Um *Framework* para Suporte ao Desenvolvimento de Aplicações Modeladas como Máquinas de Estados

Fernando Vasconcelos Mendes e Vitor Almeida dos Santos
Faculdade 7 de Setembro
fernandovm@gmail.com, vitor@fa7.edu.br

Resumo

Neste trabalho, descrevemos e avaliamos um framework de apoio ao desenvolvimento de sistemas computacionais cujo funcionamento seja adequadamente modelado através de máquinas de estados. Tal ferramenta propõe uma filosofia de desenvolvimento prática e objetiva, ao mesmo tempo em que orienta a produção de código com boas abstrações, alto grau de testabilidade, além de facilitar aspectos de manutenção devido ao melhor entendimento apoiado pelo uso de modelos visuais.

1. Introdução

Modelos adequados são requisitos importantes para o desenvolvimento de sistemas computacionais confiáveis. A escolha de modelos adequados depende, naturalmente, do comportamento e das infomações a serem modelados. Além disso, as ferramentas de suporte ao desenvolvimento de aplicações cada vez são mais capazes de manipular tais modelos de forma a agilizar a produção de código.

O paradigma orientado a objetos e seus modelos atendem satisfatoriamente muitas situações do mundo real ao representarem as entidades deste através de objetos, os quais reúnem dados (propriedades) e comportamento (métodos). A interação entre tais entidades é obtida pela troca de mensagens entre objetos.

Qualquer aplicação, no entanto, é, inerentemente, uma máquina de estados [1], visto que seu fluxo de execução e dados reagem, normalmente, a interações internas – com outros elementos do sistema, por exemplo – e externas – com pessoas ou outros sistemas. Muitas vezes, desenvolvedores abdicam da utilização de modelos adequados para máquinas de estados em decorrência da reduzida complexidade dos sistemas a serem desenvolvidos. Em geral, diagramas de sequência que esclarecem as trocas de mensagens entre objetos são suficientes para modelar os estados e transições do sistema. Contudo, há situações de complexidade considerável cujo modelo adequado é imprescindível. Como exemplos, podemos mencionar os protocolos de comunicação em geral e aplicações que possuem integração com centrais telefônicas, comumente chamadas aplicações *Computer Telephony Integration* (CTI) [2].

As ferramentas atuais de suporte ao desenvolvimento de aplicações notoriamente reconhecem e interagem com os modelos orientados a objetos fornecidos pela UML. Todavia, não há soluções de suporte ao desenvolvimento de aplicações modeladas como máquinas de estados amplamente difundidas e utilizadas. Podemos mencionar como exemplo de iniciativa neste sentido, a ferramenta UNIMOD [3], desenvolvida em Java. Tal ferramenta, por ser um *plugin* da IDE Eclipse [4], requer que modelo e implementação sejam realizados em um projeto criados na própria IDE.

A proposta deste trabalho é apresentar um *framework* para apoio ao desenvolvimento de aplicações modeladas como máquinas de estados. Esta ferramenta interpretará tais modelos, os quais poderão ser desenvolvidos por ferramentas comuns de diagramação disponíveis – um diferencial em relação ao UNIMOD –, através de representação XML. Desta forma, obtemos um fluxo da aplicação projetado de forma visual, com uma abordagem de desenvolvimento autodocumentavél, de fácil entendimento e, por conseqüência, gerando produtos de boa manutenibilidade. Pudemos validar o *framework* através da implementação de uma aplicação CTI a ser descrita adiante.

2. Máquinas de estados

A escolha dentre os modelos de máquinas de estados ocorre de acordo com o objetivo a ser alcançado.

Neste trabalho, adotamos, como se observa na prática, o emprego de um misto dos modelos propostos por Moore e Mealy [5], além de uma contribuição fundamental dada por David Harel conhecida como *Harel Statecharts* [6], [7], [8] e [9].

3. O *framework*

Um modelo de máquinas de estados representa os possíveis estados a serem assumidos por um sistema, bem como as transições entre esses estados. Perceba que ações

decorrentes de transições em uma aplicação representam, normalmente, as regras de negócios do sistema, de forma que devem ser implementadas em classes de negócios separadas do núcleo de execução da implementaçao da máquina de estado em questão. Assim, uma aplicação desenvolvida com o auxílio do framework proposto é a união da máquina de estados da aplicação – cuja implementação é fornecida pelo framework a partir do modelo dado pelo desenvolvedor – com as regras de negócios implementadas pelo desenvolvedor e utilizadas, quando necessárias, como eventos disparados nas transições da máquina de estados.

A arquitetura de funcionamento do framework é apresentada na Figura 1. As linhas pontilhadas indicam os responsáveis pela transformação de cada elemento de uma camada inferior no elemento da camada superior. Ou seja, os atores envolvidos transformação de elementos dos níveis de abstração mais altos até a aplicação. Na base da arquitetura, o problema a ser resolvido diz respeito ao conjunto de necessidades a serem atendidas por uma aplicação computacional. No nível 1 da arquitetura, temos o modelo em máquinas de estados daquele problema, o qual é fornecido pelo usuário (desenvolvedor).

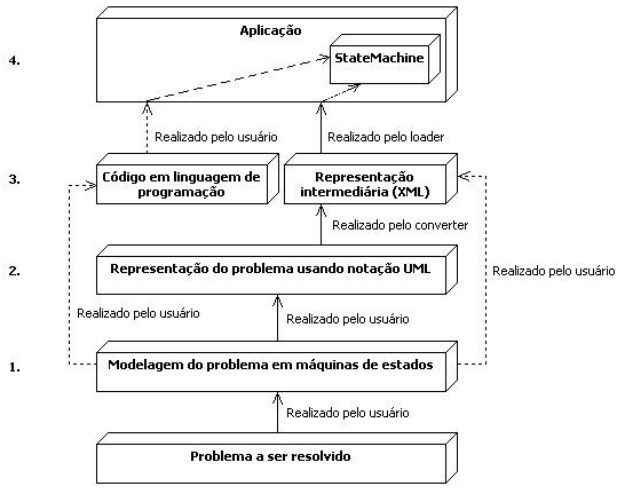

Figura 1: Arquitetura do *framework*

No nível 2, o framework requer uma representação em UML [10] da modelo de máquina de estados do nível 1. Neste nível 2, o framework admitirá uma representação em XMI [11], que é a aplicação XML para os diagramas da UML. Muitas ferramentas de diagramação em UML adotam tal representação.

No nível 3, temos duas possibilidades de representação. A "Representação intermediária (XML)" corresponde à representação interpretada pelo framework, o qual não entende arquivos XMI. Tal representação é baseada na proposta da SCXML [12], uma aplicação XML para a representação de máquinas de estados. Os arquivos desenvolvidos nesta representação intermediária são convertidos em código executável no nível 4 por elementos do framework chamados *loaders*, que funcionam como interpretadores. Conforme a Figura 2, o usuário tem a possibilidade de escrever imediatamente arquivos nesta representação intermediária, abrindo mão do modelo UML. A Figura 3 apresenta um exemplo de código de uma máquina de estados escrita em representação intermediária.

```
<?xml version="1.0" encoding="utf-8"?>
<scxml id="Exemplo01">
    <state id="Vermelho">
        <transition type="external" event="Tempo" >
            <target next="Verde" />
        </transition>
    </state>
    <state id="Amarelo">
        <transition type="external" event="Tempo" >
            <target next="Vermelho" />
        </transition>
    </state>
    <state id="Verde">
        <transition type="external" event="Tempo" >
            <target next="Amarelo" />
        </transition>
    </state>
    <initial>
        <transition type="external">
            <target next="Vermelho" />
        </transition>
    </initial>
</scxml>
```

Figura 3: Exemplo de representação intermediária em XML de máquina de estados

Ainda no nível 3 da arquitetura, o usuário tem a opção de fornecer uma representação do seu modelo de máquinas de estados em linguagem de programação, a partir das classes definidas para tal, suprimindo a representação em UML. Essa representação em linguagem de programação seria o próprio código executável do nível 4 da arquitetura.

```
//Montagem do modelo
State stRed = new State("Vermelho");
State stGreen = new State("Verde");
State stYellow = new State("Amarelo");

stGreen.AddTransition(new GenericTransition("Tempo",
                          stYellow));
stYellow.AddTransition(new GenericTransition("Tempo",
                           stRed));
stRed.AddTransition(new GenericTransition("Tempo",
                        stGreen));

//Inicialização do sistema baseado no modelo definido
StateMachine sm = new StateMachine("fsm",
  new State[] { stRed, stGreen, stYellow }, stRed);
//O sistema em execução
for (int i = 0; i <= 10; i++)
{
   Thread.Sleep(3000);
   sm.ProcessEvent(new GenericEvent("Tempo"));
}
```

Figura 4: Exemplo de representação de máquina de estados em linguagem de programação

O *framework* foi implementado fazendo uso da tecnologia .Net da Microsoft, em linguagem de programação C#.

3.1. Modelos de Execução

Durante o processo de modelagem de uma situação real através máquinas de estados precisaremos tratar a forma como a referida máquina irá lidar com os eventos externos direcionados a ela. O *framework* disponibiliza três modelos de execução.

- **Modo de espera**: Nesse modo uma instância de máquina de estados não ativa nenhum mecanismo autônomo de tratamento de eventos entrantes. Assim, o código cliente da referida classe ficará responsável por prover uma linha de execução para que o processamento evolua. Isso será feito através de chamadas a um método específico que tratará o evento obrigatoriamente de forma síncrona.
- **Modo baseado em temporização:** Nesse modo, uma instância de máquina de estados utiliza-se do mecanismo de temporização, presente no sistema operacional, para tratar automaticamente os eventos na periodicidade estabelecida. Neste caso usa-se o método específico, o qual tratará o evento de forma assíncrona.
- **Modo baseado em *thread*:** Nesse modo uma instância de máquina de estados utilizar-se-á de uma *thread* dedicada exclusivamente ao tratamento periódico dos eventos. Neste caso, usa-se um método específico, o qual tratará o evento de forma assíncrona.

A princípio, nenhum destes modelos exige maiores controles de concorrência, uma vez que é garantido o tratamento de apenas um evento por vez. Contudo, deve-se observar que, havendo múltiplas instâncias de máquinas de estado vinculadas às mesmas ações ou condições, é necessário assegurar que estas estejam preparadas para chamadas concorrentes. A escolha do modelo dependerá do cenário de aplicação e condições as quais a instância de máquina de estados irá se adequar. Caso a aplicação hospedeira já contenha um *pool* de *threads* dedicados ao tratamento de *requests* e *responses*, o primeiro modelo se adequará muito bem. Caso contrário, os demais modelos podem ser considerados.

4. Estudo de caso

Validamos o funcionamento do *framework* a partir da implementação de uma aplicação do mundo real e que se encontra em uso atualmente. Trata-se da aplicação *Tactium IP* descrita em [13] que é composta por um conjunto de módulos e aplicativos que juntos proporcionam um rico cenário de integração com centrais telefônicas. Através deles pode-se usar, de maneira integrada, grande parte dos recursos providos por uma central telefônica, a citar: geração, recebimento e distribuição de chamadas; gravação, reprodução, cópia de segurança e restauração de gravações de áudio; monitoração *online* do áudio das ligações; discagem automática; árvores IVR; supervisão dos estados de chamadas e operadores *online* etc.

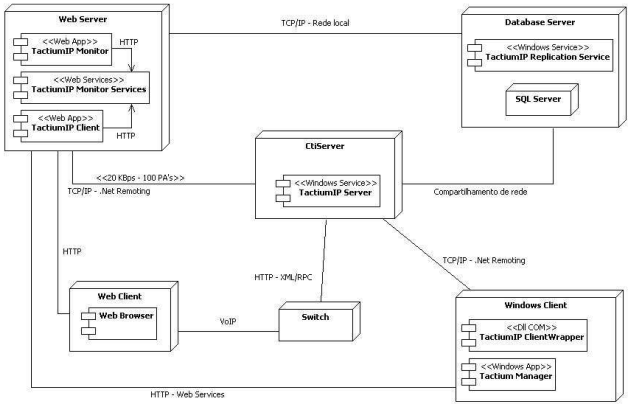

Figura 2: Componentes do *Tactium IP*

Grande parte da solução *Tactium IP* faz uso do *framework* proposto, incluindo o seu núcleo, representado na Figura 2 pelo componente *TactiumIP Server*, situado no nó *CtiServer*. Por ser uma aplicação de telefonia, o *Tactium IP* envolve complexas situações de comunicação e trocas de mensagens, os quais levam o sistema a assumir diversos estados no decorrer da sua execução. Assim, o modelo por máquinas de estados é bastante adequado à situação. O uso do *framework* foi, portanto, imediato a partir do modelo criado, amenizando consideravelmente o esforço de implementação.

Conforme dito anteriormente, o comportamento telefônico, em alto nível, ocorre segundo um protocolo definido. Dessa forma foram criados diagramas de estados para cada um dos dispositivos físicos manipulados pela solução, inclusive o contexto da comunicação com um dado cliente remoto.

Um dos grandes ganhos advindos dessa abordagem, principalmente no tocante a softwares complexos, é a previsibilidade imposta por ela. A premissa primeira é que nenhuma operação é permitida, apenas aquilo formalmente descrito nos diagramas de estados é aceito. A capacidade de termos um sistema solidamente disposto apenas um conjunto limitado e bem definido de estados acrescenta uma robustez significativa. Ressaltando ainda que isso é conseguido com um esforço de codificação ainda menor que na abordagem tradicional.

5. Conclusões e trabalhos futuros

Neste trabalho, propusemos um *framework* cujo objetivo é apoiar o desenvolvimento de aplicações cujo comportamento é melhor representado por máquinas de estados. Através da abordagem proposta, podemos modelar o comportamento de sistemas complexos através de representações gráficas – diagramas UML. O *framewok* criado fornece meios de traduzir tais diagramas para uma representação intermediária a partir da qual será provido o controle do fluxo de execução correspondente.

Através do estudo de caso mencionado, pudemos perceber as principais vantagens do uso do *framework*, citadas a seguir.

- **Documentação**: Uma das grandes dificuldades enfrentadas em projetos de software – que é a geração, e principalmente manutenção, de documentação – foi satisfatoriamente atenuada. O processo de desenvolvimento através de máquinas de estados, por ser inerentemente gráfico e claro, leva a geração implícita de documentação comportamental, fornecido pelos próprios modelos.
- **Abstrações e responsabilidades**: Na arquitetura do *Tactium IP* há uma central telefônica que envia eventos para o aplicativo. A máquina de estados que coordena o fluxo de execução da aplicação, no entanto, não tem nenhum vínculo com a central. Ela apenas manipula eventos. A substituição da central telefônica simplesmente por uma entidade que gere eventos para a máquina de estados manteria o funcionamento do sistema. Tal tratamento de abstrações e correta separação de responsabilidades exerce papel fundamental em um projeto de boa qualidade, de maneira que o desenvolvimento baseado em máquinas de estados influencia o processo orientando para esse objetivo.
- **Testabilidade**: A capacidade de se testar aplicações de forma automatizada é uma qualidade importante. Devido à boa separação de responsabilidades, como mencionamos anteriormente, conseguimos representar os cenários aos quais o aplicativo pode ser submetido como testes de unidades, cada um destes cenários contendo uma dada seqüência de eventos. A validação dos testes se deu de duas formas: pelos estados pelos quais passou o sistema no decorrer do cenário e pelas ações disparadas por ele durante esse processo.

5.1. Trabalhos futuros

A implementação atual do *framework* é capaz de lidar com uma infinidade de situações do mundo real, permitindo-nos representar sistemas verdadeiramente complexos. No entanto, há, pelo menos, dois aspectos que, uma vez implementados, estenderão as capacidades da ferramenta: condições embutidas e eventos internos motivados pelo tempo. Condições embutidas são aquelas expressões lógicas especificadas no próprio modelo UML. Tais expressões oferecem a vantagem de introduzirmos no próprio modelo UML a avaliação de condições mais complexas. Na versão atual do *framework*, condições são implementadas por entidades externas ao modelo de máquinas de estados, as quais são requisitadas quando tais condições precisam ser avaliadas. O outro aspecto – eventos internos motivados pelo tempo – possibilita a modelagem de sistemas com restrições temporais, ou seja, sistemas nos quais algumas transições devam ocorrer caso um evento externo não seja percebido em um dado limite de tempo. Isto normalmente é utilizado para tomar medidas reativas uma vez que um limite previsto foi violado

6. Referências

[1] WAGNER, F. et al. “Modeling Software with Finite State Machines: A Practical Approach”, AUERBACH, 2006.

[2] WALTERS, R. “Computer Telephony Integration”, Artech House, 1999.

[3] http://unimod.sourceforge.net/

[4] http://www.eclipse.org/

[5] BOERGER E., STAERK R., “Abstract state machines. A method for high-level system design and analysis”, Springer, 2003.

[6] HAREL, D., “The STATEMATE Semantics of Statecharts”, ACM Transactions on Software Engineering and Methodology, 1996.

[7] HAREL, D., “STATEMATE: A Working Environment for the Development of Complex Reactive Systems”, IEEE Transactions on Software Engineering, 1990.

[8] HAREL, D., “Statecharts: A visual Formalism for complex systems”, The Science of Computer Programming, 1987.

[9] HAREL, D., “Executable Object Modeling with Statecharts”, IEEE Computer, Vol. 30, issue 7, Julho 1997.

[10] BOOCH, G., RUMBAUGH, J., JACOBSON I., “The Unified Modeling Language User Guide”, Addison-Wesley, 2nd ed, 2005.

[11]http://www.omg.org/technology/documents/modeling_spec_catalog.htm#XMI

[12] http://www.w3.org/TR/scxml

[13] http://www.softium.com.br/tactium.asp