Um ambiente para o processamento de Linguagens Adaptativas de Programação

Aparecido Valdemir de Freitas
IMES - Universidade Municipal de São Caetano do Sul
Av. Goiás N° 3400 – Vila Barcelona – São Caetano do Sul – CEP 09550-051
São Paulo – Brasil - avfreitas@imes.edu.br

e

João José Neto
Escola Politécnica da Universidade de São Paulo
Depto. de Engenharia de Computação e Sistemas Digitais
Av. Prof. Luciano Gualberto, trav. 3, N° 158 - Cidade Universitária
São Paulo – Brasil - joao.jose@poli.usp.br

Abstract

Adaptive devices show the characteristic of dynamically change themselves in response to input stimuli with no interference of external agents. Occasional changes in behavior are immediately detected by the devices, which right away react spontaneously to them. Chronologically such devices derived from researches in the field of formal languages and automata. However, formalism spurred applications in several other fields. Programs with self-modifying code have been avoided since the advent of Software Engineering (1970's) but are nowadays being used in several applications. One way to generate self-modifying programs is by using specially-designed programming languages. Adaptive programming languages are adaptive devices that use an usual programming language as underlying mechanism. A group of adaptive functions, defined through extension of the language, causes the self-modification of the device, resulting in a language with adaptive style. The paper shows design and implementation aspects of an environment to manager the adaptive language execution. With the use of adaptive language, a new programming style is conceived, once its behavior is directly associated to the set of rules that defines it, which change as the code is executed.

Keywords: Adaptive devices, self-modifying devices, adaptive programming language.

Resumo

Dispositivos adaptativos apresentam a característica de se modificarem dinamicamente em resposta a estímulos de entrada, sem interferência de agentes externos. Eventuais necessidades de modificação de comportamento são automaticamente detectadas por estes dispositivos para, em seguida, reagirem a elas de forma espontânea. Historicamente tais dispositivos emergiram das pesquisas na área de linguagens formais e autômatos. No entanto, o formalismo suscitou aplicações em diversas outras áreas. Programas com código auto-modificável, que perderam terreno em conseqüência do advento da Engenharia de Software nos anos 70, voltaram à vida recentemente em aplicações diversas. Uma das formas de programação de código auto-modificável é a utilização de linguagens de programação especificamente projetadas para isso. 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 auto-modificável em decorrência da ativação de suas ações adaptativas. O artigo apresenta aspectos do projeto e implementação de um ambiente para gerenciar a execução de uma linguagem adaptativa. Com o emprego de linguagem adaptiva, um novo estilo de programação é concebido, uma vez que o seu comportamento está diretamente associado ao conjunto de regras que o define, o qual se altera à medida que o código é executado.

Palavras chaves: Dispositivos adaptativos, dispositivos auto-modificáveis, linguagem de programação adaptativa.

1 INTRODUÇÃO

Dispositivos adaptativos caracterizam-se pela habilidade de executar ações adaptativas, as quais podem ser vistas como procedimentos internos aos mesmos e que são ativados em resposta ou reação à detecção das situações que exigem alterações comportamentais. [1]

A operação destes dispositivos é iniciada em uma determinada configuração e prossegue com a aplicação de alguma regra de seu conjunto fixo e finito de regras até que não existam mais estímulos de entrada ou até que se alcance uma configuração à qual nenhuma regra possa ser aplicada. Neste momento pode-se determinar, com base na configuração atingida, se o dispositivo aceita ou rejeita a seqüência completa de estímulos de entrada.

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.

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

A partir dos autômatos adaptativos, outros dispositivos adaptativos foram concebidos e associados a outras áreas com diferentes mecanismos subjacentes, como por exemplo, árvores de decisão[2], statecharts[4], redes de Markov[5], processamento de linguagens naturais[6], gramáticas[7], ambientes multilinguagens[8], robótica[9], aprendizagem computacional[10], tabelas de decisão[11], etc.

2 OBJETIVO

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 à alguma linguagem de programação que será empregada como mecanismo subjacente. [12][18]

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

Ao empregarmos linguagens adaptativas, uma forma própria de pensar ou de se raciocinar deve ser desenvolvida, de maneira natural e espontânea, como conseqüência da incorporação ao código dos mecanismos de auto-modificação presentes na tecnologia adaptativa.

Essa forma de pensar induz um novo estilo de programação, uma vez que o comportamento de um dispositivo adaptativo está diretamente associado ao conjunto de regras que o define, trazendo como consequência alteração de código com o correr da evolução de sua execução.

Essa característica, típica dos dispositivos adaptativos, exige cuidados especiais da parte de quem desenvolve aplicações dessa natureza, envolvendo um estilo de raciocínio e uma disciplina de programação que antecipe os efeitos das ações adaptativas sobre o comportamento do dispositivo.

O emprego de linguagens de programação como mecanismo subjacente de dispositivos adaptativos já foi investigado, resultando como conseqüência a concepção de linguagens adaptativas.

Em relação aos trabalhos já publicados, este artigo acrescenta a apresentação de um ambiente capaz de gerenciar a execução, com auto-modificação de código, de modo que a linguagem adaptativa possa ser convenientemente processada.

Assim, este artigo apresenta aspectos do projeto e da implementação de um ambiente cuja responsabilidade é interpretar e executar as chamadas adaptativas, processar as correspondentes alterações dinâmicas de código, controlar chaveamentos de contexto entre o dispositivo subjacente e o ambiente adaptativo, e executar as demais ações necessárias para garantir a exeqüibilidade da linguagem adaptativa.

Adotaremos como mecanismo subjacente, um dialeto da linguagem funcional Lisp, com escopo dinâmico. [13] [14].

3 O MECANISMO SUBJACENTE

Neste artigo, a título de ilustração do conceito, empregaremos uma linguagem funcional, com características de extensibilidade, para a incorporação dos mecanismos do formalismo adaptativo. [13][14]

A escolha do paradigma funcional não restringe a concepção de linguagens adaptativas atreladas apenas a este paradigma. Uma vez assegurada a exeqüibilidade das linguagens adaptativas, outras pesquisas podem ser iniciadas com vistas a 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. [15]

Em geral as linguagens de programação são subdivididas em linguagens orientadas a expressões e linguagens orientadas a comandos (*statements*). [16]

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: booleanos, relacionais ou aritméticos.

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 afetam o resultado final. Por exemplo, a seqüência de comandos: x = x +2; y = x + y; tem um efeito diferente da seqüência: y = x +y; x = x +2; .

Expressões, no entanto, guardam uma propriedade no qual o valor de uma expressão, ou seu significado, depende unicamente dos valores de suas sub-expressões. Esta propriedade habilita o programador a construir uma estrutura composta de estruturas mais simples e independentes. Com isso, é possível projetar-se a estrutura de um programa de forma mais aderente à estrutura do problema a ser resolvido. [17]

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

A *Figura-1* apresenta a expressão (3pq + 2pr) representada numa estrutura do tipo árvore e nela observamos que cada operação depende tão somente da sub-árvore que a representa. Quando todos os nós abaixo de um determinado nó estiverem decorados com valores, a operação poderá ser

aplicada reduzindo-se assim a expressão e conseqüentemente a árvore que a representa.

Considerando que o *cálculo lambda* é a base do paradigma funcional de programação, adotaremos como mecanismo subjacente, uma linguagem que suporte o *cálculo lambda*. [13] [14]

Esta 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.

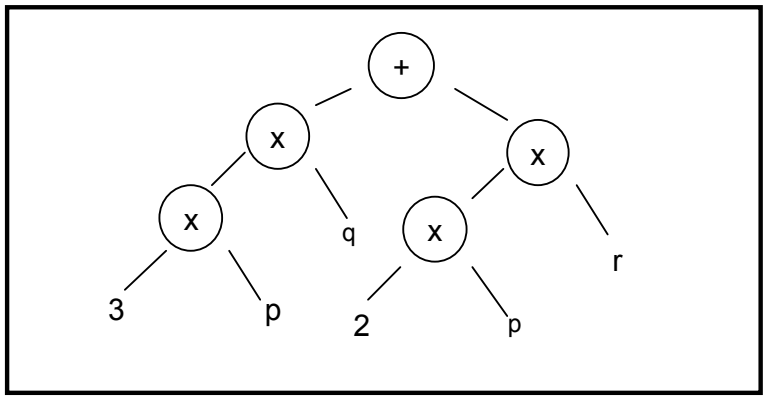

Figura-1 – Representação de uma expressão em árvore

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.

O núcleo funcional atuará de forma interativa. Desta 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.

Assim, nosso núcleo funcional será semelhante à clássica linguagem funcional Lisp. A razão para esta escolha decorre naturalmente da utilização dos recursos nativos da linguagem hospedeira (Lisp), facilitando a obtenção de protótipos que possam ser rapidamente testados quanto aos conceitos adotados. Em futuro próximo pretende-se reavaliar essa escolha, em função dos resultados concretos que se apresentarem nesta primeira experiência.

Com isto, neste artigo dispensa-se a especificação formal da linguagem hospedeira (sintática e semântica) uma vez que as funções adaptativas serão definidas por meio dos construtos disponibilizados pelas linguagem funcional clássica adotada como mecanismo subjacente.

Considerando que a linguagem subjacente empregada neste trabalho é baseada em expressões, qualquer programa escrito nesta linguagem pode se reduzir a uma única expressão (com o primeiro elemento denotando uma função), a qual pode ser composta (aninhada) a partir de diversas outras expressões (nativas ou codificadas pelo usuário).

4 CÓDIGO AUTO-MODIFICÁVEL

Muito comum na época em que nasciam os primeiros computadores, devido à escassez de memória e conseqüente necessidades de reaproveitamento da mesma, os programas que possuem a capacidade de modificar a si próprios, sem intervenção externa, foram pouco a pouco caindo em desuso devido às dificuldades de compreensão de seu código, de desenvolvimento da sua lógica, da verificação de consistência e de depuração de seu código. Isso coincidiu com o advento da Engenharia de Software em resposta à famosa Crise do Software dos anos 1970. [21]

Depois de um longo período de estagnação, código auto-modificável reaparece recentemente em diversas situações características da nossa época. 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.

Alguns trabalhos que têm sido publicados relatam e discutem os problemas decorrentes da utilização dessa técnica. [22]

Outros trabalhos procuram caminhos para a utilização segura desse recurso e propõem mecanismos para isso. [21][23] [24] [26]

Nosso trabalho insere-se na segunda dessas categorias, e pretende contribuir para que se possa desenvolver projetos que utilizem de forma consistente essa antiga prática de programação e dela desfrutar as vantagens.

Acrescentam-se a esses trabalhos publicados em âmbito internacional, diversos outros da recentemente desenvolvida linha de pesquisa sobre "adaptatividade" [25], segundo a qual propõem-se aumentar o poder computacional de formalismos usuais orientados por regras, acrescentando-se-lhes através de ações adaptativas (ver seção 1) o poder de se auto-modificar espontaneamente.

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

5 OPERAÇÃO DO CÓDIGO ADAPTATIVO

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 ações adaptativas, nosso código adaptativo também evoluirá durante a sua execução através da inclusão ou exclusão de funções, caracterizando dessa forma a dinamicidade do código adaptivo. [12]

Formalismos adaptativos materializados em linguagens de programação (em nosso caso, uma linguagem funcional) apresentarão, num instante inicial, um bloco de código que é diretamente processado pelo interpretador do núcleo funcional, até que ocorra a execução de alguma ação adaptativa especificada no programa representado neste bloco de código. [18]

Com o processamento das ações adaptativas, uma nova instância do programa é obtida (no caso funcional), e a execução é novamente chaveada para o núcleo funcional que dará continuidade à execução.

Desta 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 funcional 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 auto-modificação de código, 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.

Essas ações elementares, em tempo de execução, efetivarão a inclusão, exclusão ou alteração das expressões, que sejam convenientes ao particular problema em questão.

Tendo em vista que as expressões correspondentes aos códigos dos programas escritos na linguagem subjacente, apresentam-se como estruturas em árvore, podemos identificar cada nó da árvore com as respectivas aberturas de parênteses -- que indicam chamadas de funções componentes – por meio de rótulos (*labels*) que permitirão efetuar as referências.

Obviamente, no projeto das funções adaptativas somente será necessário vincular rótulos às expressões que realmente participarão da auto-modificação do código.

Dessa forma, por meio de rótulos associados às diversas funções (nativas ou definidas pelo usuário) que compõem o código do usuário, podemos projetar as funções adaptativas -- responsáveis pela auto-modificação do programa -- as quais terão um comportamento semelhante aos procedimentos clássicos de edição de árvores. [20]

Uma vez definido o mecanismo de rótulos da linguagem adaptativa, para a escrita das funções adaptativas será necessário que se tenha à disposição um conjunto de funções de biblioteca que possibilite incluir novos códigos ou eliminar/atualizar códigos existentes. Estas funções de biblioteca farão parte da camada adaptativa da linguagem.

Sendo nosso núcleo básico funcional um interpretador do *cálculo lambda*, qualquer função definida no programa poderá ser representada por uma chamada em *cálculo lambda*. [13]

Portanto, analogamente ao processamento de nós de uma árvore, em nosso modelo de linguagem adaptativa, as funções adaptativas farão um "*string-processing*" na *expressão lambda* correspondente ao programa, gerando um novo string (ou uma nova *expressão lambda*) aderente aos rótulos (*labels*) endereçados pelas ações adaptativas.

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:

1. O módulo adaptativo efetua a chamada do interpretador funcional baseado em *cálculo lambda*, passando-lhe o código fonte inicial codificado pelo usuário.
2. O interpretador funcional 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 funcional não adaptativo.
3. O controle é retornado para o módulo adaptativo que irá providenciar o tratamento da chamada adaptativa.
4. Considerando-se que as ações adaptativas são compostas por ações elementares, o módulo adaptativo poderá se utilizar de funções da camada adaptativa.
5. Como resultado da execução das ações adaptativas, um novo código fonte é gerado.
6. Este novo código fonte é entregue ao interpretador funcional que se encarregará da continuidade da execução do programa adaptativo.

6 PROJETO DAS FUNÇÕES ADAPTATIVAS

As chamadas de funcões adaptativas (denotadas por ações adaptativas) irão propiciar as características de auto-modificação do dispositivo. Estas 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 destas 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 atendessem a uma determinada consulta.

Para que as funções da camada adaptativa possam efetuar suas tarefas, precisamos de algum modo endereçar as funções que compõem nosso 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 rótulo irá corresponder a um nó desta árvore.

Com o uso deste 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.

Para que nossas funções adaptativas possam efetuar a edição na árvore, deveremos endereçar, através de rótulos, as diversas funções que participarão da adaptatividade de código.

Podemos atribuir um rótulo a uma determinada função do programa, por meio de:

```
> (define (label  rotulo x) (set rotulo x)  x )
(lambda (rotulo x ) (set rotulo x)  x)
```

Por exemplo, seja *prod* uma função que retorna o produto de dois argumentos. Podemos associar um rótulo *R1* à função *prod*, por meio de:

```
> (label 'R1 prod)
(lambda  ( ( * x x ) ) ) )
```

Com isso , a função *prod* estará associada ao rótulo *R1*.

```
> R1
(lambda  ( ( * x x ) ) ) )
> (R1  5)
25
> ( (lambda ( ( * x x ) ) ) 5)
25
```

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 aos rótulos (*labels*) endereçados pelas ações adaptativas.

7 PROJETO DO MÓDULO ADAPTATIVO

O código adaptativo será processado por um módulo adaptativo que irá solicitar do usuário o texto fonte correspondente à instância inicial do programa e seus parâmetros.

Este texto fonte será salvo numa área do módulo adaptativo, a qual chamaremos de *buffer*. A área relativa aos parâmetros será denotada por *param*.

Assim, em tempo de inicialização do módulo adaptativo será processada a seguinte operação de *save*:

```
(set  'buffer (load "fonte.lsp"))
(set  'param (load "param.lsp"))
```

A princípio, o módulo adaptativo irá interpretar a instância de código inicial, gerando o seu equivalente em *expressão lambda* e atualizando a variável *buffer*.

Após a tradução para *expressão lambda*, o módulo adaptativo passará o controle ao interpretador da linguagem subjacente, por meio da função (eval buffer param) .

Neste instante temos um chaveamento de contexto para o interpretador do mecanismo subjacente o qual irá interpretar o código da forma usual com que é tratado um código funcional.

O processamento irá prosseguir até que alguma função adaptativa seja encontrada. Caso esta função não seja avaliada, o código não será adaptativo, uma vez que nenhuma função adaptativa foi especificada para o ambiente.

O módulo adaptativo deverá possuir um manuseador de funções adaptativas que será responsável pelo chaveamento de contexto do módulo subjacente para o módulo adaptativo e processar a função adaptativa responsável pela alteração de código. Denotaremos este manuseador por *eval_adap*, o qual terá como parâmetro a função adaptativa e seus parâmetros.

Como resultado da execução do manuseador de funções adaptativas, um novo código fonte será produzido e gravado na área *buffer* do módulo adaptativo.

Para ilustrarmos a operacionalidade do manuseador de funções adaptativas, vamos considerar um programa composto de três rótulos *A*, *B* e *C* e uma função adaptativa *adap1* que em um determinado ponto do programa irá eliminar o rótulo B e dar seqüência à execução do código adaptativo. Consideraremos inicialmente, para simplicidade da explanação, que o nosso código adaptivo, não apresente efeitos colaterais representados por expressões de atribuição. A *Figura*-2 representa esquematicamente o nosso código adaptativo inicial.

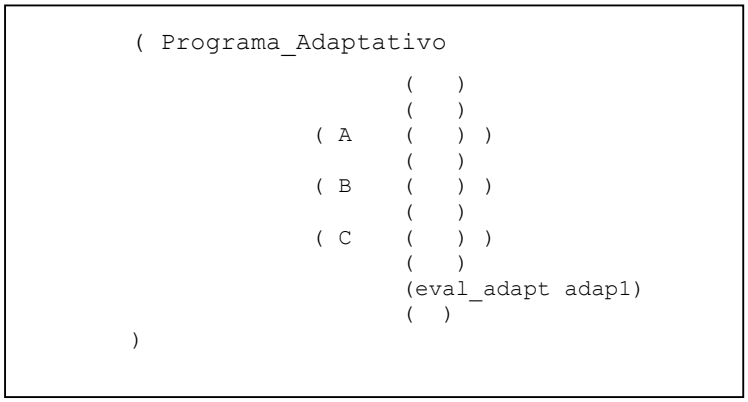

Figura-2 Esquema de um código adaptativo

Neste exemplo, admitiremos que a função adaptativa *adap1* irá eliminar o rótulo *B*, e portanto, todo o código referenciado pelo mesmo. Quando o interpretador subjacente encontrar uma chamada de

função adaptativa, o controle será retornado ao módulo adaptativo que se encarregará de processar o chaveamento de contexto. Neste instante, o módulo adaptativo irá processar a função adaptativa de acordo com as regras associadas aos rótulos do código.

Uma vez processada a função adaptativa, o controle deverá ser retornado ao interpretador subjacente no ponto imediamente após a chamada adaptativa, e para isso o módulo adaptativo deverá salvar a posição conveniente ao retorno e incluir os devidos rótulos para garantir o fluxo normal de execução.

Considerando que o rótulo de controle inserido no código seja ROT1, a *Figura*-3 apresenta um esquema do código após o chaveamento de contexto.

```
        ( Programa_Adaptativo_2
                      (GO ROT1)
                      (   )
                      (   )
              ( A     (   ) )
                      (   )
                      (   )
              ( C     (   ) )
                      (   )
                      (eval_adapt adap1)
              ROT1    (   ) )
        )
```

Figura-3 Esquema após chaveamento de contexto

No exemplo acima descrito, estamos considerando que o tratamento da função adaptativa está no mesmo escopo do programa principal.
Em havendo chamadas de funções adaptativas em escopos mais internos, o módulo adaptativo necessitará processar uma busca em profundidade para processar a correta inclusão dos rótulos de retorno.

Caso o nosso código adaptativo tenha expressões impuras (expressões com chamadas de funções de atribuição), o módulo adaptativo deverá guardar um histórico das variáveis referenciadas para garantir o acesso a estas variáveis, por meio de efeitos colaterais, mesmo após o processamento das funções adaptativas.

Para exemplicar esta questão da existência de efeitos colaterias, vamos considerar a *Figura-4* no qual temos a ocorrência das variáveis VAR1, VAR2 e VAR3 por meio de chamadas de funções de atribuição.

```
      ( Programa_Adaptativo
                    (   )
                    (SETQ VAR1 . . .  )
              ( A   (   ) )
                    (SETQ VAR2 . . .  )
                    (   )
              ( B   (   )
                    (SETQ VAR3 . . .  )
                    )
                    (   )
              ( C   (   ) )
                    (   )
                    (eval_adapt adap1)
                    (   )
                 (SETQ VAR3 . . .  )
      )
```

Figura-4 Código adaptativo com expressões impuras

Conforme a *Figura*-4, temos no código adaptativo a corrência de variáveis oriundas de chamadas de funções de atribuição, e portanto, durante o processamento das funções adaptativas, estas variáveis deverão ser mantidas pelo módulo adaptativo.

```
( Programa_Adaptativo_3
              (SETQ  ASSOC_LIST
                     (
                            (VAR1, v1
                            (VAR2, v2)
                            (VAR3, v3)
                     )
              )
              (GO ROT1)
              (   )
              (SETQ VAR1 . . .  )
       ( A    (   ) )
              (SETQ VAR2 . . .  )
              (   )
       ( B    (   )
              (SETQ VAR3 . . .  )
              )
              (   )
       ( C    (   ) )
              (   )
              (eval_adapt adap1)
         (ROT1  (    ) )
              (SETQ VAR3 . . .  )
)
```

Figura-5 Código adaptativo com lista de associação

Para que este tratamento seja implementado, representaremos o ambiente (o contexto com a qual as variáveis estão associadas a valores) numa lista de associação de variáveis a valores.

Esta lista de associação será endereçada pelas instâncias do código adaptativo no correr de sua execução.

A *Figura*-5 esquematiza uma nova instância do código adaptativo com a inclusão do rótulo de retorno e a lista de associação de variáveis.

8 APLICAÇÕES DA TÉCNICA E TRABALHOS FUTUROS

As linguagens adaptativas caracterizam-se pela auto-modificação de código, em tempo de execução, decorrente do processamento de ações adaptativas.

Adaptatividade por modificação de código tem recebido um interesse recente na literatura acadêmica, principalmente na área de prevenção de engenharia reversa indesejada. [21]

Embora a engenharia reversa tenha como objetivo recriar o projeto de software por meio da análise do produto final, ela pode também ser empregada para fins maléficos.

Código auto-modificável é adequado para estas aplicações uma vez que é a sua presença dificulta a ação da engenharia reversa. [22]

Além da área de proteção de software, adaptatividade por modificação de código também pode ser empregada em dispositivos que utilizam regras dinâmicas tais como aprendizagem computacional, inferência, jogos, reconhecimento de padrões, processamento de imagens, compactação de arquivos, etc.

Códigos auto-modificáveis, especialmente os empregados em linguagens de baixo nível, são usualmente difíceis de serem escritos, documentados e mantidos. [19]

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 auto-modificação 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 e formalização do crescimento de código seja desenvolvido para quantificar o tempo e o espaço do código produzido 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 neste 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.

9 CONCLUSÕES

Neste trabalho apresentamos os aspectos básicos para o projeto e a implementação de um ambiente responsável pela gerência de execução de códigos adaptativos. Este ambiente se torna necessário em função das características de auto-modificação presentes nos mesmos.

O 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 serão assegurados, tais como, o gerenciamento de escopo de variáveis, definição de rótulos de retomada de processamento e chamadas de funções adaptativas.

A nova forma de pensar introduzida pela tecnologia adaptativa induz um novo estilo de programação, uma vez que o comportamento dos programas adaptativos está diretamente associado ao código executável que o define, o qual se altera à medida que o processamento evolui.

Essa característica, típica dos dispositivos adaptativos, exige cuidados especiais da parte de quem desenvolve aplicações dessa natureza, envolvendo um estilo de raciocínio e uma disciplina de programação que antecipe os efeitos das ações adaptativas sobre o comportamento do dispositivo, e requer um esforço de pesquisa adicional para resolver os interessantes problemas dele decorrentes.

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. (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] 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] Santos, J.M.N. Um formalismo adaptativo com mecanismo de sincronização para aplicações concorrentes. Dissertação (Mestrado) – Escola Politécnica da USP - São Paulo, Brasil, 1997.
[5] Basseto, B. A., Neto, J.J. A stochastic musical composer based on adaptative algorithms. In: Anais do XIX Congresso Nacional da SBC-99. PUC-Rio, Rio de Janeiro, Brasil – 1999.
[6] Menezes, C.E.D. – Um método para a construção de Analizadores Morfológicos, Aplicados à Lingua Portuguesa, B aseado em Autômatos Adaptativos. Dissertação (Mestrado) – Escola Politécnica da Universidade de São Paulo, São Paulo – Brasil, Julho 2000.
[7] Iwai, M. K. – Um formalismo gramatical adaptativo para Linguagens dependentes de Contexto. Tese (Doutorado) – Escola Politécnica da Universidade de São Paulo, São Paulo, Brasil, 2000.
[8] Freitas, A. V.; Neto, J.J. Uma ferramenta para construção de aplicações multilinguagens de programação. In: CACIC 2001 – Congreso Argentino de Ciencias de la Computacion. - Argentina.
[9] Souza, M. A. A., Hirakawa, A. R., Neto, J. J. Adaptive Automata for Mobile Robotic Mapping. Proceedings of VIII Brazilian Symposium on Neural Networks - SBRN'04. São Luís/MA - Brazil. September 29 - October 1, 2004.
[10] Rocha, R. L. A.; Neto, J.J. Uma proposta de método adaptativo para a seleção automática de soluções. In: Proceedings of ICIE Y2K – International Congress on Informatics Engineering. Buenos Aires, Argentina. 2000.
[11] Neto, João José - Adaptive Rule-Driven Devices - General Formulation and Case Study**.** Lecture Notes in Computer Science. Watson, B.W. and Wood, D. (Eds.): Implementation and Application of Automata 6th International Conference, CIAA 2001, Vol.2494, Pretoria, South Africa, July 23-25, Springer-Verlag, 2001, pp. 234-250.
[12] 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) – Special Session Artificial Intelligence and Soft Computing.
[13] Barendregt, H.P. - The Lambda Calculus: its syntax and semantics – (2nd ed.), North-Holland, 1984.
[14] McCarthy, J. - Recursive Functions of Symbolic Expressions and Their Computation by Machine, Part-I. CACM 3,4 (1960), 184-195.
[15] MacLennan, Bruce J. – Functional Programming Practice and Theory – Addison-Wesley Publishing Company, Inc. 1990.
[16] Backus, John – Can Programming Be Liberated fromo the von Neumann Style? A Functional Style and Its Algebra of Programs - Communications of the ACM 21, 8 (August): 613-641.
[17] Burge, W. H. – Recursive Programming Techniques, Addison-Wesley, Reading, Mass. – 1975.
[18] Freitas, A. V. - Neto, João José - Conception of Adaptive Languages – International Conference on Modelling and Simulation - MS – 2006 – May 24 –26, 2006.
[19] Philip K. McKinley, Seyed Masoud, Sadjadi, Eric P. Kasten, Betty H. C. Cheng – Composing Adaptive Software - Michigan State University - IEEE Computer Society – 2004.
[20] Langsam Y., Augenstein M. J. , Tenenbaum A. M. – Data Structures using C and C++ - Second Edition – Prentice Hall – 1996.
[21] Anckaert B., Madou M. and Bosschere K.D. – A model for Self-Modifyng Code - Ghent University - Proceedings of the 8th Information Hiding Conference, 10-12 July 2006.
[22] Cifuentes C., Gough K.J. – Decompilation of Binary Programs –Software – Practice and Experience, Vol. 25(7), 811-829 – July – 1995.
[23] 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.
[24] 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.
[25] Homepage – Adaptive Technology - www.pcs.usp.br/~lta .
[26] 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.