaliação, estou numa sinuca de bico junto com você. Nós temos aí os modelos clássicos, os trabalhos, provas, e eu entendo que esse tipo de modelo de avaliação hoje já não é mais apropriado. Surge a sua pergunta: qual é então? E eu te respondo: não sei. Estamos aí de volta às falas do professor Jairo: o que a gente está fazendo? Porque a gente precisa encontrar novos modelos de avaliação, que procurem olhar para o processo de aprendizagem, e não exclusivamente para saber se o aluno adquiriu certo grau de conhecimento. Eu acho que o foco primário está no processo de aprendizagem, Então como é que eu faço uma acomodação entre o modelo de avaliação e o processo de aprendizagem? Por exemplo, aula expositiva vai ter um modelo de avaliação, uma aula conversacional, uma aula grupal, uma aula de encontros com similares, exigirão modelos de avaliação diferenciados. Quais serão esses modelos? Precisamos pensar no lado operacional, precisamos pensar no objetivo, preciso quantificar o conhecimento que o aluno adquiriu, de alguma forma, porque é assim que a gente diz se está aprovado ou não, mas eu vejo que em relação a essa pergunta, mais especificamente, eu me sinto bloqueado, eu não sei dizer exatamente, eu não sei como responder essa pergunta.

Ricardo:

Algumas das questões que você colocou, acabam se reportando também ao aluno que entra na universidade. Então, só para lembrar, não sei se você estava na hora em que o professor Amaury fez a pergunta, ele falou que vários alunos entram, e alguns deles não são capazes sequer de ler e compreender aquele texto que eles estão lendo. Este tipo de problema acarreta essas questões que você levantou. Qual é o papel do aluno dentro de um curso? Essa é a primeira, anterior mesmo às suas. O que ele está fazendo aqui? Aí depois vem: o que é o curso dele? Como é que a gente pode enquadrar isso dentro do perfil que se espera desse aluno posteriormente? Então na verdade, a gente vai acabar voltando ao ensino médio. E aí no ensino médio a gente já sabe o que aconteceu e está um pouco fora do âmbito da discussão, mas para não deixar sem comentar, eu acho que é importante. Mesmo os alunos diferenciados, alunos que a gente pega em vestibular com trinta candidatos por vaga, (estamos quantificando), mas se todos têm uma formação deficiente no ensino médio, então não adianta nada. Então a gente acaba recebendo (bom pelo menos para nós aqui não é tão mau, porque não recebemos analfabetos funcionais). Os nossos alunos sabem ler. Mas vêm com alguns desses problemas que você comentou. Isso é fato. É raro o aluno que acaba, por vontade própria, ou talvez por incentivo de professores, a buscar alternativas. Ler outras coisas como você mesmo falou que fez, está certo?

Amaury:

Só para fechar a questão da avaliação. Eu acho que eu também não tenho resposta. Mas eu tenho uma alternativa, que eu tenho utilizado nos últimos anos. É aquela questão do "vou conquistar a confiança do aluno para que ele não me veja como seu inimigo, já que eu vou avaliá-lo". Ele não pode, em momento algum, me ver como inimigo, E normalmente a visão do aluno quando ele chega à universidade: aquele é o cara que vai me avaliar? Vou furar o pneu do carro dele! É aquele que eu tenho que dar uma rasteira. E conquistar o aluno num primeiro momento faz com que você obtenha alguns resultados um pouco melhores. Por exemplo, uma consciência maior da tarefa, do papel dele dentro da universidade, de modo que quando ele for mal em sua prova, ele bate nas suas costas entregando a prova, e fala assim: professor, desculpe, eu fui

mal em sua prova, mas eu vou me recuperar na próxima. Então eu acho que este é um ponto crucial e fundamental para que você possa obter um resultado melhor da avaliação. Não existe uma forma, eu acho, mas eu acho que parte daquela questão da humanidade.

Ricardo:
Então eu acho que nós vamos encerrar 'mesmo' agora. Obrigado a todos.

****

PREMIAÇÃO

Foram nessa ocasião distribuídos três prêmios:
Um para membros da platéia presentes ao encerramento, um para os autores de trabalhos presentes ao encerramento, e outro, para o trabalho mais pontuado entre os revisores que fizeram a seleção.
O melhor trabalho foi concedido à professora Cláudia e Rosalía, que apresentaram o melhor trabalho neste workshop.
O segundo, sorteado, foi para autores e apresentadores de artigos que estavam assistindo à mesa redonda.
O terceiro, também sorteado, foi para a um membro da platéia.

****

ENCERRAMENTO

João:
Gostaria de dizer que estou satisfeitíssimo com o que aconteceu este ano, realmente nós tivemos uma quantidade muito boa de trabalhos com nível excelente, tivemos um número de submissões bastante alto também, e esperamos que o ano que vem possamos repetir isso, e até ampliar. Aqueles que ainda não publicaram no WTA, aqueles que ainda não escreveram artigos para submeter ao evento estão convidados a submeter, não fiquem chateados se não forem selecionados, mas existem alguns critérios objetivos para a seleção dos artigos. Estamos fazendo o possível para que este evento ganhe cada vez mais vulto e credibilidade como foi mencionado anteriormente, e também solicitado pelo nosso diretor na abertura. Então para conseguirmos obter um êxito em uma tarefa como essa não é fácil, todas as pessoas que estão aqui estão por diletantismo, e a maior parte são, alguns (meia dúzia) que estavam coordenando, que estavam organizando este evento que tiveram mais uma motivação pessoal direta para trabalhar nisso, mas a massa – eu estive contando o número de pessoas ao longo do evento e percebi uma população média de quarenta e cinco a cinqüenta e cinco pessoas flutuando durante o dia, nós temos a platéia que está aqui, agora, que é quase totalmente diferente daquela que iniciou a menos de uma população fixa de pessoas que ficaram o tempo todo aqui – houve uma flutuação grande, e eu estimo que tenhamos recebido umas cento e vinte pessoas ao todo (eu não tenho os números oficiais) e nós temos apenas o registro das pessoas que passaram pela secretária, e se registraram, então, pode haver um pequeno erro nisso, mas independente de qualquer coisa, foi mais um evento bem sucedido graças ao esforço de cada um que está aqui. Eu queria fazer um agradecimento especial às pessoas de fora de São Paulo especialmente às pessoas do exterior que vieram para cá para prestigiarem o nosso evento. Isso não tem preço e gratifica o trabalho que estamos fazendo, e que nos incentiva a repetir essas ocasiões no próximo ano novamente.
Então vocês estão todos convidados para o próximo ano, de preferência com contribuições também.
Então eu queria mais uma vez agradecer a todos pela presença, dizer que pessoalmente gostaria de dar um grande abraço em cada um, essa parte emocional para mim é muito importante, é eu me sinto amigo de fato de vocês antes de qualquer coisa.
E essa parte afetiva, como disse o Danilo, é o que mais interessa.

E eu queria, para completar, – eu sei que fui proibido – o professor Ricardo disse: definitivamente terminou, eu queria lembrar só duas coisinhas, sobre aquele debate que estava acontecendo aqui. Um deles: provas - acabou o problema das provas se o professor - não o aluno - o professor - considerar a prova como um meio de ensinar, e não de cobrar aquilo que foi ensinado. É uma forma de motivar o aluno a, pela última vez, estudar uma determinada matéria. Então eu ponho essa proposta, que eu costumo empregar, e é uma coisa que eu tenho feito para tentar modificar um pouco essa situação. Eu faço das provas uma ocasião de estudo, em que o aluno vai aprender. Se ele não aprendeu até lá ele tem ainda a chance de aprender durante a prova. Então se o educador se conscientizar de que isso é possível, então podemos mudar até a maneira como o aluno enxerga a prova.
Outra preocupação minha de longa data é sobre o futuro dessa nação, do ponto de vista tecnológico, científico, etc. Quem é que ministra as aulas? Naturalmente a universidade investe o professor de um poder especial, de ensinar. Ele pode não ensinar nada, mas ele tem esse poder oficial. O que é que esse professor consegue passar para os seus alunos? No máximo, aquilo que ele sabe. Então, duas coisas: um cuidado da universidade de indicar, para ensinar determinados assuntos, professores que realmente entendem desses assuntos. Isso pode gerar muita polêmica, mas, ouvindo os alunos, e fazendo algumas avaliações pessoais de algumas coisas que acontecem a gente vê que nem sempre isso acontece, pois nem sempre o professor que está ministrando uma disciplina tem o domínio daquele assunto que seja suficiente para poder ensinar. Como é que se resolve isso? Fazendo com que esse professor tenha um treinamento suplementar. Essa é uma maneira de fazer isso, e isso pode ser feito pessoalmente, aprendendo por conta própria, e alguns que têm outros interesses, que não tanto de ensinar os alunos, e este não vai se interessar em fazer isso. E vai continuar a ser um professor medíocre. Nós professores não podemos ser medíocres, porque nós queremos ser educadores, nós temos que fazer da nossa aula uma ocasião em que compartilhamos um pouco o conhecimento com os alunos. Então temos que nos instruir, se percebermos que não entendemos de alguns assuntos. Isso é um ponto crucial.
O outro ponto que eu queria tocar, para encerrar definitivamente : quanto tempo leva para formar um professor? Quanto tempo leva para eu, sem saber muitas

coisas de uma área grande como essa, quanto tempo leva para eu me instruir o suficiente para poder ser um bom professor dessa matéria ou disciplina? Eu tenho que aprender assimilar o conhecimento, e tenho também que ter uma coisa importante, que foi mencionada aí, que é a prática. Eu tenho que saber atuar naquela área, eu tenho que saber entender o que eu estou falando, e aplicar com propriedade as coisas que eu quero passar para os outros. Então isso é um processo lento, é um processo trabalhinho de formiga, que leva muitos anos. Formar uma boa equipe de professores, em menos de quinze anos, é muito difícil: é preciso até formar uma mentalidade. Da maneira como as coisas estão correndo, – e isso foi o que motivou a escolha desse tema para esse debate – é que por causa dessa deterioração daquilo que a gente considera um ensino de excelência, e isto está acontecendo mundialmente, e não só aqui, por motivos diversos, que não vem ao caso mencionar de novo aqui, porque já foi falado no debate, nós estamos perdendo pessoas que estão bem formadas hoje em dia. Já faz no mínimo uns cinco ou dez anos que o processo de ensino está em decadência, Estamos perdendo as pessoas que entendem as coisas fundamentais, que seriam essenciais para que o aluno pudesse bem exercer sua profissão. Então, quanto tempo vamos levar a partir do momento em que se esgotam os recém formados capazes de assumir uma posição de professor e educador? Quanto tempo leva para formarmos uma pessoa dessas - já que não há outras - para que eles venham ensinar as próximas turmas? Para evitar que um cego guie outro cego. Porque uma pessoa que nunca foi submetida a certo conhecimento, que não sabe nem mesmo que aquilo existe, não tem a mínima idéia se aquilo pode ter importância ou não. Certamente para ele aquilo não tem importância. Eu já ouvi alguém falando que por não ter um determinado diploma aquilo não o impede de exercer um cargo alto. Nós podemos – coincidências aí acontecem – extrapolar isso para os nossos casos. Então se não tivermos esse cuidado de consertar, nem que seja a nosso modo, – mais um grãozinho da formiga – e procurar consertar essa situação, nosso país, dentro de uns 10 anos, estará sem pessoas capazes de dar aulas do que quer que seja que tenha um mínimo de profundidade. Nós vamos ter uma porção de gente com muito conhecimento pontual, mas que não têm lastro fundamental nenhum. E isso é uma coisa muito preocupante. Então eu deixo mais isso para que todos se preocupem e procurem encontrar soluções. Espero, de coração, que nós, o Brasil ache um caminho para essas coisas, porque se não, nós vamos para um buraco. Nosso ensino médio já faliu, o primário nem se fala, e o superior, se não tomarmos cuidado, também vai afundar. O secundário está em um processo de tentativa de recuperação, pelo menos as notícias dizem isso, então vamos ver o que vai acontecer com o ensino médio, vamos ver se ao menos recupera um pouco, e se nós podemos fazer alguma coisa para que a repercussão desse decaimento do ensino médio não nos afete tanto, e que possamos recuperar a nossa parte também.

Então um grande abraço em todos, e até o ano que vem. Muito obrigado.

****

Este texto foi inicialmente transcrito, traduzido para o português nas partes originalmente em espanhol, e pré-elaborado por João José Neto a partir das gravações do evento.

Foi posteriormente revisado, corrigido e editorado, para ser colocado nesta sua forma definitiva por Ricardo Luis de Azevedo da Rocha.

****