Proposal of modification of the DJ Library for implementing Adaptive Technology

R. Caya, and J. J. Neto

***Abstract*—Implementation of adaptive programs have being an open concern in both scientific and commercial communities. The former one worrying about a clear and formal identification of the elements and behavior of adaptive mechanisms. The second one caring about the practical approach by which such behavior can be included at conventional applications. This paper presents a strategy for implementing Adaptive Technology in Java language using a standard library. Specifically, approaches about adaptive behavior in the Demeter Project and LTA are analyze to found correlated conceptual points. The objective is applying modifications over a tool, the DJ library, to achieve a best suit with the formalisms developed at LTA, without losing expressiveness at the implementation level. A particular case is analyzed by working with adaptive automata theory. From this case, key points are identified for future modifications through conversions in the representation of the underlying model and the traversal strategy used by the DJ library.**

***Keywords*— Adaptive Programming, Aspect-Oriented Programming, Adaptive Configuration, Adaptive Automata, Java.**

I. INTRODUÇÃO

ATUALMENTE, a adaptatividade é um dos recursos mais procurados dentro das aplicações de software. Independentemente do tamanho, a complexidade, ou do tipo de tecnologia utilizada, a capacidade de desenvolver peças de software que possuam a habilidade de reagir às mudanças dentro do contexto de execução, e sejam capazes de modificar o seu próprio comportamento para se adequar a novas situações é o principal requerimento das tecnologias de vanguarda.

Particularmente, espera-se que a adaptatividade permita que o software que a contém possa sobreviver as mudanças ao longo da sua vida útil por meio da aplicação de ajustamentos, de modo que os sistemas ganhem estabilidade e tolerância. Adicionalmente, o valor da adaptatividade se vê acrescentado ao ser entendida como uma base sobre a qual é possível desenvolver e aplicar teorias contemporâneas como *context-awareness*, *machine learning* [1], e *evolving computing*, as quais estão sendo foco de grande quantidade de pesquisas.

Ao longo dos anos, na tentativa para desenvolver peças de software com caraterísticas adaptativas têm seguido diferentes abordagens, tanto em paradigmas de programação quanto em metodologias, e até mesmo a terminologia utilizada para identificar um programa com comportamento adaptativo apresenta diversidade, aparecendo termos muito próximos como: auto-modificável e reconfigurável. No entanto, um contrato tácito, permite estabelecer uma definição geral de um programa adaptativo como aquele que é capaz de modificar o seu comportamento de acordo com as mudanças que acontecem dentro do seu ambiente de execução em determinado ponto (estado) de sua configuração [1]. Assim, o conceito de adaptatividade foi implementada dentro de programas que tem seguido diversos paradigmas de programação, de modo que existem programas adaptativos seguindo programação funcional [2], programação orientada a objeto [1], programação orientada a componentes [3], e mais recentemente programação orientada a aspectos [4].

Basicamente, esta diversificação é possível porque a adaptatividade é uma capacidade que pode ser abstraída e expressa por meio de teorias e métodos formais que expõem claramente os elementos e mecanismos que permitem o seu comportamento, dando-lhe um caráter autônomo. Não obstante, ao implementar esses dispositivos com tecnologia adaptativa é preciso levar a especificação teórica para o plano das linguagens de programação, as quais convencionalmente não fornecem suporte para adaptatividade [1]. É durante este processo que a adaptatividade fica embutida no meio do código convencional, camuflada, e perde tanto identidade quanto legibilidade.

Para superar esse problema é preciso ter um modo de implementação prático, que permita manter a clareza e expressividade que a adaptatividade tem no nível dos métodos formais, e a traduz para os vários paradigmas e linguagens de programação ao pôr em evidencia os elementos que a a compõem. Desta forma, é possível estabelecer um fluxo contínuo e facilmente reconhecível da correspondência dos elementos e métodos de um dispositivo adaptativo partindo da especificação formal até uma implementação concreta, tornando-se um elemento auto-descriptivo e diferençável dentro do código comum. Estas características podem ser aproveitadas não só no ambiente acadêmico, com finalidade didática, mas também para facilitar tarefas do desenvolvimento de software no ambiente comercial, tais como: documentação, manutenção, e até para expor as funcionalidades do sistema.

Neste trabalho se propõe o estabelecimento de uma correlação entre a especificação formal de um dispositivo adaptativo e a sua implementação concreta para uma linguagem de programação, mantendo o máximo de rastreabilidade entre os elementos que compõem o dispositivo em ambas as dimensões, e especialmente, facilitando a identificação da natureza particular do comportamento adaptativo par ao programador.

As próximas seções deste artigo estarão organizadas da seguinte maneira: na seção 2 serão abordados os conceitos correspondentes ao paradigma de Programação orientada a aspectos, sua filosofia e vantagens. Na seção 3 será apresentado o Projeto Demeter, que apresenta o conceito de Programação

Adaptativa e os mecanismos para desenvolver programas segundo aquele conceito. A seção 4, descreverá a biblioteca DJ que fornecerá a base para o trabalho deste projeto. A seção 5 fornecerão a correlação entre os conceitos de adaptatividade entre o Projeto Demeter e os trabalhos no LTA, e a seção 6 irá descrever a estratégia de mapeamento entro as descrições formais e as implementações dos dispositivos adaptativos. A seção 7 fornece uma revisão breve dos trabalhos relacionados para a tradução de notações formais e implementações nos casos de adaptatividade. Por fim, na seção 8 estarão as considerações finais.

II. PARADIGMA DE PROGRAMAÇÃO ORIENTADA A ASPECTOS

Programação Orientada a Aspectos (*Aspect-oriented programming* ou *AOP*) é um paradigma de metaprogramação que permite separar e organizar o código de um programa de acordo com a importância que ele tem dentro da aplicação (*separation of concerns*) [5]. Assim, contrário à Programação Orientada a objetos, na qual a é baseado na construção de entidades únicas que encapsulam semelhanças de estrutura e comportamento, AOP permite encapsular e modularizar aqueles métodos espalhados transversalmente na estrutura de classes destinados para atender funcionalidades de interesse global (*crosscutting concerns*). Esta nova abstração é possível de realizar graças à incorporação de um novo conceito denominado aspecto. De acordo com isso, diferentes módulos do programa são especificados durante o projeto de construção de software, cada um deles tomando conta de só um aspecto, e logo esses aspectos são entrelaçados no código principal para produzir um programa que aborde todos os requerimentos. Aqueles aspectos são entrelaçados por AOP por meio de "pontos de execução", ou *join points*, os quais são pontos dentro do programa de conformidade com as especificações do programador e que estão em capacidade de executar um comportamento adicional. Aqueles pontos podem ser entrada ou saída de métodos, criação ou eliminação de objetos [6].

As principais vantagens que a AOP traz consigo são que:

(i) um aspecto pode alterar o comportamento de um código estático (parte do programa não orientada a aspectos);
(ii) fornecem uma simplificação ao programa original;
(iii) fornecem facilidade para manutenção para o sistema pois tornasse tolerável para mudanças nas especificações por causa da evolução do software.

Apesar de que a ideia de entrelaçar aspectos é muito padronizada, o método pode variar substancialmente e se tornar muito dinâmico por meio do uso de reflexão, ou ficar completamente estático utilizando geração de código.

III. O PROJETO DEMETER

O Projeto Demeter [7] é um projeto de pesquisa desenvolvido na *Northeastern University*, cujo objetivo principal é melhorar a produtividade dos desenvolvedores de software em uma ordem de magnitude [8]. Para atingir isso são desenvolvidos métodos e ferramentas suportadas pelas teorias de Programação Adaptativa(*Adaptive Programming* ou AP) e Programação Orientada a Aspectos (*Aspect-Oriented Programming* ou AOP), as quais principalmente procuram manter um baixo acoplamento entre os objetos e as operações contidas em um programa, de modo que seja possível aplicar mudanças em qualquer aqueles aspectos sem causar impactos sérios dentro do outro [7]. As ferramentas desenvolvidas pelo equipe do projeto estão orientadas para incrementar as capacidades de tecnologias comerciais amplamente utilizadas, tais como as linguagens de programação C++, Java, C#, Tcl/Tk, Perl 5. No entanto não se limita a essas tecnologias e também trabalha em coordenação com UML e os padrões de projeto de software, de modo que os resultados podem ser más abrangentes [8]. Além disso, de acordo com o site oficial, eles estão trabalhando atualmente no desenvolvimento de ferramentas para XML e algumas linguagens de programação para criação de componentes de software.

As diferentes ferramentas disponibilizadas pelo Projeto Demeter encontrasse hierarquizadas na Fig. 1.

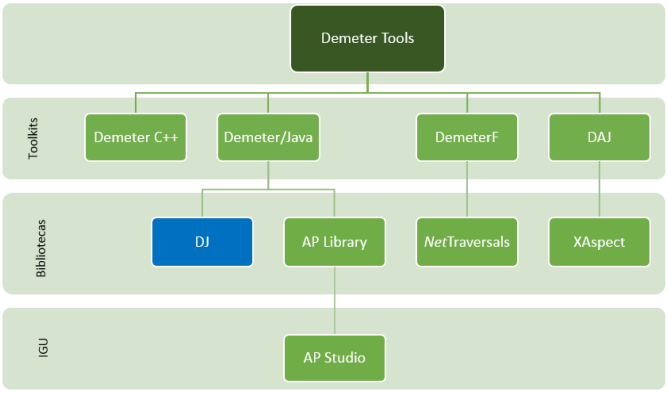

Figura 1. Ferramentas desenvolvidas pelo Projeto Demeter

A. Programação Adaptativa

De acordo com [9], a Programação Adaptativa é um caso particular do paradigma de AOP, no qual alguns dos blocos que compõem um programa são expressados em termos de grafos, e outros blocos podem consultar os primeiros por meio de conjuntos de estratégias de trajetórias. Assim, por meio de especificações breves é possível descrever o comportamento de programas nos níveis de projeto e implementação de software. A principal diferencia entre AOP e AP é que AOP tem como principal objetivo a separação de interesses que compõem o sistema, enquanto que AP ocupasse de aportar "timidez", entendido como desconhecimento parcial entre as implementações dos interesses e as estruturas onde são executados.

O objetivo principal da AP é separar as diferentes preocupações envolvidas na resolução de um problema minimizando as dependências entre as partes que o compõem, de modo que mudanças futuras em alguma das partes (sejam

arbitrarias ou produto da evolução do programa) tenha um impacto mínimo nas outras partes.

A vantagem que espera-se obter com esta abordagem é a construção de programas minimamente entrelaçados, mais flexíveis, compreensíveis, e mais curtos, sem que isso signifique perda de eficiência em tempo de execução. Em geral a AP procura elevar a Programação Orientada a Objeto para um nível maior de abstração permitindo ao usuário se focar nas classes essenciais do programa e nos aspectos invariantes das relações entre elas. Como consequência, AP utiliza protocolos que permitem especificar precondições, post condições, e invariantes para as diferentes classes participantes, e colaboradoras de uma operação lógica.

Assim, as características do AP realçam claramente a tolerância a mudanças e a incorporação de novo comportamento sobre uma base estática (invariável) como motivações importantes para o seu desenvolvimento. O importante para notar é que ambos conceitos representam pilares fundamentais dentro da teoria de Tecnologias Adaptativas especificadas e utilizadas no LTA. Esta proximidade é detalhada na seção V do presente documento.

B. Estratégias de travessia

A ideia central detrás das estratégias de travessia utilizadas em AP é implementar programas independentes da estrutura de classes especifica (*structure-shy*) de modo que a mesma estratégia pode ser utilizada com diferentes estruturas de classes [8]. Uma estratégia de travessia pode ser entendida como uma especificação parcial de um grafo, na qual são assinalados alguns nós invariantes (origem, destino, e alguns membros intermédios) e aristas estratégicas.

De modo geral, uma estratégia de travessia permite percorrer um grafo de objetos $\mathcal{O}$ (uma representação de um fluxo de dados não trivial) procurando todos os objetos de tipo c' dado um determinado objeto o de classe c, sem ter maior conhecimento sobre todas conexões da cadeia, só aquele fornecido pelo correspondente grafo de classes $\mathcal{C}$. Uma especificação formal e detalhada dos algoritmos que implementam a estratégia de travessia pode-se encontrar em [10] [11].

Assim, quando acontecem mudanças (chamadas como evoluções do programa) em partes não críticas a estratégia de travessia permanece invariante e válida, o que significa que a mesma codificação de comportamento é aplicável, e não é preciso realizar nenhuma alteração manual para adapta-a a aquela mudança, como máximo só é necessário atualizar a estrutura sobre a qual ira ser aplicada.

C. Padrão do Visitante Adaptativo

O *Visitor Pattern* é um padrão comportamental de projeto de construção de software desenvolvido em [12] para definir a comunicação entre as classes e os objetos que o compõem. A ideia principal é separar o comportamento de uma estrutura complexa de dados da funcionalidade que será executada sobre aqueles dados que ela possui. O benefício principal da implementação deste padrão é que pode ser incorporada nova funcionalidade para cada elemento da estrutura de dados sem conhecer antecipadamente qual é a relação com o resto do modelo. Essencialmente, o *Visitor Pattern* consiste em 2 partes: um método `visit()` implementado no visitante que é chamado por todos os elementos da estrutura de classes, e os métodos `accept()` em cada uma das classes que recebe um objeto visitante e acessa dinamicamente ao método `visit()` correspondente.

O padrão Visitante Adaptativo implementado nas ferramentas de Demeter é uma forma modificada do *Visitor Pattern* [13] a qual trabalha com ajuda das técnicas de reflexão para executar funcionalidades no tempo de execução. Uma das principais diferencias é que no Visitante Adaptativo só um conjunto mínimo de métodos deve ser definido, mais exatamente, só aqueles que correspondem com pontos desejados dentro da estratégia de travessia, e não todos como ocorre na versão original. Outra diferencia, é que no Visitante Adaptativo não é preciso implementar nenhum método `accept` e também não existe referência sobre o comportamento de travessia dentro dos métodos do visitante [9]. Além disso, o visitante adaptativo permite estabelecer comportamentos em quatro momentos da travessia:

- `start()`: invocado no início da travessia antes de começar o percurso do grafo.
- `before(ClassName c)`: é invocado quando um objeto `c` da classe `ClassName` é atingido. O método é executado antes de atravessar os atributos do objeto `c`.
- `after(ClassName c)`: é invocado quando um objeto c da classe `ClassName` é atingido. O método é executado depois de atravessar os atributos do objeto `c`.
- `finish()`: invocado no final da travessia, isto é, depois que todos os nós da estrutura foram percorridos

Assim, de acordo com [14] o padrão de implementação de um visitante permite construir uma unidade de comportamento independente das mudanças na estrutura de objetos e também da estratégia de travessia, de modo que pode ser reutilizado com outras configurações.

IV. BIBLIOTECA DJ

É uma biblioteca para Java, o nome vem do Demeter for Java, desenvolvida pelo equipe do Projeto Demeter com a finalidade de fornecer suporte, em código Java-nativo, para a implementação e experimentação das ideias da AOP [9]. A biblioteca DJ encontra-se disponível em [15], e em contraste com DemeterJ, para trabalhar com ela só e preciso realizar a incorporação padrão para uma biblioteca externa o qual simplifica o trabalho do programador.

Ao mesmo tempo, DJ procura cumprir a Lei, Demeter [16] [9], uma regra de estilo de programação para garantir o baixo acoplamento entre os aspectos estruturais e comportamentais de um programa, mas superando as desvantagens do espalhamento

de código por ela gerado quando ao seguir outros paradigmas de programação. A ideia principal em DJ é permitir a interpretação, em tempo de execução, de "métodos adaptativos" [13], entendidos como unidades de comportamento independentes do conhecimento do modelo de objetos do programa, mas que podem mudar a execução global do mesmo.

Aqueles métodos, junto com o comportamento que encapsulam, são expressados através dois mecanismos [13] [14]:

(1) Uma representação da estrutura de objetos do programa, chamada **grafo de classes**, a qual é gerada pela Classe *ClassGraph* de DJ utilizando reflexão em tempo de execução com ajuda do *ClassLoader* na JVM. A classe permite indicar o pacote de classes do qual deverá ser gerado o grafo, assim como também adicionar explicitamente outras classes ou pacotes ao grafo gerado, para os casos nos quais o código não se encontra disponível localmente ou as classes não foram carregadas ainda na JVM [9].

(2) Uma **estratégia de travessia**, a qual descreve como alcançar os participantes envolvidos na computação de um determinado comportamento em um alto nível, se referindo só para um número reduzido das classes no modelo de objetos do programa. Particularmente, só é preciso assinalar: uma raiz, uma classe alvo, e alguns pontos intermediários e restrições para garantir que a travessia ira seguir só aquelas trajetórias desejadas.

(3) Um **visitante adaptativo**, o agente que de fato vai percorrer o modelo de objetos de acordo com a estratégia, e ira aplicar um conjunto de ações (métodos *before* e *after*) quando um participante de determinado tipo seja alcançado.

Exemplo básico do uso de DJ em Java

```
// Importação das classes ...

public class Main {
  public static void main(String[] args) {

  // Criação de objetos e inicializações ...

    ClassGraph cg = new ClassGraph("transit");
    Strategy sg = new Strategy("from transit.BusRoute through transit.BusStop to
                                transit.Person");

    TraversalGraph tg = new TraversalGraph(sg, cg);
    tg.traverse(route, new Visitor() {
       public void start() { System.out.println("Scanning the state of a bus route");}
       public void finish() { System.out.println("end of the route scanning"); }

       public void before(BusStop b) { System.out.println("On "+ b.getName()); }
       public void after(BusStop b)  { System.out.println("going to next bus stop"); }

       public void after(Person p)  { System.out.println(p.getName()); }
    });

  // resto da implementação do método main
  }
}
```

Figura 2. Exemplo de programa utilizando a biblioteca DJ

A Fig.2 mostra um exemplo de um programa utilizando DJ para o caso de um sistema de trânsito com rotas e pontos de ônibus. O único requerimento é a impressão de todas as pessoas que aguardam em aquela rota. No exemplo, o objeto `cg` do tipo *ClassGraph* contém o grafo das classes conteúdas no pacote indicado como parâmetro [9], neste caso "`transit`", e o objeto `sg` implementa a estratégia de travessia de acordo com a cadeia de texto indicada como parâmetro. Neste caso é preciso mencionar o pacote no qual estão conteúdas as classes pois a referência para elas está sendo efeituada desde o método `main` que pertence ao pacote por defeito.

Com ambos objetos contendo a informação necessária é criado o grafo de travessia, `tg`, que contém os objetos instanciados no programa. O método `traverse` de `tg` expõe o mecanismo de percurso dentro de `tg`, iniciando no nó indicado como raiz (`route`), um objeto do tipo Visitor vai ativar os métodos adaptativos correspondentes. No exemplo, são implementados métodos `before` e `after` para o caso em que o visitante chegar para uma instancia da classe `BusStop`, e um método `after` para o caso em que uma instancia da classe `Person` seja encontrada dentro do percurso do grafo.

A codificação integral do exemplo pode ser obtida e testada de livremente desde https://github.com/rosedth/WTA2014.

V. RELACIONAMENTO ENTRE ABORDAGENS: PROJETO DEMETER E LTA

Apesar de ter sido desenvolvidos em lugares distantes, com objetivos e motivações diferentes, existem acordos do nível conceitual entre a proposta de Programação Adaptativa feita pelo Projeto Demeter e os trabalhos com dispositivos adaptativos realizados no Laboratório de Tecnologias Adaptativas (LTA) da Escola Politécnica da Universidade de São Paulo. Nesta seção irão ser apresentadas ambas abordagens de adaptatividade, primeiro de maneira individual, e em seguida, irão se expor as suas semelhanças.

A. Adaptatividade segundo Demeter

O Projeto Demeter incorpora o conceito de adaptatividade desde dois abordagens diferentes: o abordagem de transcendência dos programas às mudanças estruturais, e o abordagem de incorporação de comportamentos em tempo de execução. A continuação iremos detalhar ambos os abordagens.

Primeiro, segundo [17] para o Projeto Demeter a adaptatividade em um programa é entendida como a capacidade de aquele programa para se adaptar automaticamente as mudanças que acontecem dentro do seu contexto, particularmente mudanças estruturais, de maneira que o impacto gerado por elas seja o mínimo possível dentro do sistema integral. Para atingir esse objetivo um programa adaptativo é expressado por meio de fragmentos de código que encapsulam os conceitos e as funcionalidades requeridas para desenvolver uma determinada funcionalidade, mas com conhecimento mínimo sobre a estrutura na qual iram ser aplicadas. Assim esses fragmentos apresentam baixo acoplamento entre eles mas cooperam entre si para formar o

sistema integral [14]. De acordo com isso, é possível aplicar o mesmo comportamento para diferentes estruturas de classes, sempre e quando se cumpra com manter invariáveis um conjunto de especificações sucintas.

Segundo, o Projeto Demeter comparte com AOP as características atribuídas para o conceito de aspecto, mais especificamente para Demeter tanto a estratégia de travessia como o visitante adaptativo corresponde a aspectos em AOP. Devido a isso também admitem, por definição, a capacidade que um aspecto tem para alterar o comportamento de um programa por meio da aplicação de um comportamento adicional (*advice* em AOP, e para Demeter os métodos implementados por o Visitante Adaptativo) em um determinado ponto de execução (*join point* em AOP, e para Demeter um objeto dentro da travessia para o qual existe implementado um comportamento adicional).

B. Tecnologias Adaptativas no LTA

O Laboratório de Tecnologias Adaptativas (LTA) da Escola Politécnica da Universidade de São Paulo tem realizado pesquisas relacionadas com tecnologias adaptativas desde os anos 90. Como resultado diversos formalismos, técnicas, métodos, ferramentas e aplicações foram desenvolvidas com base na incorporação da adaptatividade em diversas áreas.

Dentro do contexto dos trabalhos desenvolvidos no LTA, a adaptatividade é entendida como a capacidade que tem um sistema de responder para estímulos externos determinando e realizando auto reconfigurações dinâmicas de tipo estrutural [18] e comportamental [19], levando em conta para efeituar aquelas mudanças a informação acumulada durante experiências anteriores [20]. Assim, o termo Tecnologia Adaptativa refere-se a aplicação da adaptatividade e fins práticos e concretos.

Um dispositivo adaptativo é entende-se como um dispositivo cuja operação é guiada por regras e que é capaz por seus próprios médios de realizar mudanças nesse conjunto regras. Na construção formal de aqueles dispositivos é possível identificar duas componentes participantes [19]: um **dispositivo subjacente** (usualmente não adaptativo), e um **mecanismo adaptativo**, responsável pela incorporação da adaptatividade. Desse modo, no momento em que uma dependência de contexto é detectada pelo dispositivo, uma ação adaptativa de automodificação é efeituada, a qual a sua vez pode mudar o conjunto de regras definidas para a execução do dispositivo e o estado corrente em que se encontra [21].

C. Relacionamento entre ambos abordagens

De acordo com o exposto, os métodos es técnicas desenvolvidos dentro do Projeto Demeter para suportar a Programação Adaptativa fornecem uma plataforma que é possível de aproveitar para a implementação de tecnologia adaptativa. Particularmente, ambas as abordagens:

- Separam conceitualmente os mecanismos adaptativos a serem aplicados da estrutura que que representa e controla o funcionamento principal do dispositivo.
- Identificam o recorrido de uma a representação abstrata do funcionamento do programa como dispositivo subjacente sobre o qual iram acontecer as mudanças adaptativas.
- Identificam as duas oportunidades de aplicação de mecanismos adaptativos: uma imediatamente antes e outra imediatamente depois de "atravessar" um elemento da estrutura subjacente.
- Admitem que a execução das ações adaptativas pode alterar o comportamento geral do dispositivo.

Essas aproximações conceituais permite identificar as ferramentas desenvolvidas no Projeto Demeter como suporte para a implementação de dispositivos adaptativos partindo da especificação formal e expondo claramente os elementos que os compõem. Assim, a expressividade outorgada na especificação formal de aqueles dispositivos pode ser identificada dentro da sua codificação, tornando-a descritiva.

VI. ESTRATÉGIA DE MAPEAMENTO ENTRE ABORDAGENS

Esta seção apresenta um exemplo que permite identificar mais claramente a compatibilidade entre os abordagens seguidos no Projeto Demeter e os Dispositivos Adaptativos do LTA. Neste caso se trabalha com a teoria de Autómatos Adaptativos (AA) descrita em [22] por apresentar uma versão simplificada e expressiva.

Um AA está definido por um dispositivo subjacente que corresponde com um autômato de pilha estruturado, e um mecanismo adaptativo. O autômato de pilha estruturado é identificado por a 7-tupla: $M = (Q, \Sigma, \Gamma, \delta, q_0, \mathrm{Z}, F)$ onde:

- Q: é um conjunto finito de estados
- Σ: é o alfabeto de entrada
- Γ: é o alfabeto da pilha
- δ: são o conjunto de regras de transição: internas, externas e de chamada e retorno de sub-máquina.
- q_0: é o estado inicial do autômato
- Z: é o símbolo inicial na pilha
- F: é o conjunto de estados finais do autômato

Entanto o mecanismo adaptativo é expressado como a aderência de um conjunto opcional de (zero, uma ou até duas) ações adaptativas correspondentes com a regra de transição a ser aplicada. Existem dois momentos nos quais aquelas ações podem ser executadas: antes ou depois da aplicação da regra selecionada.

Uma ação adaptativa descreve as modificações a serem aplicadas no autômato adaptativo no momento em que forem chamadas. As modificações geradas por uma ação adaptativa podem ser de três tipos de notadas da seguinte maneira:

- $?$: ação de tipo inspeção, a qual procura as regras de transição que cumprem um determinado padrão.
- $-$: ação de tipo remoção, a qual remove uma regra do conjunto de transições se ela cumpre com determinado padrão.
- $+$: ação de tipo inserção, a qual incorpora uma nova regra no conjunto de transições de acordo com determinado padrão.

Assim, em cada estado da execução do autômato são executadas tanto as regras de transição convencionais como as ações adaptativas, até atingir um estado final.

Partindo dessa formalização é possível descrever o relacionamento que permite aproveitar as técnicas do Projeto Demeter para a implementação de um autômato adaptativo.

Uma primeira proposta para representar o dispositivo subjacente é aproveitar o paradigma de programação orientada a objetos para representar os elementos do autômato. Assim seria possível implementar uma hierarquia de classes como a que se mostra na Fig.3.

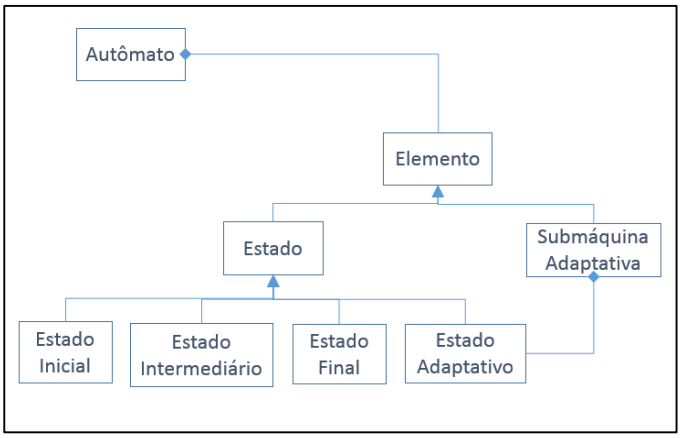

Figura 3. Proposta da hierarquia de classes para a representaçaõ de um AA

De modo que Q está representado pela lista de elementos que formam parte do autômato, e Σ corresponde à entrada, entendida como fluxo de dados, com a qual irá trabalhar o autômato. O estados identificados por q_0 e F estão representados por meio de herança na hierarquia de objetos apresentada, e formam parte da lista de elementos do autômato.

Os elementos que se referem ao trabalho com pilha, Γ e Z, vão se encontrar encapsulados dentro do mecanismo de travessia de DJ, o qual indica o caminho a percorrer. Assim, também é possível efeituar as chamadas a sub-máquina de maneira evidente e natural graças à possibilidade de inclusão de uma sub-máquina como um elemento dentro da lista de elementos do autômato.

O conjunto de transições está representado pela estratégia de travessia, o *ClassGraph* e o *ObjectGraph*. Com esses elementos DJ estabelece as regras de transição como a possibilidade de acesso entre um objeto e outro segundo a hierarquia de objetos embebida no *ClassGraph*. Atualmente, DJ só permite uma trajetória dirigida, na qual percorre cada nó do grafo uma única vez. Dentro das modificações desejadas estaria a implementação de trajetórias que permitam atravessar novamente por alguns estados.

O mecanismo adaptativo é expressado por meio da implementação do padrão do Visitante Adaptativo em DJ, no qual as ações adaptativas podem ser adicionadas para serem executadas nos momentos em que determinados estados forem atingidos. Os momentos nos quais é possível incrementar um comportamento que potencialmente realizara uma mudança no comportamento do programa estão expressos dentro de DJ como os métodos adaptativos `before` e `after` correspondentes a um objeto determinado. Dentro dos métodos adaptativos é possível realizar inspeção (examinar os valores de alguma variável ou atributo dos objetos), remoção e inserção de elementos.

Neste ponto, é importante identificar que existe inclusive um consenso de terminologia entre ambos os abordagens respeito ao nome dos mecanismos que permitem a adaptatividade.

VII. MODIFICAÇÕES PROPOSTAS

A maior parte das modificações a serem realizadas na biblioteca DJ correspondem à representação do dispositivo subjacente e à implementação da estratégia de travessia, assim como possíveis efeitos colaterais das mudanças nela aplicadas.

Inicialmente, e preciso modificar o modelo de representação de uma árvore de classes para um modelo de grafo que permita refletir mais livremente o comportamento do dispositivo.

Também é necessário fornecer regras de transição mais enriquecidas dentro da estratégia de travessia que permitam realizar a execução do autômato de maneira mais natural que aquela equivalente ao recorrido de grafos dirigidos. Atualmente, a estratégia de travessia só estabelece como regras de transição se um elemento é alcançável ou não a partir de outro com base na hierarquia de classes (ClassGraph).

Particularmente, é necessário incorporar dentro da estratégia de travessia a possibilidade de retornar para determinados estados dentro da lista de elementos do autômato para os casos nos quais existem laços entre estados.

As modificações dentro da estratégia de travessia deveriam ser transparentes para o programador já que DJ fornece a possibilidade de inserir a especificação de uma estratégia como uma cadeia de caracteres ou a partir de um arquivo de texto.

VIII. TRABALHOS RELACIONADOS

Com o surgimento das teorias sobre comportamento adaptativo, surgiu também a necessidade de elementos computacionais que permitam fazer uso da potencialidade por ela fornecida. Particularmente, a existência de uma linguagem de programação que suporte com facilidade a implementação de mecanismos adaptativos por meio da inclusão de primitivas adequadas é identificada como uma grande necessidade.

Tanto [2] quanto [1] são exemplos de esforços realizados pela comunidade cientifica para fornecer suporte para a incorporação de adaptatividade dentro dos programas convencionais. No entanto, ambos as abordagens propõem a criação de novas linguagens de programação que implementam as primitivas necessárias para adaptatividade.

Uma outra necessidade é aquela de fornecer uma transição clara entre a fundamentação teórica de um dispositivo e a implementação do mesmo. O objetivo neste caso é do nível didático e para facilitar o entendimento do comportamento de

aquele dispositivo. A biblioteca JFLAP [23](Formal Languages and Automata Theory Package for Java) procura realizer dita aproximação provendo uma ferramenta para visualização e teste de especificações formais.

Neste trabalho utiliza-se as técnicas desenvoltas dentro de Projeto Demeter, particularmente na biblioteca DJ, que são métodos que permitem o desenvolvimento de programação adaptativa e servem para resolver diversos problemas computacionais em diversos linguajes de programação e paradigmas. O intuito é adaptá-las para permitir uma representação mais natural dos elementos conteúdos nas especificações formais dos dispositivos adaptativos. Desse modo, a principal diferencia com os trabalhos anteriores é que não se propõe a modificação da linguagem ou a criação de novas linguagens para a implementação de adaptatividade, si não que se procura aproveitar as facilidades que prove DJ, como biblioteca nativa, para apresentar aos programadores uma alternativa que de forma padrão permita-lhes experimentar programas com características adaptativas.

IX. CONCLUSÃO

Diante do exposto, conclui-se que, é possível realizar um mapeamento entre as especificações formais dos dispositivos adaptativos para uma implementação assistida pelos conceitos desenvolvidos dentro do Projeto Demeter. Implementação que seja capaz de conservar e refletir a expressividade, identidade e particularidade dos elementos que definem o comportamento adaptativo. Embora, algumas modificações precisam ser feitas para atingir uma compatibilidade total, a correlação encontrada entre os abordagens seguidos tanto por o equipe do Projeto Demeter como pôr o equipe do LTA, permitem estabelecer uma plataforma sobre a qual trabalhar com miras a atingir o objetivo de facilitar a inclusão de tecnologias adaptativas pelos programadores nos sistemas.

A viabilidade do projeto de pesquisa é suportada devido a que o Projeto Demeter é um projeto de software livre que até hoje conta com uma comunidade ativa que trabalha continuamente sobre os objetivos para ele estabelecidos. Do mesmo modo, as formulações formais dos dispositivos de Tecnologia Adaptativa permitem pôr à disposição do programador uma ferramenta poderosa a qual faria possível uma mudança na metodologia de programação: programação baseada em formulações teóricas, uma conexão entre teoria e prática desejada por longo tempo.

O potencial de lograr uma implementação com clareza dos mecanismos adaptativos aporta não só como finalidade didática si não que também permite aproveitar os benefícios de um uso ciente das vantagens que eles fornecem. Ao respeito, trabalhar sobre a base de uma das ferramentas desenvolvidas pelo Projeto Demeter permite pensar na possibilidade de propagar o trabalho para outras ferramentas que suportam diferentes linguagens e paradigmas de programação.

REFERÊNCIAS

[1] C. Simpkins, S. Bhat, C. J. Isbell e M. Mateas, "Towards Adaptive Programming: Integration Reinforcement Learning into a Programming Language," em *23rd ACM SIGPLAN conference on Object-oriented programming systems languages and applications*, New York, 2008.

[2] U. A. Acar, G. E. Blelloch e R. Harper, "Adaptive Functional Programming," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 28, nº 6, pp. 990-1034, 2006.

[3] H. Cervantes e R. S. Hall, "A Framework for Constructing Adaptive Component-Based Applications: Concepts and Experience," em *Lecture Notes in Computer Science. Component-Based Software Engineering*, vol. 3054, I. Crnkovic, J. A. Stafford, H. W. Schmidt e K. Wallnau, Eds., Springer Berlin Heidelberg, 2004, pp. 130-137.

[4] W. Vanderperren, D. Suvée, B. Verheecke, M. A. Cibrán e V. Jonckers, "Adaptive programming in JAsCo," em *4th international conference on Aspect-oriented software development*, New York, USA, 2005.

[5] B. Donkervoet e G. Agha, "Reflecting on aspect-oriented programming, metaprogramming, and adaptive distributed monitoring," em *5th international conference on Formal methods for components and objects*, Amsterdam, The Netherlands, 2007.

[6] P. Cointe, H. Albin-Amiot e S. Denier, "From (Meta) Objects to Aspects: A Java and AspectJ Point of View," em *Formal Methods for Components and Objects. Lecture Notes in Computer Science*, vol. 3657, F. S. Boer, M. M. Bonsangue, S. Graf e W. Roever, Eds., Berlin, Springer Berlin Heidelberg, 2005, pp. 70-94.

[7] Northeastern University, "Demeter: Aspect-Oriented Software Development," College of Computer and Information Science, 2013. [Online]. Available: http://www.ccs.neu.edu/home/lieber/what-is-demeter.html. [Acesso em 28 Outubro 2013].

[8] K. Lieberherr, "Demeter and aspect-oriented programming," em *STJA Conference*, Erfurt, Germany, 1997.

[9] K. Lieberherr e D. H. Lorenz, "Coupling Aspect-Oriented and Adaptive Programming," em *Aspect-Oriented Software Development*, R. E. Filman, T. Elrad, S. Clarke e M. Akşit, Eds., Boston, Addison-Wesley, 2005, pp. 145-164.

[10] K. J. Lieberherr, B. Patt-Shamir e D. Orleans, "Traversals of object structures: Specification and Efficient Implementation," *ACM Trans. Program. Lang. Syst.*, vol. 26, nº 2, pp. 370-412, 2004.

[11] K. Lieberherr e M. Wand, "A Simple Semantics for Traversals in Object Graphs," Boston, Massachusetts, USA, 2001.

[12] E. Gamma, R. Helm, R. Johnson e J. Vlissides, Design P atterns: Elements of Reusable Object-Oriented Software, Addison-Wesley, 1995.

[13] K. Lieberherr, D. Orleans e J. Ovlinger, "Aspect-oriented programming with adaptive methods," *Commun. ACM*, vol. 44, nº 10, pp. 39-41, Outubro 2001.

[14] D. Orleans e K. Lieberherr, "DJ: Dynamic Adaptive Programming in Java," em *Metalevel Architectures and Separation of Crosscutting Concerns. Lecture Notes in Computer Science*, vol. 2192, A. Yonezawa e S. Matsuoka, Eds., Berlin, Springer Berlin Heidelberg, 2001, pp. 73-80.

[15] Demeter Research Group, "DJ: Dynamic Structure-Shy Traversals and Visitors in Pure Java," College of Computer Science, Northeastern University, [Online]. Available: http://www.ccs.neu.edu/research/demeter/DJ/.

[16] K. J. Lieberherr e I. Holland, "Assuring Good Style for Object-Oriented Programs," *IEEE Software*, pp. 38-48, Setembro 1989.

[17] K. J. Lieberherr, Adaptive Object-Oriented Software: The Demeter Method with Propagation Patterns, Boston: PWS Publishing Company, 1996.

[18] M. A. A. Sousa, A. R. Hirakawa e J. J. Neto, "Adaptive Automata for Mapping Unknown Environments by Mobile Robots," *Lecture Notes in Artificial Intelligence. Advances in Artificial Intelligence - IBERAMIA 2004*, pp. 562-571.

[19] J. J. Neto, "Um Levantamento da Evolução da Adaptatividade e da Tecnologia Adaptativa," em *Revista IEEE América Latina*, 7 ed., vol. 5, IEEE, 2007, pp. 496-505.

[20] A. H. Tchemra, "Aplicação da Tecnologia Adaptativa em Sistemas de Tomada de Decisão," *Revista IEEE América Latina*, vol. 5, nº 7, pp. 552-556, Novembro 2007.

[21] J. J. Neto e A. V. Freitas, "Using Adaptive Automata in a Multi-Paradigm Programming Environment," em *International Conference on Applied Simulation and Modelling, ASM2001 - IASTED*, Marbella, Espanha, 2001.

[22] J. J. Neto e C. Bravo, "Adaptive Automata - a reduced complexity proposal," em *7th International Conference of Implementation and Application of Automata (CIAA 2002)*, Tours, France, 2002.

[23] S. H. Rodger, E. Wiebe, K. M. Lee, C. Morgan, K. Omar e J. Su, "Increasing engagement in automata theory with JFLAP," em *40th ACM technical symposium on Computer science education, SIGCSE '09*, Chattanooga, TN, USA, 2009.

Rosalia Edith Caya Carhuanina nasceu na cidade de Lima, Perú, em 27 de fevereiro de 1986. Recebeu o título de Bachiller em Ciencias e Ingeniería con Mención em Ingeniería Informática da Pontificia Universidad Católica del Perú (PUCP) em Lima, Perú. Entre os anos de 2007 e 2011, participou do Grupo de Pesquisa em Inteligência Artificial do departamento de Engenharia Informática desta Universidade. Atualmente é aluna de Mestrado em Engenharia Elétrica, na área de Engenharia de Computação, da Universidade São Paulo (USP), desenvolvendo sua pesquisa no Laboratório de Tecnologias Adaptativas. Suas principais áreas de pesquisas são: Inteligência Artificial, Processamento de Linguagem Natural e Tecnologia Adaptativa.

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