Linguagens de Programação aderentes ao Paradigma Adaptativo

A. V. Freitas e J. J. Neto

***Resumo*—An adaptive device is made up of an underlying mechanism, for instance, an automaton, a grammar, etc., to which is added an adaptive mechanism that is responsible for allowing a dynamic modification in the structure of the underlying mechanism. Adaptive languages have the basic feature of allowing the development of programs that self-modify through adaptive actions at runtime. The conception of such languages calls for a new programming style, since the application of adaptive technology suggests a new way of thinking. The adaptive programming style may be a feasible alternate way to obtain self-modifying consistent codes, which allow its use in modern applications for self-modifying.**

***Keywords*—Adaptive Language, Adaptive Technology, Adaptive Programming style, Adaptive Mechanism.**

I. INTRODUÇÃO

Um dispositivo adaptativo é constituído por um mecanismo subjacente, por exemplo, um autômato, uma gramática, uma árvore de decisão, etc., ao qual acrescemos o que se denomina mecanismo adaptativo, responsável por permitir que a estrutura do mecanismo subjacente seja dinamicamente modificada.

O conceito fundamental que caracteriza um dispositivo adaptativo é a sua capacidade de executar ações adaptativas, as quais podem ser interpretadas como chamadas de procedimentos, internos aos dispositivos, que são ativados em resposta ou reação à detecção das situações que exigem do mesmo alterações comportamentais [1].

Para uma dada configuração corrente e um determinado estímulo recebido em sua entrada, o dispositivo, através da aplicação de alguma das regras que definem seu comportamento, é movido para uma outra configuração.

Descritos por um conjunto finito e bem definido de regras, tais dispositivos iniciam sua operação em alguma configuração inicial predeterminada. As cláusulas que compõem esse conjunto de regras testam a configuração corrente do dispositivo e determinam sua nova configuração.

Aparecido Valdemir de Freitas desenvolve a pesquisa em Linguagens Adaptativas através do IMES – Universidade Municipal de Ensino Superior de São Caetano do Sul – Av. Goiás No. 3400 – Vila Barcelona – São Caetano do Sul – SP - Brasil. (avfreitas@imes.edu.br)

João José Neto é o responsável pelo LTA – Laboratório de Tecnologias Adaptativas, vinculado ao Departamento de Computação e Sistemas Digitais da Escola Politécnica da Universidade de São Paulo – Av. Prof. Luciano Gualberto, trav. 3, No. 158 – Cidade Universitária – São Paulo - SP - Brasil. (joao.jose@poli.usp.br)

Dispositivos adaptativos tiveram sua origem em formalismos simples de usar, os autômatos de estados finitos, de tal modo que com o acréscimo de um conjunto de regras estes tornam-se capazes de representar problemas mais complexos, envolvendo linguagens não-regulares e até mesmo dependentes de contexto [2].

Ao acrescentarmos um mecanismo adaptativo a um autômato de estados finitos, este passa a sofrer inclusões ou remoções de transições e/ou estados durante o processamento, o que aumenta seu poder de expressão [3].

Linguagens adaptativas de programação são dispositivos adaptativos que empregam uma linguagem de programação convencional como mecanismo subjacente.

Com o correr de sua execução, um programa escrito em uma linguagem adaptativa exibirá um comportamento automodificável em decorrência da ativação de suas ações adaptativas.

Da mesma forma que um autômato finito adaptativo incorpora um autômato finito como mecanismo subjacente, uma linguagem de programação adaptativa deve estar atrelada a alguma linguagem de programação que será empregada como mecanismo subjacente [4] [5].

O mecanismo adaptativo presente na linguagem adaptativa fará com que a mesma tenha a capacidade de gerar programas com comportamento automodificável.

II. CÓDIGOS AUTOMODIFICÁVEIS

Muito comum na época em que nasciam os primeiros computadores, devido à escassez de memória e conseqüente necessidade de reaproveitamento da mesma, os programas que possuem a capacidade de se automodificarem, sem intervenção externa, foram pouco a pouco caindo em desuso devido às dificuldades de compreensão de seu código, de desenvolvimento da sua lógica, de verificação de consistência e de depuração de seu código [6][7].

Isso coincidiu com o advento da Engenharia de Software em resposta à famosa Crise do Software dos anos 1970.

Depois de um longo período de estagnação, o código automodificável reapareceu recentemente em diversas situações características da nossa época [8][9].

Entre as principais grandes aplicações em que se tem atualmente utilizado de forma extensiva esse recurso, destacam-se: proteção de programas, compressão de código, engenharia reversa indesejada, otimização de código, sistemas de computação evolucionária, etc. [10][11].

Este artigo pretende contribuir para que se possam desenvolver projetos que utilizem de forma consistente essa antiga prática de programação e dela desfrutem as vantagens.

A proposta deste artigo envolve a utilização da adaptatividade como forma disciplinada de automodificação, aplicada às regras correspondentes às diversas instruções de um programa, conceituando assim as linguagens de programação adaptativas.

III. ESTILO ADAPTATIVO DE PROGRAMAÇÃO

Paradigmas descrevem teorias e procedimentos que, quando utilizados de forma conjunta, podem representar uma forma de organizar o conhecimento [12].

Há uma tendência natural em aceitarmos os paradigmas que condizem com nossa forma de pensar e, conseqüentemente, rejeitarmos aqueles que são novos, diferentes ou que, de alguma forma, conflitam com nossa forma de pensar.

Quanto mais estivermos vinculados a uma dada forma de pensar, mais resistências ofereceremos a quaisquer evidências ou argumentos contrários.

Entre as várias formas de se exteriorizar um paradigma ou forma de pensar, talvez a linguagem seja uma das mais representativas.

A linguagem na qual um pensamento é expresso reflete de forma fundamental a natureza de uma idéia. A associação entre pensamento e linguagens se torna ainda mais crítica se estendermos o conceito para o domínio das linguagens de programação.

Assim, a linguagem utilizada por um programador na resolução de um problema está fortemente relacionada à sua maneira de pensar, ou à sua forma de implementar a solução do problema [13].

Quando o escopo descrito por um dado paradigma contém os itens e relacionamentos que existem no domínio de interesse do problema, a tarefa de modelar uma solução dentro daquele domínio é facilitada.

Por exemplo, o paradigma lógico tende a materializar a solução de um problema através da composição de predicados e relações, enquanto o paradigma funcional enfoca o uso e composição de funções.

Se estivermos de posse de uma linguagem de programação na qual o modelo de arrays possa ser diretamente representado, teremos maior facilidade para a resolução de problemas envolvendo, por exemplo, aritmética de matrizes.

Por outro lado, caso a linguagem de programação a ser utilizada não abstraia diretamente as entidades do domínio do problema (no caso, matrizes), a solução deverá ser implementada através da simulação de tais entidades com a ajuda dos elementos disponíveis na linguagem de programação.

Essa tarefa dificulta o processo de implementação da solução, diminuindo sua expressividade, uma vez que a forma de pensar do programador não estará diretamente mapeada nos construtos disponíveis na linguagem.

Com essas reflexões, podemos inferir que as linguagens de programação, nas quais as soluções dos problemas são escritas, direcionam a mente do programador de forma tal que este trate o problema de acordo com uma determinada visão, imposta pelo paradigma de programação.

É comum nos depararmos com programadores que trabalham há muito tempo com uma determinada linguagem de programação, cujo paradigma está tão fortemente associado à sua forma de pensar que eles quase sempre têm dificuldades em quebrar o paradigma usual e empregar outros paradigmas de programação.

A nova forma de pensar incutida pela tecnologia adaptativa e aplicada à programação automodificável motiva a pesquisa de um novo estilo de programação, uma vez que o comportamento dos programas escritos em linguagens adaptativas dependem da execução de funções adaptativas que estabelecem a forma pela qual o código é dinamicamente modificado.

Em programas nos quais não haja adaptatividade, o código estático não é modificado em tempo de execução, permitindo, em conseqüência, que metodologias clássicas de desenvolvimento de software sejam aplicadas.

Na programação adaptativa, entretanto, as diversas instâncias do código em execução, geradas pelas funções adaptativas, devem ser consideradas.

Dessa forma, programação adaptativa requer métodos não convencionais de programação, as quais devem ser objetos de investigação [14].

IV. CONCEPÇÃO DAS LINGUAGENS ADAPTATIVAS

Neste artigo, a título de ilustração do conceito, empregamos uma linguagem de programação funcional como base para a incorporação da tecnologia adaptativa.

Considerando que o cálculo lambda é a base conceitual do paradigma funcional de programação, adotaremos como núcleo da linguagem proposta uma linguagem que suporte esse cálculo [15][16].

Tal escolha é justificada pela maior proximidade com os conceitos da tecnologia adaptativa, uma vez que o paradigma funcional permite a construção de programas em que o conceito de código dinâmico é natural.

Com essas considerações, partiremos de um núcleo funcional baseado em cálculo lambda que irá operar como um interpretador da linguagem com a qual o usuário codificará seus programas.

Sobre esse núcleo funcional básico, projetaremos uma camada adaptativa extensível, que irá avaliar chamadas de funções adaptativas, responsáveis pela automodificação de código.

Da mesma forma que durante a evolução de um autômato adaptativo ocorrem inclusões e/ou exclusões de transições e/ou estados, como decorrência de chamadas de funções adaptativas, nosso código adaptativo também evoluirá durante a sua execução através da inclusão ou exclusão de linhas de

código, caracterizando dessa forma a dinamicidade do código adaptativo [5].

Formalismos adaptativos materializados em linguagens de programação apresentarão, num instante inicial, um bloco de código que é diretamente processado pelo interpretador da linguagem subjacente, até que ocorra a execução de alguma ação adaptativa especificada no programa representado nesse bloco de código [17].

Com o processamento das ações adaptativas, uma nova instância do programa é obtida e a execução é novamente chaveada para o núcleo funcional, que dará continuidade à execução.

Dessa forma, a linguagem adaptativa será formada pelo espaço de códigos CF_1, CF_2, ..., CF_n, de tal modo que, a partir do código inicial CF_1 e por meio de chamadas de funções adaptativas, seu código evolui para sucessivas configurações CF_2, CF_3,.., CF_n à medida que a execução for sendo processada.

Para que nossa linguagem adaptativa possa ser processada, teremos de criar um ambiente de processamento composto pelo núcleo funcional e por um módulo de controle, representado pela máquina adaptativa, que terá como responsabilidade gerenciar a execução da parte automodificável dos códigos escritos nessa linguagem.

Para que as funções adaptativas possam produzir a automodificação de código desejada, precisamos, de algum modo, endereçar as expressões do programa fonte que sofrerão adaptatividade e alterá-las por meio de chamadas de ações elementares, presentes na camada adaptativa.

Expressões adaptativas são expressões usuais, construídas na linguagem hospedeira, que apresentam chamadas de funções adaptativas (ações adaptativas) e que têm como responsabilidade implementar o comportamento automodificável característico das linguagens adaptativas.

Os seguintes eventos irão ocorrer durante o processamento da linguagem adaptativa:

- O módulo adaptativo efetua a chamada do interpretador subjacente, passando-lhe o código fonte inicial formulado pelo usuário.
- O interpretador subjacente irá iniciar o processamento da avaliação das funções de forma usual até que alguma chamada de função adaptativa ocorra. Caso não haja chamadas adaptativas, o avaliador atuará tal qual um ambiente não-adaptativo.
- O controle é retornado para o módulo adaptativo, que irá providenciar o tratamento da chamada adaptativa.
- Considerando-se que as ações adaptativas são compostas por ações elementares, o módulo adaptativo poderá se utilizar de funções da camada adaptativa.
- Como resultado da execução das ações adaptativas, um novo código fonte é gerado.
- Esse novo código fonte é entregue ao interpretador subjacente, que se encarregará da continuidade da execução do programa adaptativo.
- O processamento irá prosseguir até que alguma outra função adaptativa seja encontrada.

Ações adaptativas materializam a execução de funções adaptativas. Podemos definir uma função adaptativa $\mathcal{F}$ em um código automodificável através da seguinte aplicação:

$$\mathcal{F} : \mathcal{D} \rightarrow \mathcal{D}$$

onde $\mathcal{D}$ representa o conjunto de códigos da linguagem hospedeira empregada como formalismo subjacente da linguagem adaptativa.

V. A CAMADA ADAPTATIVA

As chamadas de funções adaptativas (denotadas por ações adaptativas) irão propiciar as características de automodificação do dispositivo.

Essas funções serão constituídas por chamadas de funções elementares, disponíveis na camada adaptativa, as quais, por meio de operações básicas de consulta, inclusão e deleção de funções da linguagem, proporcionarão a adaptatividade ao dispositivo.

Assim, nossa camada adaptativa será composta pelas ações elementares *query*, *insert* e *delete*, respectivamente responsáveis pelas ações de consulta, adição e eliminação de funções (expressões) da linguagem subjacente.

As funções adaptativas serão definidas por meio da composição dessas ações elementares e correspondem às regras que, em tempo de execução, proporcionarão a adaptatividade do código adaptativo.

Obviamente, para cada problema em particular, extrairemos da camada adaptativa as ações elementares, na quantidade e seqüência convenientes para o problema em questão.

Para generalizar a função de consulta, "*queries*", modeladas por "*pattern matching*", poderão ser especificadas de modo a retornar a lista de funções do programa que atendem a uma determinada consulta.

As funções adaptativas necessitam endereçar os diversos elementos de código do programa fonte do mesmo modo que uma ação adaptativa, ao eliminar ou inserir uma transição num autômato adaptativo, necessita referenciar cada uma das transições ou estados do autômato original.

Tendo em vista que neste trabalho estamos empregando uma linguagem subjacente baseada em expressões, um programa pode se reduzir à chamada de uma única função, a qual é formada pela composição de diversas outras funções.

Considerando que um programa funcional pode ser representado por uma estrutura do tipo árvore, podemos considerar que cada nó dessa árvore representa uma função componente do programa.

Para que nossas funções adaptativas possam efetuar a edição na árvore, deveremos endereçar, através de marcas adaptativas, os diversos nós que serão acessados pelas funções adaptativas responsáveis pela adaptatividade de código.

Com o uso desse esquema de endereçamento, único para cada componente do programa, poderemos especificar

funções adaptativas, que, por meio de chamadas da camada adaptativa, irão acrescentar, retirar ou alterar nós da árvore, gerando, como conseqüência, código adaptativo.

Podemos atribuir uma marca adaptativa a uma determinada função do programa por meio de:

```
> (define (marca_adaptativa func x) (set func x)  x )
  (lambda (marca_adaptativa x ) (set func  x)  x)
```

Por exemplo, seja *prod* uma função que retorna o produto de dois argumentos. Podemos associar uma marca adaptativa *AA* à função *prod* por meio de:

```
> (marca_adaptativa  'AA prod)
(lambda  ( ( * x x ) ) ) )
```

Portanto, analogamente ao processamento de nós de uma árvore, as funções adaptativas farão um "*string-processing*" na *expressão lambda* correspondente ao programa, gerando um novo *string* (ou uma nova *expressão lambda*) aderente às marcas adaptativas endereçadas pelas ações adaptativas.

VI. EXPERIMENTOS

Neste artigo, conforme já mencionado, empregamos uma linguagem funcional, com características de extensibilidade, para a incorporação dos mecanismos do formalismo adaptativo [15][16].

A escolha do paradigma funcional não restringe a concepção de linguagens adaptativas atreladas apenas a esse paradigma. Uma vez assegurada a exeqüibilidade das linguagens adaptativas, outras pesquisas podem ser iniciadas com vistas à investigação do emprego delas em outros paradigmas de programação.

O estilo funcional de programação, também conhecido por programação aplicativa ou programação orientada a valor, tem como características a programação com ausência ou diminuição do uso de atribuições e operação com altos níveis de abstração [18].

Em geral, as linguagens de programação são subdivididas em linguagens orientadas a expressões e linguagens orientadas a comandos (*statements*).

Expressões correspondem a construtos de linguagens de programação cujo propósito é a obtenção de um valor, fruto de uma avaliação.

Usualmente, expressões aparecem ao lado direito de operações de atribuição e podem ser de vários tipos, tais como: booleanas, relacionais ou aritméticas.

Comandos usualmente podem alterar o fluxo de controle (*if*, *loop*, *goto*, chamada de *procedures*, etc.) ou podem representar alterações de estado (memória) da máquina, as quais são processadas por comandos de atribuição.

Assim, comandos são empregados com o objetivo de promover alguma alteração, seja no fluxo de controle ou no estado da memória primária.

Expressões e comandos guardam uma importante diferença. Ao se trabalhar com comandos, a ordem na qual as operações são efetuadas afeta o resultado final.

Por exemplo, a seqüência de comandos: a = a + 3; b = a + b; tem um efeito diferente da seqüência: b = a + b; a = a + 3; .

Expressões, no entanto, guardam uma propriedade na qual o valor de uma expressão, ou seu significado, depende unicamente dos valores de suas subexpressões.

Essa propriedade habilita o programador a construir uma estrutura composta de subestruturas mais simples e independentes.

Com isso, é possível projetar a estrutura de um programa de forma mais aderente à estrutura do problema a ser resolvido [19].

Essa propriedade pode ser melhor entendida se representarmos uma expressão numa estrutura do tipo árvore.

Em nossos experimentos, estamos considerando que o núcleo funcional atuará de forma interativa. Dessa forma, o usuário sempre define uma expressão, que é avaliada pelo interpretador. Como resultado da avaliação, uma nova expressão é retornada ao usuário.

Uma expressão, em geral, é uma lista cujo primeiro elemento seleciona uma função e cujos elementos seguintes correspondem aos seus argumentos. Cada expressão deverá retornar um valor.

Neste trabalho, estamos considerando uma linguagem hospedeira (utilizada como dispositivo subjacente) que permita a definição de funções pelo usuário (*define*), que apresente construções de decisão (*if*), de iteração (*while*), de agrupamento de blocos (*begin*), de atribuições (*set* e *setq*), de desvio (*go*), de avaliação de expressões (*eval*), de tratamento de listas, bem como ofereça suporte a expressões lógicas, aritméticas e relacionais [17].

Assim, nosso núcleo funcional será semelhante às clássicas linguagens funcionais Lisp/Scheme [18][19][20]. A razão para essa escolha decorre naturalmente da utilização dos recursos nativos da linguagem hospedeira (Scheme/Lisp), facilitando a obtenção de protótipos que possam ser rapidamente testados quanto aos conceitos adotados [22].

Em futuro próximo, pretende-se reavaliar essa escolha, em função dos resultados concretos que se apresentarem nestas primeiras experiências.

VII. CONCLUSÃO

Códigos automodificáveis, especialmente os empregados em linguagens de baixo nível, são usualmente difíceis de serem escritos, documentados e mantidos [23].

Nossa proposta, no entanto, está centrada no emprego da tecnologia adaptativa, a qual utiliza para isso funções adaptativas. Estas devem ser projetadas por meio de regras que podem, se bem projetadas, assegurar maior usabilidade ao mecanismo proposto.

Obviamente, nas técnicas adaptativas, há o perigo de a automodificação de código gerar uma explosão de espaço de código e de tempo, portanto, convém impor disciplinas sobre as operações executadas pelas ações adaptativas para que as alterações delas decorrentes possam ser previsíveis e controladas.

Para isso, é desejável que um estudo de formalização do crescimento de código seja desenvolvido para quantificar o

tempo e o espaço do código produzidos pelas sucessivas aplicações das ações adaptativas.

Essas limitações, acompanhadas das respectivas funções analíticas de crescimento de código e de tempo, devem fazer parte de um método de projeto de funções adaptativas, a ser desenvolvido em trabalhos futuros.

Para auxiliar o projeto das funções adaptativas, é recomendado que o ambiente adaptativo disponibilize ao projetista ferramentas de depuração do código dinamicamente alterado nesse tipo de programas, ferramentas essas que deverão efetuar todo o controle das regiões alteráveis do código, indicando, de forma também dinâmica, as partes do programa sujeitas à adaptatividade, aquelas que já foram modificadas, a modificação em curso, etc.

Neste trabalho apresentamos os aspectos básicos para o projeto e a implementação de um ambiente responsável pela gerência de execução de códigos escritos em linguagens adaptativas.

Esse ambiente terá como objetivo principal o tratamento do chaveamento de contexto entre o mecanismo subjacente e o módulo adaptativo, no qual todos os requisitos para a execução do código adaptativo deverão estar assegurados, tais como o gerenciamento de escopo de variáveis, definição de marcas adaptativas para endereçamento de funções e chamadas de funções adaptativas.

Para assegurarmos que a linguagem adaptativa proposta esteja aderente ao formalismo adaptativo, será necessário que se especifique, formalmente, a sintaxe e semântica dos construtos associados à camada adaptativa da linguagem.

REFERÊNCIAS

[1] Neto, João José – Adaptive Rule-Driven Devices – General Formulation and Case Study**.** Lecture Notes in Computer Science. Watson, B.W. and Wood, D. – 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] Pistori, Hemerson – Tecnologia em Engenharia da Computação: Estado da Arte e Aplicações. Tese de Doutorado – Escola Politécnica da Universidade de São Paulo – 2003.

[3] Neto, João José – Adaptive Automata for Context-Sensitive Languages. SIGPLAN NOTICES, Vol. 29, n. 9, pp. 115-124, September, 1994.

[4] Freitas, A. V. - Neto, João José – Adaptive Device with underlying mechanism defined by a programming language - 4th WSEAS International Conference on Information Security, Communications and Computers – ISCOCO 2005.

[5] Freitas, A. V. - Neto, João José – Conception of Adaptive Languages – International Conference on Modelling and Simulation – MS – 2006 – May 24 –26, 2006 – Proceedings of the 17th IASTED International Conference – Montreal, QC, Canada.

[6] Anckaert B., Madou M. and Bosschere K.D. – A model for Self-Modifyng Code – Ghent University – Proceedings of the 8th Information Hiding Conference, 2006 .

[7] Anckaert B., Madou M. and Bosschere K.D. – Code (De) Obfuscation – ELIS, Ghent University, Sint-Pieternieuwstraat 41, 9000 Ghent, Belgium.

[8] Cifuentes C., Gough, K.J. – Decompilation of Binary Programs – Software – Practice and Experience, Vol. 25(7), 811-829 – July – 1995.

[9] Madou M., Anckaert B., Moseley P. Debray S., Sutter B.D. – Bosschere K. – Software Protection through Dynamic Code Mutation – Conference International Workshop on Information Security Applications – WISA 2005, LNCS 3786 pp 194-206.

[10] Yamamoto L., Tshudin C. – Harnessing Self-Modifying Code for Resilient Software – Proc 2nd IEEE Workshop on Radical Agent Concepts (WRAC), NASA Goddard Space Flight Center Visitor's Center, September 2005.

[11] Giffin J.T., Christodevescu L.K. – Strengthening Software Self-Checksumming via Self-Modifying Code – 21st Annual Computer Security Applications Conference – December 5-9, 2005 – Tucson, Arizona.

[12] Thomas S.Kuhn -The Structure of Scientific Revolutions*,* University of Chicago Press, 1962.

[13] Timothy A. Budd – Multiparadigm Programming in LEDA, Oregon State University, Addison-Wesley Publishing Company, Inc, 1995.

[14] Freitas, A. V. – Adaptive Languages and a new Programming Style – Proceedings of the 6th WSEAS International Conference on Applied Computer Science, Tenerife, Canary Islands, Spain, December 16-18, 2006.

[15] Barendregt, H.P. – The Lambda Calculus: its syntax and semantics – (2nd ed.), North-Holland, 1984.

[16] McCarthy, J. - Recursive Functions of Symbolic Expressions and Their Computation by Machine, Part-I. CACM 3,4 (1960), 184-195.

[17] Freitas, A. V. - Neto, João José – Um ambiente para o processamento de Linguagens Adaptativas – CACIC – Congreso Argentino de Ciencias de la Computación, 2006.

[18] MacLennan, Bruce J. – Functional Programming Practice and Theory – Addison-Wesley Publishing Company, Inc. 1990.

[19] Burge, W. H. – Recursive Programming Techniques, Addison-Wesley, Reading, Mass. – 1975.

[20] Guy L. Steele Jr. – Common Lisp, the Language, second edition. Digital Press, 1990.

[21] Guy L. Steele Jr. and Gerald J. Sussman – The revised report on Scheme, a dialect of Lisp. MIT AI Memo 452, Massachusetts Institute of Technology, January 1978.

[22] Mueller, Lutz – NewLisp User's Manual and Reference V. 7.50 – 2004 – www.newlisp.org.

[23] Philip K. McKinley, Seyed Masoud, Sadjadi, Eric P. Kasten, Betty H. C. Cheng – Composing Adaptive Software – Michigan State University - IEEE Computer Society – 2004.

Aparecido V. Freitas Professor e Coordenador dos cursos de Computação do IMES – Universidade Municipal de São Caetano do Sul, SP, Brasil, desde 1988. Ex-professor da Universidade Metodista de São Paulo – Brasil. Doutorando em Engenharia Elétrica pela Escola Politécnica da Universidade de São Paulo, Brasil (2003). Mestre em Engenharia Elétrica pela Escola Politécnica da Universidade de São Paulo – Brasil (2000). Bacharel em Engenharia pela Escola de Engenharia Mauá – São Caetano do Sul, SP, Brasil, (1979). Bacharel em Matemática pela Fundação Santo André, SP (1974). Desenvolve através do IMES pesquisa na área de Linguagens Adaptativas.

João J. Neto Graduado em Engenharia de Eletricidade (1971), mestre em Engenharia Elétrica (1975), doutor em Engenharia Elétrica (1980) e livre-docente (1993) pela Escola Politécnica da Universidade de São Paulo, SP, Brasil. Atualmente, é professor associado da Escola Politécnica da Universidade de São Paulo e coordena o LTA – Laboratório de Linguagens e Tecnologia Adaptativa do PCS – 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.