Sistemas de Markov Adaptativos: Formulação e Plataforma de Desenvolvimento

D.A. Alfenas, D.P. Shibata, J.J. Neto, M. R. Pereira-Barretto

***Abstract*— This paper extends Adaptative Markov Machines formalism and presents a framework allowing to design, implement, test and embed Markov adaptive systems into Java applications.**

***Keywords*— Markov machines framework, adaptivity, adaptive Markov machines.**

I. INTRODUÇÃO

O desenvolvimento de aplicações baseadas em tecnologias adaptativas ainda é incipiente. Uma das razões é que as ferramentas de suporte ao desenvolvimento destes softwares não alcançaram plena maturidade, mesmo que diversas pesquisas recentes tenham demonstrado como as técnicas adaptativas podem ser aplicadas nas mais diversas áreas. Foi exibida a aplicação no processamento digital de imagens e aprendizado de máquina [1], compiladores [1] [2], aprendizado à distância [1], tradução texto-fala [3], segurança da informação [4] e geração de música [5], entre muitas outras áreas.

Especialmente relevante para este artigo é o trabalho apresentado em [5]. Nele foi desenvolvido o LASSUS, um sistema capaz de gerar e executar música em tempo real. O autor utilizou geradores de sequências pseudo-aleatórias para abastecer um núcleo de execução baseado em máquinas de Markov adaptativas. A aleatoriedade faz com que cada execução do LASSUS produza uma composição diferente e imprevisível. Embora as qualidades musicais nos trechos compostos pelo sistema sejam reconhecidas por qualquer um que escute as saídas produzidas pelo sistema, não houve continuidade do trabalho nos dez anos seguintes.

As máquinas de Markov adaptativas utilizadas na geração de música podem ser generalizadas e aplicadas em outras áreas, pois viabilizam simular ações criativas através de decisões aleatórias, isto é, sistemas cuja evolução não depende, obrigatoriamente, da interação entre o usuário do sistema e a máquina.

Este artigo apresenta o *MMAdapt (Máquinas de Markov Adaptativas)*, uma plataforma de desenvolvimento de sistemas baseados em máquinas de Markov adaptativas. Descreve-se a estrutura interna e a interface de programação (*API*), assim como sugere possíveis aplicações. Descreve também a formulação matemática necessária para mostrar como é a teoria que originou toda a plataforma.

D. A. Alfenas, Fundação para o Desenvolvimento da Engenharia (FDTE), São Paulo, Brasil, d.alfenas@fdte.org.br.

D. P. Shibata, Fundação para o Desenvolvimento da Engenharia (FDTE), São Paulo, Brasil, d.shibata@fdte.org.br

J. J. Neto, Escola Politécnica da USP (EPUSP), São Paulo, Brasil, joao.jose@poli.usp.br

M. R. Pereira-Barretto, Escola Politécnica da USP (EPUSP), São Paulo, Brasil, marcos.barretto@poli.usp.br.

II. MÁQUINAS DE MARKOV ADAPTATIVAS

A formalização das máquinas de Markov adaptativas utilizada neste trabalho se baseia, ao mesmo tempo em que a estende, naquela introduzida originalmente em [5], como parte do desenvolvimento do LASSUS. Neste item, faz-se uma introdução sucinta acerca de adaptatividade e cadeias de Markov para, em seguida, apresentar o formalismo em que a construção da plataforma é baseada.

A. Adaptatividade

Uma das idéias centrais no formalismo adaptativo é que dispositivos mais poderosos, em relação à capacidade de expressão e de computação, podem ser obtidos gradualmente a partir dos recursos oferecidos por um dispositivo mais simples.

Pode-se generalizar o conceito, aqui aplicado nas redes de Markov, para qualquer dispositivo guiado por regras. As regras mapeiam cada configuração do dispositivo em uma nova configuração, eventualmente estimulado por alguma entrada ou gerando alguma saída. Ele inicia em uma determinada configuração e segue aplicando regras até que não haja mais estímulos de entrada ou se atinja uma configuração à qual nenhuma regra possa ser aplicada.

Em um dispositivo não-adaptativo, o conjunto de regras é o mesmo durante toda a sua execução. Um dispositivo adaptativo adiciona um conjunto especial de regras, chamadas de ações adaptativas, que agem modificando o conjunto original através de adições, alterações e remoções. A camada composta desse conjunto de ações adaptativas é chamada de *camada adaptativa*; aquela composta das regras originais tem o nome de *camada subjacente*. Além das redes de Markov, a adaptatividade já foi incorporada a outros dispositivos como: autômatos [6], gramáticas [9], árvores e tabelas de decisão [8] e statecharts [7], entre outros. Uma visão ampla acerca da adaptatividade encontra-se em [6], [1].

B. Cadeias de Markov

Define-se uma cadeia de Markov como uma tripla

$$K = (Q, q_0, T) \quad (1)$$

em que Q é um conjunto finito com n estados, q_0 é o estado inicial da rede e T é um conjunto de transições entre estados. Cada transação γ_{ij} é caracterizada como uma tripla

$$\gamma_{ij} = (q_i, q_j, \rho_{ij}) \quad (2)$$

em que ρ é a probabilidade do estado q_j ser atingido estando em q_i. Observe que cada transição γ_{ij} é única em T, isto é, não há duas transições diferentes partindo de um mesmo estado q_i e chegando a q_j. Além disso, seja Γ_q como o conjunto de todas as transições que se originam em um estado q. Então

$$\forall q \in Q, \sum_{\gamma \in \Gamma_q} \rho_\gamma = 1 \quad (3)$$

Isto é, a soma das probabilidades de todas as transições iniciando em q é exatamente um, para todo q pertencente a Q.

A definição dada por (1), (2) e (3) garante que a probabilidade de um dado estado ser atingido depende exclusivamente do estado atual da rede, isto é, a cadeia de Markov aqui definida é de primeira ordem. Pode-se provar também que, partindo-se do estado inicial q_0 e após um número suficientemente grande de iterações da rede, a probabilidade da rede encontrar-se em um de seus n estados é constante [10].

Cabe ressaltar uma diferença em relação à representação utilizada na definição original [10]: enquanto aqui T é um conjunto de transições, como na definição de autômatos, o texto original o apresenta como uma matriz n x n, com as mesmas propriedades. Desta forma, uma transição com ρ igual a zero é interpretada diferente de uma transição inexistente, o que será importante para a definição das máquinas de Markov adaptativas.

C. Máquinas de Markov

Uma máquina de Markov $\mathcal{M}$ é definida pela quádrupla

$$\mathcal{M} = (Q, S, T, q_0) \quad (4)$$

em que Q, T e q_0 são como definidos anteriormente para cadeias e S é o alfabeto de saída. A equação (2) é substituída por

$$\gamma_{ij} = (q_i, q_j, \rho_{ij}, p_{ij}) \quad (5)$$

como definição de transição. Quando γ_{ij} é acionada, o símbolo $p_{ij} \in S \cup \{\varepsilon\}$ é inserido na cadeia de saída de $\mathcal{M}$.

Desta forma, é possível utilizar $\mathcal{M}$ como um dispositivo gerador de linguagem; a linguagem formada pelas possíveis cadeias de saída de $\mathcal{M}$ é linear. É possível mostrar, também, que o método de produção de linguagens por meio da máquina de Markov é formalmente equivalente ao das gramáticas lineares [5].

D. Máquina de Markov Adaptativa

Uma máquina de Markov adaptativa M é definida por uma quíntupla

$$M = (Q, S, T, q_0, F) \quad (6)$$

em que Q, S, T, e q_0 são como definidos anteriormente para $\mathcal{M}$ e F é um conjunto de funções adaptativas. Toda função adaptativa pertencente a F pode ser definida como uma quádrupla

$$f = (\Psi, V, G, C) \quad (7)$$

em que:

- Ψ é um conjunto de parâmetros formais (ψ_1, ψ_2, ψ_3,..., ψ_m);
- V é um conjunto de identificadores de variáveis (v_1, v_2, ..., v_n), cujos valores são desconhecidos no instante de chamada de f mas que uma vez preenchidos terão seus valores preservados durante toda a execução da função;
- G é um conjunto de identificadores de geradores (g^*_1, g^*_2,..., g^*_n), variáveis especiais que são preenchidas com novos valores, ainda não utilizados pelo autômato, a cada vez que a função é chamada;
- C é uma sequência de ações adaptativas elementares executadas em f. Ressalta-se que a imposição de uma ordem nas ações não faz parte da definição original do formalismo adaptativo [6].

A equação (5) é substituída por sua forma final

$$\gamma_{ij} = (q_i, q_{j,} \rho_{ij}, p_{ij}, a_{ij}). \quad (8)$$

como definição de transição, em que q_i, $q_{j,}$ ρ_{ij} e p_{ij} são como definidos anteriormente e a_{ij} é uma ação adaptativa da forma

$$a_{ij} = f(\omega_1, \omega_2, ..., \omega_n) \cup \{\varepsilon\} \quad (9)$$

sendo f uma função adaptativa pertencente a F e (ω_1, ω_2, ..., ω_n) uma lista de argumentos que correspondem posicionalmente à lista de parâmetros Ψ declaradas para f. $\{\varepsilon\}$ representa o conjunto vazio, isto é, é possível que uma transição não tenha ação adaptativa associada.

Definidos desta forma, o procedimento de mudança de estado de uma máquina de Markov adaptativa M é:

1. Obtém-se a lista de transições possíveis Γ_{qi} a partir do estado ativo q_i em M;
2. Escolhe-se uma transição γ_{ij} a disparar utilizando-se as probabilidades ρ_{ik} das transições $\gamma_{ij} \in \Gamma_{qi}$;
3. A máquina muda para o estado q_j;
4. O símbolo p_{ij} é inserido na cadeia de saída de M;
5. Se $a_{ij} \neq \varepsilon$, então a função adaptativa f é chamada com os argumentos (ω_1, ω_2, ..., ω_n), realizando a ação adaptativa sobre M.

1) Ações adaptativas elementares

As ações adaptativas elementares utilizadas em C podem ser dos seguintes tipos:

- *Consulta de transição*: busca transições conforme um filtro recebido por parâmetro e as retorna;
- *Inserção de transição*: cria uma nova transição entre dois estados existentes. Se ela já existir, a substitui;
- *Remoção de transições*: remove uma ou mais transições recebidas por parâmetro.

Além destas, foram introduzidas três novas ações adaptativas elementares para aumentar o poder de expressão, mas que, entretanto, não aumentam o poder de computação:

- *Alteração de transições*: altera ρ, p_{ij} ou a_{ij} em uma ou mais transições recebidas como parâmetro. É equivalente a uma ação de remoção de transição entre os estados q_i e q_j seguida por outra de inserção de transição entre os mesmos estados q_i e q_j;
- *Chamada de ação adaptativa não-elementar*: recebe como parâmetro uma função adaptativa f e uma lista de argumentos (ω_1, ω_2, ..., ω_n) que correspondem posicionalmente à lista de parâmetros Ψ declaradas para

f. Introduzida aqui para melhorar a reutilização das funções adaptativas;

- *Compara duas variáveis*: esta ação recebe quatro parâmetros (ω_1, ω_2, $a_=$, $a_\neq$). Se ω_1 e ω_2 forem iguais, executa-se $a_=$; caso contrário, executa-se $a_\neq$.

Para demonstrar que não há aumento do poder computacional com a definição de ação elementar de comparação de duas variáveis, seja uma função adaptativa *x* que recebe os mesmos quatro argumentos (ω_1, ω_2, $a_=$, $a_\neq$) mas que é construída somente com ações elementares dos cinco tipos anteriores e é capaz de:

- executar $a_=$ apenas se ω_1 e ω_2 são iguais;
- executar $a_\neq$ apenas se ω_1 e ω_2 são diferentes

Para isso, constrói-se a máquina de Markov de forma que, entre os vários estados $q \in Q$, existam dois estados q_1 e q_2, chamados de *marcadores*. Eles precisam ser inatingíveis, isto é, não existe transição da forma γ_{ik} em *T*, $k \in \{1,2\}, i \notin \{1,2\}$. Além disso, q_1 não possui nenhuma transição partindo de si próprio e q_2 possui transições para todos $q \in Q$. Então, constrói-se *x* contendo as seguintes ações:

1. Insere-se uma transição γ_{1i};
2. Executa-se uma ação elementar de consulta de transição começando em q_1 e terminando em *v*. Se *v* e q_i forem iguais, retorna $r_1 = \gamma_{1i}$, caso contrário, retorna $r_1 = \varepsilon$;
3. Executa-se uma ação adaptativa $b_=$ com os mesmos argumentos passados para *x* e mais um argumento contendo r_1. A única ação dentro da função de $b_=$ é chamar $a_=$. Se r_1 for ε, a ação adaptativa com $b_=$ não é executada e a primeira condição (executar $a_=$ apenas se iguais) é satisfeita;
4. Remove-se a transição γ_{2i};
5. Executa-se uma ação elementar de consulta de transição começando em q_2 e terminando em *v*. Se *v* e q_i forem diferentes, retorna $r_2 = \gamma_{2v}$, caso contrário, $r_2 = \varepsilon$;
6. Executa-se a chamada de ação adaptativa $b_\neq$ com os mesmos argumentos passados para *x* e mais um argumento contendo r_2. A única ação dentro da função de $b_\neq$ é chamar $a_\neq$. Se r_2 for ε, a ação adaptativa com $b_\neq$ não é executada e a segunda condição (executar $a_\neq$ apenas se diferentes) é satisfeita;
7. Remove γ_{1i} e insere γ_{2i}, para manter a condição inicial de q_1 e q_2.

Com estes sete passos, a demonstração está feita.

Quando se executa uma ação de alteração de probabilidade ou remoção ou inserção com $\rho > 0$ para uma transição γ_{ij}, é necessário reajustar as probabilidades para as demais transições pertencentes a Γ_q de tal forma que (3) continue válida. Seja ρ' a nova probabilidade (considere ρ' igual a zero no caso de remoção) e ρ_{ij} a probabilidade antiga (considere ρ_{ij} igual a zero no caso de inserção). Então

$$\forall \gamma_{ik} \in \Gamma_q, k \neq j, \rho_{ik} = \rho_{ik} \bullet \frac{(1-\rho')}{1-\rho_{ij}} \qquad (10)$$

A linguagem $L(M)$ gerada por uma máquina de Markov adaptativa é sensível ao contexto [5]. É importante ressaltar que a definição acima estende o formalismo original em três pontos: (i) formaliza o conceito de funções adaptativas; (ii) inclui novas funções adaptativas e (iii) introduz a idéia de *adaptatividade de segunda ordem*, ou seja, a possibilidade da alteração da função adaptativa [14]. Aplicações destas extensões são demonstradas adiante, neste trabalho.

E. Sistema de Markov Adaptativo

Define-se um SMA (Sistema de Markov Adaptativo) como um conjunto de máquinas adaptativas de Markov que podem se relacionar através de novas ações adaptativas, introduzidas a seguir. Assim, pode-se estabelecer uma relação de escopo sobre as ações adaptativas definidas para cada máquina, classificando-as em: ações que modifiquem apenas a topologia local da máquina e ações que podem interferir no comportamento de outras máquinas pertencentes ao sistema. Desta forma, cada máquina M^k pertencente a um sistema de Markov adaptativo é definida como uma quíntupla:

$$M^k = (Q^k, \Sigma, T^k, {q_0}^k, F^k) \qquad (11)$$

em que Σ é o alfabeto de saída, comum a todas as máquinas do sistema, e Q^k, T^k, ${q_0}^k$ e F^k são como na definição de *Q*, *T*, q_0 e *F*, respectivamente.

Além dos tipos de ações elementares definidos para uma máquina de Markov adaptativa, as máquinas pertencentes a um SMA podem executar novos tipos, que operam sobre as outras máquinas do sistema. São eles:

- Dos mesmos tipos definidos para *M*, porém operando sobre outras máquinas do sistema;
- *Desativa* M^w, $w \neq k$. Desabilita a máquina M^w.Uma máquina desabilitada permanece em seu estado mais recente, não acionando qualquer transição ou executando qualquer ação adaptativa, até que seja novamente habilitada;
- *Ativa* M^w, $w \neq k$. Habilita a máquina M^w;
- *Consulta estado de* M^w, $w \neq k$. Consulta o estado atual (mais recente) de M^w e o retorna.

A definição de sistemas de Markov adaptativos serve apenas a fins práticos, uma vez que a distribuição do controle em várias máquinas não aumenta o potencial de computação do sistema [5].

III. A PLATAFORMA

O MMAdapt foi feito para acelerar o desenvolvimento de aplicações baseadas em sistemas de Markov adaptativos. Ao desenhá-lo, algumas diretrizes foram adotadas:

- Controle de execução: necessidade de criar um mecanismo que permita ao desenvolvedor ter o controle do instante em que as transições são acionadas;
- Opção de representação do sistema através de XML. Isso adiciona independência de linguagem (supondo que, eventualmente, haverá implementações do MMAdapt para outras linguagens), maior independência do resto do sistema e maior facilidade de compartilhamento;
- Futura criação de uma ferramenta gráfica, baseada no MMAdapt, para desenhar os sistemas de Markov

adaptativos, semelhante ao que o Adaptools faz para autômatos adaptativos [11];
- Permitir a customização do alfabeto de saída, através de implementação de uma interface disponibilizada pela plataforma.

Para viabilizar essas diretrizes, foi necessário fazer novas extensões ao formalismo, adaptando-o ao ambiente de execução. O texto abaixo, sempre que introduzir uma alteração, a ressaltará, de forma a não passar despercebida.

A. *Arquitetura Geral*

A arquitetura geral está mostrada na Fig. 1.

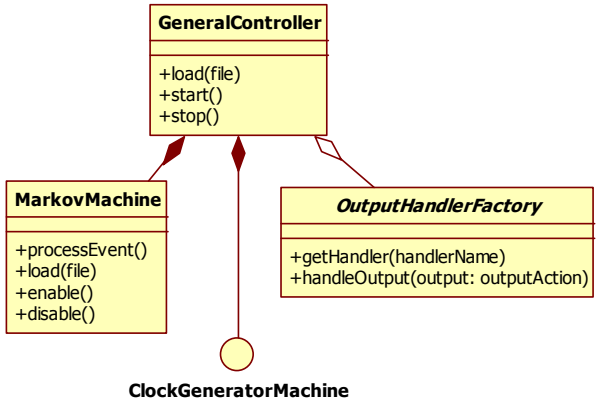

Figura 1. Diagrama com as principais classes do MMAdapt.

No MMAdapt, existem três classes principais que definem o comportamento do SMA:

- *MarkovMachine* (MM): implementa uma MMA. Ao menos uma precisa ser definida, adaptativa ou não;
- *ClockGeneratorMachine* (Máquina geradora de clock, ou CGM): determina o momento em que as transições das máquinas de Markov são acionadas, sem interferir nas probabilidades de escolha de transição. Ao menos uma ClockGeneratorMachine precisa ser definida;
- *OutputHandlerFactory* (OHF): permite estender o modo como o alfabeto de saída é interpretado. Sua existência é opcional.

A plataforma permite que a configuração do SMA seja definida através de um arquivo XML, como o mostrado na Fig. 2. Alternativamente, as classes podem ser criadas a partir de chamadas à API.

```
<?xml version="1.0" encoding="UTF-8"?>
- <root>
    <outputHandlerFactory name="composer.Band"/>
    <clock name="clock1" file="clock1.xml"/>
    <clock name="clock2" file="clock2.xml"/>
    <clock name="clock3" file="clock3.xml"/>
    <markov name="pri" file="primeira.xml"/>
    <markov name="ritmo" file="ritmo.xml"/>
    <markov name="regente" file="wta_regente.xml"/>
    <markov name="beat" file="rockbeat2.xml"/>
    <markov name="tempo" file="tempo.xml" enabled="false"/>
  </root>
```

Figura 2. Exemplo de arquivo de configuração de um SMA

Na Fig. 2 é possível observar as definições de várias MM e CGM, além de definir uma classe customizada para manipulação das saídas (*composer.Band*). A configuração de cada componente do sistema fica em arquivo próprio.

Para iniciar uma SMA, basta instanciar a classe *GeneralController*, chamar o método *load* passando como parâmetro o arquivo *.sma* e, finalmente, chamar o método *start*. Para parar a execução, basta chamar *stop* na mesma instância do *GeneralController*.

B. *Máquinas de Markov*

As máquinas de Markov implementadas no MMAdapt tem poucas mas importantes diferenças em relação ao formalismo apresentado anteriormente, introduzidas com base em sua utilização, visando aumentar a praticidade na construção de aplicações.

A primeira é relativa aos símbolos de saída, que são substituídos por *ações de saída* (*OutputAction*).

A segunda é que o conceito de função adaptativa é generalizado, passando a conter ações não só adaptativas, mas também ações de saída, entre outras. O nome do conceito generalizado é *função de Markov* e a transição passa a ser uma quádrupla,

$$\gamma_{ij} = (q_i, q_j, \rho, a_{ij}). \quad (12)$$

Em (12), a_{ij} é uma *ação de Markov* da forma

$$a_{ij} = f(\omega_1, \omega_2, ..., \omega_n) \cup \{\varepsilon\} \quad (13)$$

sendo f uma função de Markov declarada para a mesma máquina que contém a transição e $(\omega_1, \omega_2, ..., \omega_n)$ uma lista de argumentos que correspondem posicionalmente à lista de parâmetros de f.

Um exemplo de um arquivo simples de configuração de MM pode ser visto na Figura 3.

```
- <markov>
    <general name="beat" stepControl="clock1" initial="s1"/>
  - <outputHandler name="perscussion">
      <property name="texture" value="0"/>
      <property name="tempo" value="120"/>
    </outputHandler>
    <state name="s1"/>
    <state name="s5"/>
    <trans name="t1" src="s1" dest="s1" p="0.1" a="cym60"/>
    <trans name="t2" src="s1" dest="s5" p="0.9" a="lft"/>
    <trans name="t3" src="s5" dest="s1" p="0.9" a="cym60"/>
    <trans name="t4" src="s5" dest="s5" p="0.1" a="lft"/>
  - <function name="cym60">
    - <output>
        <property name="note" value="cymbal_1"/>
        <property name="volume" value="60"/>
      </output>
    </function>
  - <function name="lft">
    - <output>
        <property name="note" value="low_floor_tom"/>
      </output>
    </function>
  </markov>
```

Figura 3. Exemplo de MM sem funções adaptativas.

A primeira tag, chamada de *general*, serve para configurar propriedades gerais da máquina, como nome da máquina, nome do CGM que controla os passos e estado inicial.

A segunda tag, *outputHandler*, traz configurações adicionais sobre como manipular as saídas desta máquina específica. Ela só pode ser definida se o arquivo principal do SMA define um OHF. Porém, a definição do *outputHandler* na MM é opcional. Neste caso, as saídas serão direcionadas diretamente para o OHF.

Em seguida, observam-se três seções do XML. A primeira contém a definição de dois estados. A segunda contém a definição de quatro transições, cada uma contendo estado de origem, estado de destino, probabilidade e ação de Markov (veja que não há parâmetros para as funções utilizadas). A definição deste último é opcional, no caso de não haver qualquer ação a executar no disparo da transição.

Na última seção, estão as próprias funções de Markov. Há um tópico dedicado a elas no final deste capítulo.

C. *Máquinas Geradoras de Clock (CGM)*

Uma CGM é definida pela interface *CockGeneratorMachine*, que possui três métodos:

- *setAsControllerOf*: adiciona uma MM na lista de máquinas controladas. É chamada pelo *framework* durante a leitura do arquivo de máquina de Markov;
- *start*: inicia a execução da CGM. É chamada pelo framework de dentro do método *start* do *GeneralController*;
- *stop*: interrompe a execução. É chamada pelo *framework* de dentro de método *stop* do *GeneralController*.

A Figura 4 mostra os três tipos de CGM existentes.

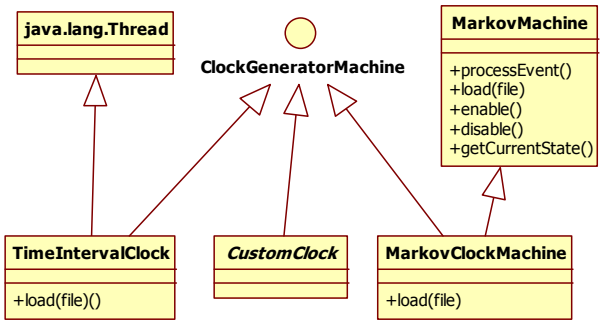

Figura 4. Diagrama de classe contendo os três tipos diferentes de clock.

São eles:

- *TimeIntervalClock*. Gera os clocks em tempos constantes. A Figura 5 contém um exemplo de aquivo de configuração desse gerador. O tamanho dos intervalos é definido em milissegundos. É declarada no arquivo *.sma* através da tag *clock*;

```
<?xml version="1.0" encoding="UTF-8"?>
<clock>
    <general timeInterval="200" name="clock1"/>
</clock>
```

Figura 5. Exemplo de máquina de clock gerado por intervalo de tempo

- *MarkovClockMachine*: É, ao mesmo tempo, uma MM e uma CGM. É declarada no arquivo *.sma* através da tag *markovClock*. Aciona as transições nas máquinas de Markov controladas através de uma ação especial, chamada de *ChangeStateAction*;
- *CustomClock*: É um gerador de clocks adaptado pelo desenvolvedor. Permite controle máximo da execução e é útil quando se deseja processar os eventos de acordo com um estímulo externo, ou processar a máquina de Markov através de um loop no código Java. Basta implementar a interface da CGM e declará-lo no arquivo *.sma* com a tag *customclock*. O nome da classe customizada deve ser passada no atributo *name*.

D. *OutputHandlerFactory e OutputHandlers*

Sempre que uma instância de *OutputAction* é encontrada durante o processamento de uma função de Markov, o *framework* obtém a instância do *OutputHandler* associada à máquina de Markov e chama o método *handleOutput*, cuja obrigação é, finalmente, processar a saída de fato.

O *OutputHandler* de cada máquina é obtido durante a leitura do arquivo da MM. Quando a tag *outputHandler* for encontrada, o *framework* obtém a instância do *OutputHandlerFactory*, declarado no arquivo principal do SMA, e chama o método *getHandler*, cuja obrigação é retornar um manipulador adequado ao nome passado como parâmetro.

A Fig.6 mostra as classes envolvidas.

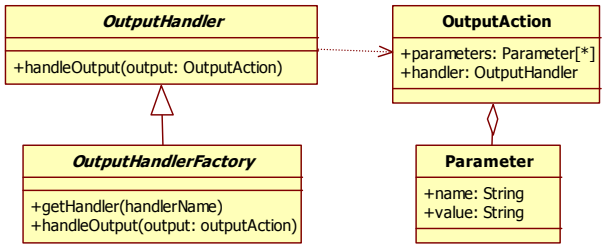

Figura 6. Diagrama com as classes para manipulação da saída de um MM.

O próprio OHF é, ele mesmo, um *OutputHandler*, utilizado sempre que o arquivo MM não declara a necessidade de uma instância específica. Outro fator importante é que todo SMA possui um OHF, mesmo que ele não tenha sido declarado. Essa instância implícita aceita como saída apenas caracteres, que serão direcionados para o *system.out* do Java. Nesta situação, qualquer propriedade do *OutputAction* é ignorada, e apenas o nome é impresso.

E. *Funções de Markov*

A função de Markov implementada pela plataforma é muito similar à do formalismo, tendo pequenas diferenças para aumentar a reutilização. Existem três grupos de dados que podem ser configurados em cada função:

- *Parâmetros*: Agrupados dentro da tag *parameters*. Opcional;
- *Variáveis*: Agrupados dentro da tag *vars*. Opcional;
- *Ações*: As ações devem vir ordenadas logo após o grupo das variáveis. A existência de pelo menos uma ação é obrigatória.

Algumas palavras são reservadas pelo MMAdapt e não podem ser utilizadas como nomes de variáveis, funções ou parâmetros. São elas:

- *this*: Palavra reservada para representar a transição que chamou a função. Por exemplo, é possível passar *this* como parâmetro ou usar *this.src* para obter o estado origem da transição atual;
- *each*: Palavra reservada para representar o valor de qualquer item em uma coleção. Pode ser utilizado em ações que recebem coleções como parâmetros: *UpdateProbabilitiesAction* e *UpdateFunctionsAction*;
- *null*: Palavra reservada para representar o valor nulo, que pode ser retornado pelas ações de consulta quando nenhuma transição atende ao filtro de pesquisa. Pode ser utilizada como parâmetro da ação *ChangeStateAction*.

F. Ações Elementares (Adaptativas e não adaptativas)

Neste tópico, são sucintamente apresentadas as ações atualmente suportadas pela plataforma de desenvolvimento. O objetivo é demonstrar as potencialidades atuais da plataforma.

Sempre que uma ação aceitar uma MM como parâmetro, ele é opcional, e quando não informado a ação é executada na máquina local.

Um parâmetro do tipo *limites de probabilidade* pode ter as seguintes formas:

- *Variável*: uma variável que contém o valor de probabilidade a atribuir;
- *Número*: valor exato de probabilidade;
- *Variável* "+" *Número*: combinação variável mais valor;
- *"[Combinação1,Combinação2]"*: combinação1 e combinação2 podem ser substituídos por qualquer uma das opções anteriores. Será gerado um valor aleatório de probabilidade, em tempo de execução, entre os valores calculados para as combinações.

1) FindUniqueAction

Descrição: Procura uma transição iniciando em q_i e terminando em q_j.
Parâmetros: Estado de origem, estado de destino, MM, parâmetro de retorno com a transição encontrada.
Retorno: Transição se encontrada, *null* em caso contrário.

2) SearchAction

Descrição: Procura por transições que obedeçam aos critérios de pesquisa.
Parâmetros: Estado de origem (opcional), estado de destino (opcional), MM, parâmetro de retorno com uma lista de transições.
Retorno: Lista de transições encontradas, *null* em caso contrário.

3) SetTransitionAction

Descrição: Configura uma transição conforme os parâmetros. Se a transição não existir, ela é criada.
Parâmetros: Estado de origem, estado de destino, função de Markov (pode ser nula), limites de probabilidade, MM.

4) RemoveTransitionAction

Descrição: Remove uma transição, que não será mais retornada pelas consultas.
Parâmetros: Lista de transições, MM.

5) UpdateProbabilitiesAction

Descrição: Atualiza as probabilidades das transições recebidas como parâmetro. Aceita a palavra reservada *each* como parte do parâmetro de limites de probabilidade.
Parâmetros: Lista de transições, MM, limites de probabilidade.

6) UpdateFunctionsAction

Descrição: Altera as funções de Markov das transições recebidas como parâmetro.
Parâmetros: Lista de transições, MM, função de Markov.

7) CompareVarsAction

Descrição: Compara duas variáveis e, dependendo do resultado, executa a função para igualdade ou para diferença. Pelo menos uma das duas funções precisa ser fornecida.
Parâmetros: variável 1, variável 2, função para igualdade (opcional), função para diferença (opcional).

8) EnableMachineAction

Descrição: Ativa uma máquina de Markov.
Parâmetros: MM.

9) DisableMachineAction

Descrição: Desativa uma máquina de Markov.
Parâmetros: MM.

10) ChangeStateAction

Descrição: Chama o método *processEvent* em uma MM, que procederá com a mudança de estado
Parâmetros: MM.

11) GetCurrentStateAction

Descrição: Consulta uma máquina de Markov para saber qual é o estado ativo.
Parâmetros: MM.
Retorno: Estado ativo em MM no momento do processamento da ação. Se ela estiver ocupada, no momento da chamada, com uma mudança de transição, o último estado ativo será retornado.

12) OutputAction

Descrição: Aciona o *OutputHandler* da máquina para receber a saída atual.
Parâmetros: nome da saída, lista de parâmetros.

G. Observações sobre a boa utilização da Plataforma

Algumas considerações devem ser feitas quanto a possíveis problemas relativos à concorrência. Cada máquina pode executar apenas uma tarefa simultaneamente, para evitar problemas de sincronismo. Por exemplo, uma máquina M^l não pode mudar de estado enquanto outra M^k, $l \neq k$, executa uma ação em M^l. Deste modo, a mudança de estado de M^l permanece em espera até que M^k termine.

Também é importante notar que o MMAdapt não impõe uma hierarquia de chamada às maquinas. Qualquer uma delas pode executar ações adaptativas em qualquer outra máquina do sistema. Isso pode fazer com que, eventualmente, ocorram *deadlocks* se o SMA não for bem desenhado, pelo menos na versão atual, devido ao modelo de sincronismo adotado.

Algumas situações, na prática, podem fazer com que a equação (3) seja totalmente incompatível com a ação adaptativa em execução. Isto acontece sempre que uma transição γ_{ij}, começando em q_i e tendo 100% de probabilidade de acionamento, e uma ação tenta diminuir essa probabilidade ou apagar a transição. Então (10) gera erro de divisão por zero e a execução do SMA falha – não há uma forma correta de

resolver este problema automaticamente. O melhor que o desenvolvedor pode fazer é alterar a função de Markov para, antes de diminuir a probabilidade de γ_{ij}, aumentar a probabilidade de pelo menos uma transição γ_{ik}, $k \neq j$.

Finalmente, a configuração inicial do SMA pode conter um estado q que não tem transições disponíveis para acionar. Nestes casos, a plataforma de execução será capaz de detectar o erro apenas quando, estando em q, um evento for acionado, mas, mesmo assim, não será capaz de corrigi-lo, e a execução do SMA é interrompida.

IV. CONCLUSÃO

O MMAdapt, como aqui descrito, já está implementado e funciona como base de uma aplicação para síntese automática de música, cujo nome é *Markovianas*. Os arquivos exibidos nas figuras 2, 3 e 5 fazem parte desta aplicação. O SMA gera as notas musicais para diversos instrumentos (guitarra, bateria e baixo) e as envia para uma classe que armazena pelo menos trinta segundos de música (isto é, é mantém um *buffer* musical) antes de começar a reproduzi-la.

Planos futuros relacionados ao MMAdapt incluem o desenvolvimento de uma ferramenta gráfica, como mencionado anteriormente, para permitir desenhar, executar e depurar um sistema de Markov adaptativo e exportar o XML deste SMA.

Além da geração de música, outras aplicações poderiam ser desenvolvidas na plataforma:

- Geração automática de texto (poesias, artigos, diálogos): uma abordagem utilizando SMA pode ser interessante se comparada às abordagens hoje utilizadas, como aquelas baseadas em templates, em técnicas evolutivas ou em gramáticas. Uma visão geral das técnicas para geração de poesias pode ser encontrada em [12];
- Geração automática de letras para música: uma abordagem que combine técnicas de geração automática de poesia àquelas utilizadas para o desenvolvimento das *Markovianas* pode ser interessante. Já existem trabalhos que demonstram a possibilidade de gerar letras a partir da melodia [13];
- Jogos: modelagem de decisões baseadas em probabilidade, como, por exemplo, quando se deseja que NPC's (*non-player character*, personagens que não são controlados pelo jogador) tomem decisões que o jogador não espera;
- Geração de casos de teste: a partir da modelagem do conjunto de dados de entrada e saída, documentos (telas, mensagens, etc.) e relacionamentos que caracterizam um sistema, e consequente classificação desses conjuntos em casos de teste positivos e negativos, pode-se desenvolver um SMA que gere milhares de casos aleatórios automaticamente;
- Geração de exercícios de aritmética (e de muitas outras disciplinas baseadas em regras matemáticas): pode-se seguir uma linha similar à da geração automática de casos de teste, porém voltada a aplicações pedagógicas, explorando a cobertura das regras em que se deseja efetuar o treinamento, bem como a variação do formato e da complexidade dos exercícios, automaticamente controlada por uma métrica voltada à avaliação do aproveitamento do aluno.

Pode-se concluir que o MMAdapt é uma ferramenta adequada para o desenvolvimento, pois tem uma API bem definida e reutilizável, pois delega toda a interpretação do alfabeto de saída para outras partes de um sistema mais abrangente. Mas é possível que, ao lidar com novas aplicações, alterações no código-fonte e na API sejam necessárias para o funcionamento desejado.

REFERÊNCIAS

[1] H. Pistori, "Tecnologia Adaptativa em Engenharia de Computação: Estado da arte e aplicações", *Tese de Doutorado*, EPUSP, São Paulo, 2003.

[2] J. C. Luz e J. J. Neto, "Tecnologa Adaptativa Aplicada à Otimização de Código em Compiladores", *IX Congreso Argentino de Ciencias de la Computación*, La Plata, Argentina, 6-10 de Outubro, 2003.

[3] D. P. Shibata, "Tradução Grafema-Fonema para a Língua Portugues Baseada em Autômatos Adaptativos", *Dissertação de Mestrado*, EPUSP, São Paulo, 2008.

[4] E. J. Pelegrini e J. J. Neto, "Applying Adaptive Technology in Data Security", *Proceedings of the 6th Peruvian Computer Week - JPC 2007*, Trjillo, Peru, pp. 31-40, Novembro 5-10, 2007.

[5] B. A. Basseto, "Um sistema de composição musical automatizada, baseado em gramáticas sensíveis ao contexto, implementado com formalismos adaptativos", *Dissertação de Mestrado*, EPUSP, São Paulo, 2000.

[6] J. J. Neto, "Contribuições à metodologia de construção de compiladores", *Post-Doctoral Tese de Livre Docência*, EPUSP, São Paulo, 1993.

[7] J. J. Neto, J. R. A. Junior e J. M. M. dos Santos, "Synchronized statecharts for reactive systems", *In: Proceedings of the IASTED International Conference on Applied Modelling and Simulation*. Honolulu, Hawaii, pp. 246-251, 1998.

[8] A. H. Tchemra, "Tabela de Decisão Adaptativa na Tomada de Decisão Multicritério", *Tese de Doutorado*, EPUSP, São Paulo, 2009.

[9] C. A. Bravo Pariente, "Gramáticas Livres de Contexto Adaptativas com Verificação de Aparênci", *Tese de Doutorado*, EPUSP, São Paulo, 2004.

[10] R. A. Howard, "Dynamic Probabilistic Systems", v. 1 e 2, 1a ed., *John Wiley & Sons, Inc.*, New York, 1971;

[11] L. Jesus, D. G. Santos, A. A. Castro Jr. e H. Pistori, "AdapTools 2.0: Aspectos de Implementação e Utilização", *Revista IEEE América Latina. Vol. 5, Num. 7*, ISSN: 1548-0992, pp. 527-532, novembro 2007.

[12] H. R. G. Oliveira, "Automatic generation of poetry: an overview", *Universidade de Coimbra*, Portugal, 2009.

[13] H. R. G. Oliveira, F. A. Cardoso e F. C. Perreira, "Exploring difference strategies for the automatic generation of song lyrics with TraLa Lyrics", *Proceedings of the Portuguese Conference on Artificial Intelligence*, Guimarães, Portugal, 2007.

[14] J. J. Neto, "Adaptive Technology and Its Applications", *Machine Learning: Concepts, Methodologies, Tools and Applications. IGI Global, 201*, doi:10.4018/978-1-60960-818-7.ch108, pp. 87-96, outubro 2011.

Daniel Assis Alfenas é formado em Engenharia da Computação pela Escola Politécnica da Universidade de São Paulo (EPUSP) em 2004. Trabalhou, nos últimos dez anos, nas diversas áreas do desenvolvimento de sistemas, desde a definição da demanda até a implantação do sistema. Recentemente tem trabalhado como coordenador técnico no desenvolvimento de sistemas complexos que envolvem simulação, otimização e interfaces ricas. Atualmente, é mestrando

no Laboratório de Técnicas Adaptativas da EPUSP e a área de pesquisa é a aplicação da tecnologia adaptativa no diálogo em linguagem natural entre usuários e sistemas.

Danilo Picagli Shibata é mestre em Engenharia da Computação pela Escola Politécnica da USP (EPUSP), com título obtido em 2008 pelo estudo da aplicação de autômatos adaptativos na tradução texto-fala para textos escritos na língua portuguesa. Trabalhou com Análise de segurança e confiabilidade de sistemas computacionais aplicados a sistemas críticos e desenvolvimento de software na área de segurança da informação. Trabalha atualmente no desenvolvimento de aplicação para simulação de transporte de fluídos.

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.

Marcos Ribeiro Pereira-Barretto é graduado em Engenharia Elétrica (1983), mestre em Engenharia Elétrica (1988) e doutor em Engenharia Mecânica (1993) pela Escola Politécnica da Universidade de São Paulo. É professor da Escola Politécnica da USP desde 1986. Atua nas seguintes áreas de pesquisa: robôs sociáveis, computação afetiva e arquitetura de sistemas para aplicações críticas como Automação Industrial.