Modelos Orientados a Estado na Especificação de Software

State-Oriented Models in Software Specification

Adilson Luiz Bonifácio[1]; Fabio Adriano Lisboa Gomes[2]

Resumo

Diversas técnicas de especificação estão sendo usadas no processo de desenvolvimento de software. Essas técnicas podem ser ou não formais, de acordo com o sistema em desenvolvimento. Neste trabalho, uma técnica de modelagem formal é aplicada num estudo de caso. Aqui, o modelo de Máquina de Estados Finita é usado para especificar as funcionalidades de uma calculadora, a qual modela as operações básicas de aritmética.
Palavras-chave: máquina de estados finita; modelagem formal; especificação de sistemas; desenvolvimento de software.

Abstract

Several specification techniques are being used in software development process. These techniques can be formal or not according to the developing system. In this work, a formal modeling technique is applied in a case study. The Finite State Machine model is used to specify the calculator functionalities, which models the basic arithmetical operations.
Key words: finite state machine; formal modeling; system specification; software development.

Introdução

O aumento da complexidade dos sistemas e o surgimento de sistemas mais rigorosos conduziram o panorama de desenvolvimento de software à utilização de técnicas mais avançadas e precisas. Esse fator tem influenciado, principalmente, as fases de especificação e modelagem de sistemas.

Sistemas complexos, em especial sistemas reativos (BONIFÁCIO; MOURA, 2000, BONIFÁCIO et al., 2000), exigem um grau de precisão muito alto em seus modelos para garantir a execução segura e correta de suas funcionalidades. Sistemas menos complexos, ao contrário que se possa pensar, também exigem a utilização de técnicas e modelos que auxiliem o processo de software. Esse fato se deve, principalmente, ao aumento de competitividade. O fornecimento de software de qualidade tem cooperado com essa competitividade, fazendo com que o processo de software seja seguido tanto para sistemas complexos, quanto para sistemas mais simples.

Dentro do processo de software, um dos modelos mais utilizados para especificação é a máquina de estados finita (HOPCROFT; ULLMAN 1979;

[1] Professor do Departamento de Ciência da Computação da Universidade Estadual de Londrina. Doutorando pelo Instituto de Computação da Unicamp. bonifacio@uel.br
[2] Doutorando pelo Departamento de Comunicações (DECOM), Faculdade de Engenharia Elétrica e Computação (FEEC), Unicamp.adrianol@decom.fee.unicamp.br.

CARROLL; LONG, 1989). Sua simplicidade e representação visual são aspectos que facilitam a especificação de sistemas. Extensões mais elaboradas desse modelo, como os autômatos híbridos, são utilizadas para a especificação e a verificação de sistemas complexos (BONIFÁCIO; MOURA, 2000; BONIFÁCIO et al., 2000).

A aplicação apresentada neste trabalho diz respeito à modelagem de uma calculadora, utilizando as máquinas de estados finitas. Uma calculadora possui funcionalidades e operações bem definidas, como expressões aritméticas que consistem de cálculos matemáticos. Por essa razão, a necessidade de uma especificação do sistema que representa tais funcionalidades.

A utilização de uma calculadora simples não minimiza a importância quanto aos resultados obtidos neste trabalho. O intuito é mostrar de que forma os problemas, geralmente resolvidos através de técnicas não formais, podem ser resolvidos com soluções expressas por modelos formais, usando rigor matemático. Nesse sentido, não foram encontrados trabalhos com tal abordagem formal para uma calculadora simples, exemplificando as etapas de sua utilização na construção de sistemas de software.

As próximas seções estão organizadas como segue. Na seção 2, é discutida a importância da modelagem de sistemas na solução de problemas. O modelo de máquinas de estados finitas é apresentado com maiores detalhes na seção 3. A aplicação do modelo explorado neste trabalho para o desenvolvimento de uma calculadora é mostrada na seção 4. O trabalho termina com algumas conclusões e considerações finais.

O Papel da Modelagem na Solução de Problemas

Imagine a construção de um prédio, na qual nenhum planejamento prévio tenha sido elaborado pelo engenheiro responsável. Nesse cenário, os andares serão erguidos um após o outro por meio de tentativa e erro. Com a construção de alguns andares é provável que toda a edificação venha abaixo, uma vez que nenhum cálculo foi realizado antes da construção para se determinar o quão forte deveriam ser os alicerces. A solução seria reforçar os alicerces da construção para suportarem o peso dos pisos superiores e, então, recomeçar a edificação. Esse ciclo, de constrói e cai, repete-se até que, finalmente, o último andar do edifício seja finalizado, sem que o prédio desabe.

O processo por tentativa e erro adotado apresenta sérios defeitos:

1. Risco de vida aos trabalhadores;
2. Grande desperdício de recursos materiais e financeiros;
3. Desperdício de tempo;
4. Uma vez construído, nada garante que o prédio não venha a desabar no futuro.

Apesar de absurdo, o cenário descrito acima ocorre com freqüência, de maneira análoga, na área de desenvolvimento de sistemas. Alguns programadores e analistas de sistemas desenvolvem seus produtos com base na tentativa e erro, sem a utilização de um método mais adequado. Os resultados são aplicativos mal feitos, cujo desenvolvimento demorou mais do que deveria, exigindo constantes manutenções. Com o objetivo de se criar bons produtos, torna-se necessário planejar cada etapa envolvida na fabricação do produto.

Dessa forma, a modelagem do problema a ser resolvido assume um papel de fundamental importância, fornecendo efetivamente uma solução adequada do problema enfrentado. Na construção de edifícios, por exemplo, o modelo do problema consiste, basicamente, num conjunto de equações matemáticas cuja solução irá determinar o quanto de concreto deve ser utilizado na construção do prédio.

Em se tratando de software, uma possível modelagem, para capturar o comportamento de um sistema, pode ser realizada com as *Máquinas de Estados* (HOPCROFT; ULLMAN, 1979; CARROLL; LONG, 1989). No desenvolvimento de

software, a modelagem de problemas pode ser feita através de diversas técnicas e métodos, tais como o diagrama de fluxo de dados, o modelo de entidade relacionamento e os modelos orientados a objetos. Cada uma dessas técnicas captura um aspecto distinto do processo de desenvolvimento, sendo observadas em diferentes fases, para diferentes tipos de sistemas, além de proporcionar diferentes níveis de abstração para o entendimento do que se deseja modelar.

Máquinas de Estados Finitas

Uma máquina de estados finita, FSM (*Finite State Machine*), é um modelo orientado a estado, muito usado para a descrição de sistemas de controle. O comportamento temporal desses sistemas é representado pelos estados e pelas transições entre estados do modelo. É importante salientar que a modelagem do comportamento temporal numa FSM é discreta. Algumas extensões, como *timed-automata* ou autômatos híbridos, modelam, além do comportamento discreto, o comportamento dinâmico dos sistemas (BONIFÁCIO; MOURA, 2000; BONIFÁCIO et al., 2000).

Basicamente, uma FSM (HOPCROFT; ULLMAN, 1979; CARROLL; LONG, 1989) é composta de um conjunto de estados, um conjunto de transições entre os estados e um conjunto de ações associadas a estes estados ou transições. De maneira formal, uma FSM é uma quíntupla $< S, I, O, f : SxI \rightarrow S, h : SxI \rightarrow O >$, onde S={$s_1$, s_2, ..., s_l} é o conjunto de estados, I={i_1, i_2, ..., i_m} é o conjunto de entradas e O={o_1, o_2, ..., o_n} é o conjunto de saídas. A função de *próximo estado*, *f*, determina o próximo estado para o qual o controle da FSM deve passar, a partir do estado corrente e da entrada. A função de *saída*, *h*, determina as saídas, também, a partir da entrada e do estado corrente. Cada FSM possui um estado inicial e um conjunto de estados finais, quando o sistema não possui execução infinita.

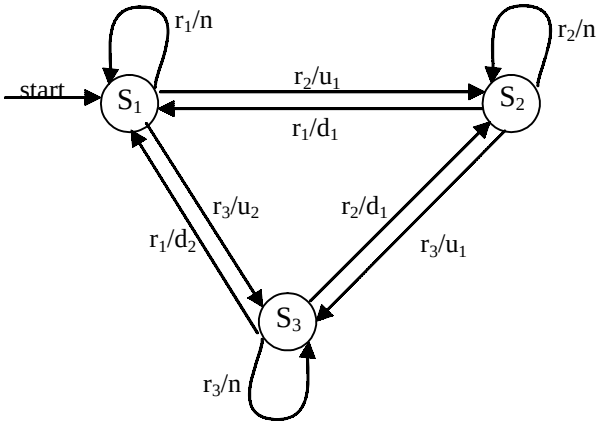

Figura 1 – Modelo FSM do controlador de um elevador

A Figura 1 mostra um exemplo de uma FSM que modela o controlador de um elevador num prédio com três andares. No modelo, o conjunto de entradas I={r_1, r_2, r_3} representa os andares requisitados. Por exemplo, a entrada r_2 significa que o segundo andar foi requisitado. Já o conjunto de saídas, O={d_2, d_1, n, u_1, u_2}, representa a direção e o número de andares que o elevador deve percorrer. Por exemplo, d_2 significa que o elevador deve descer dois andares, u_2 significa que o elevador deve subir dois andares, e n diz ao elevador para ficar parado. Na ilustração, pode-se notar que se o andar corrente for o segundo, S_2, e o primeiro andar for requisitado, entrada r_s, então a saída será d_1.

Paradigmas de modelagem formal

Diversos modelos podem ser encontrados na literatura com o objetivo de auxiliar o processo de verificação, validação e desenvolvimento de sistemas, tais como Redes de Petri e *Statecharts*.

O modelo de redes de Petri (MURATA, 1989) é utilizado, mais freqüentemente, na especificação de sistemas que expressam interação concorrente de tarefas. O modelo consiste de um conjunto de *lugares* (círculos na Figura 2, nomeados pela letra *p*), um conjunto de transições (barras horizontais, designadas pela letra *t*) e um conjunto de *marcas* (pontos no interior dos *lugares*). As *marcas* residem nos *lugares* e circulam pela rede de Petri, sendo

produzidos e consumidos por meio dos disparos das transições. Veja o exemplo da Figura 2.

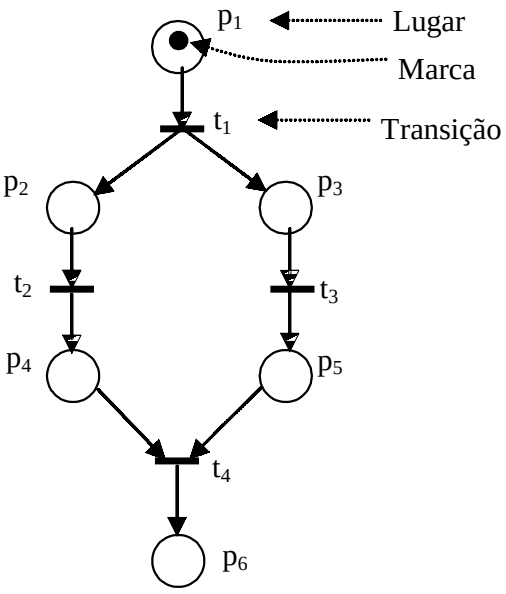

Figura 2 – Exemplo de uma Rede de Petri

Os *Statecharts* (HAREL, 1987), também são um modelo orientado a estado, uma extensão das tradicionais FSM. O modelo suporta hierarquia e concorrência, diminuindo o problema da explosão combinatória de estados quando da modelagem de sistemas grandes e complexos. Como nas FSM, os *Statecharts* possuem estados e transições, entretanto, cada estado pode ser decomposto em subestados modelando, dessa forma, a hierarquia. Além disso, os estados podem ser decompostos em subestados concorrentes, os quais são executados em paralelo, se comunicando através de variáveis globais. As transições entre estados podem ser estruturadas ou não estruturadas. As transições estruturadas são aquelas que podem ocorrer apenas entre estados do mesmo nível hierárquico. Já as transições não estruturadas são aquelas que podem ocorrer entre quaisquer estados, considerando um relacionamento hierárquico. Observe na Figura 3 o exemplo de um *Statecharts*.

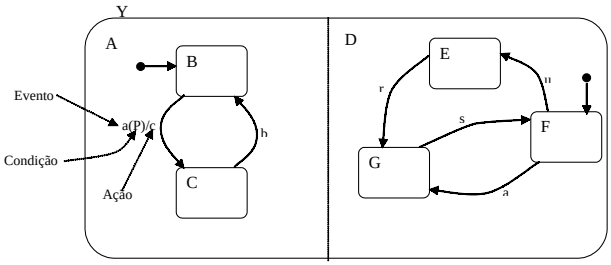

Figura 3 – Statecharts: Estados concorrentes hierárquicos

Aplicação dos paradigmas de modelagem

Quanto à aplicação dos diferentes paradigmas de modelagem nota-se, por um lado, a existência de métodos com um nível de abstração mais alto, como o modelo estruturado e orientado a objetos, inerentemente visuais. Por outro lado, existem os modelos formais que capturam um nível de abstração mais rigoroso. Dentre os modelos formais, existem as técnicas que utilizam recursos visuais, como as redes de Petri e os Statecharts, outros modelos e linguagens de especificação que são semivisuais, como Z e VDM, se aproximando mais de formulações matemáticas, e por fim, as linguagens de especificação sem recursos visuais, como é o caso de MAL e álgebra de processos.

O fato é que os objetivos das aplicações podem ser diferentes, exigindo modelos distintos. No caso de Z e VDM, são utilizadas as notações matemáticas para a representação e verificação de domínios e limites de requisitos de variáveis e características de um determinado sistema.

Os modelos orientados a estado são usados devido a sua representação precisa, quanto ao comportamento temporal e seqüencial. Esses modelos estão preocupados com os eventos e ações que ocorrem no sistema e suas transformações, além de possuir uma representação gráfica que facilita a visualização do modelo.

Já os modelos tradicionais de engenharia de software, como análise estruturada e orientação a objeto, estão centrados no comportamento estático do sistema, bem como na modelagem de dados e suas representações.

A UML (*Unified Modeling Language*), por exemplo, é um modelo orientado a objetos usado para análise e projeto de software. O modelo é composto por um conjunto de diagramas para documentar, especificar, visualizar e construir sistemas utilizando o paradigma orientado a objetos (MELO, 2002; LARMAN, 2000). Dentre esses diagramas, pode-se salientar os diagramas de classe, diagramas de casos de uso e, em particular, os diagramas de estado.

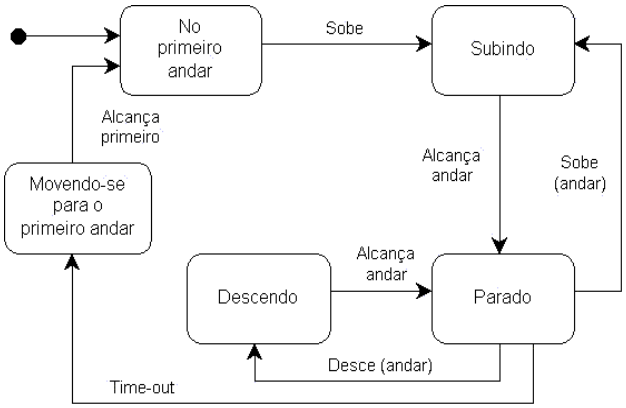

Figura 4 – Diagrama de Estado do UML para o exemplo do Elevador

O diagrama de estado do UML foi adicionado a esse conjunto de diagramas para dar visibilidade quanto ao comportamento temporal de forma precisa. Essa característica permite ao modelo capturar o ciclo de vida dos objetos, especificando o comportamento e como estes objetos diferem em função do estado atual. Os eventos ilustrados no diagrama indicam a alteração dos estados dos objetos da classe. Dessa forma, o diagrama expressa os estados que um objeto pode assumir e como os eventos, tais como recepção de mensagens, evolução do tempo, erros e condições diversas, afetam estes estados ao longo do tempo.

Na Figura 4, está representado o controlador de um elevador, modelado pelo diagrama de estado do UML, basicamente o modelo FSM clássico usado anteriormente, porém não tão rigoroso.

Modelando uma Calculadora em Máquina de Estado

Uma calculadora que fornece as funções básicas de aritmética matemática pode ser ilustrada como apresentada na Figura 5. Essa calculadora fornece as quatro operações básicas da aritmética, somar, subtrair, multiplicar e dividir, um botão para inverter o sinal de um número (+/-), um botão para limpar o visor (CE), outro para desligar a calculadora (off) e, finalmente, o botão que executa um cálculo, retornando seu resultado (=).

A calculadora apresentada foi implementada em Visual Basic 6 (REED, 2000). Na programação, foi necessário armazenar resultados parciais das operações obtidas, bem como os operadores selecionados pelos usuários. Para essas tarefas, foram utilizadas as variáveis *fResult*, para armazenar os resultados parciais e *sOp* para armazenar o operador pressionado pelo usuário. Os valores que a variável *sOp* pode assumir são:

- "+" se o usuário pressionar o botão adição;
- "-" se o usuário pressionar o botão de subtração;
- "*" se o usuário pressionar o botão de multiplicação;
- "/" se o usuário pressionar o botão de divisão.

Quando ocorre um evento na calculadora, efetuado pelo usuário, ações de alterações das variáveis devem ocorrer. Ao pressionar um número ou uma operação matemática na calculadora, por exemplo, as variáveis fResult e sOp são alteradas, armazenando o número e a operação a ser executada.

Figura 5 – Interface de uma calculadora padrão

Modelo da Calculadora

As ações a serem realizadas em resposta aos eventos provocados pelos usuários são mostradas na Tabela 1. Os campos dessa tabela têm o seguinte significado:

1. ESTADO ORIGEM: Indica o estado (situação) da calculadora antes do usuário provocar o evento.

Os possíveis estados para o modelo da calculadora são 0, 1, 2 e 3, sendo 0 seu estado inicial.

2. EVENTO: Indica o evento provocado pelo usuário. Os eventos podem ser:

a. Pressionar o botão CE: Ocorre quando se reinicia a calculadora.

b. Pressionar "=": Ocorre quando o usuário pressiona o botão "=", para finalizar uma computação.

c. Pressionar um número (Num): Ocorre quando o usuário pressiona algum dígito (1,2,3,4,5,6,7,8,9,0).

d. Pressionar um operador (Op): Ocorre quando o usuário pressiona "+", "-", "*" ou "/".

3. ESTADO DESTINO: Indica para qual estado a calculadora passará o controle após o evento provocado pelo usuário.

4. O QUE FAZER: Indica a ação que deve ocorrer em resposta ao evento provocado pelo usuário. Por exemplo, a linha 3 da tabela diz que, com o estado corrente em 0 e o usuário pressionando algum número, a calculadora deve ir para o estado 1 e o visor da calculadora mostrará o número pressionado (Visor = Num).

O modelo da máquina de estados finita descrito na Tabela 1 também pode ser representado de modo visual por meio do diagrama da Figura 6. Neste caso, os possíveis estados da calculadora são representados pelos círculos nomeados como 0, 1, 2 e 3. A ligação entre os estados da FSM, através de arcos, indica a mudança do controle da FSM. A cada transição é associado um evento provocado pelo usuário, juntamente com sua respectiva ação que deve ser realizada com a ocorrência do evento. Observe que diferentes eventos, partindo de um mesmo estado origem para um mesmo estado destino, são expressos numa única transição para indicar a mudança.

Entendendo o Modelo da Calculadora

O modelo da calculadora descrito na Tabela 1, ou no seu equivalente visual mostrado na Figura 6, permite ao programador saber com precisão o que deve acontecer em resposta aos eventos provocados pelo usuário. Suponha que um usuário deseja realizar a soma 1 + 3. Primeiramente, o usuário pressiona o número 1, em seguida, o operador aritmético + e, depois, o número 3. Por fim, o usuário pressiona o sinal de igualdade "=" e, então, a calculadora mostrará o número 4, como resultado da operação. Abaixo são mostradas as ações que o programa deverá realizar em resposta a cada um dos eventos provocados pelo usuário no exemplo apresentado. Veja a seguir a análise do exemplo, partindo do estado inicial 0:

Tabela 1 – Representação da FSM da Calculadora

ESTADO ORIGEM	EVENTO	ESTADO DESTINO	O QUE FAZER
0	CE	0	fResult = 0, Visor = 0, sOp = " "
0	=	0	fResult=Visor
0	Num	1	Visor = Num
0	Op	2	sOp = Op, fResult = Visor
1	CE	0	fResult = 0, Visor = 0, sOp = " "
1	=	1	fResult=Visor
1	Num	1	Visor = Visor + Num
1	Op	2	sOp = Op, fResult = Visor
2	CE	0	fResult = 0, Visor = 0, sOp = " "
2	=	0	fResult = fResult sOp Visor, Visor = fResult
2	Num	3	Visor = Num
2	Op	2	sOp = Op
3	CE	0	fResult = 0, Visor = 0, sOp = " "
3	=	0	fResult = fResult sOp Visor, Visor = fResult
3	Num	3	Visor = Visor + Num
3	Op	2	fResult = fResult sOp Visor, Visor = fResult, sOp = Op

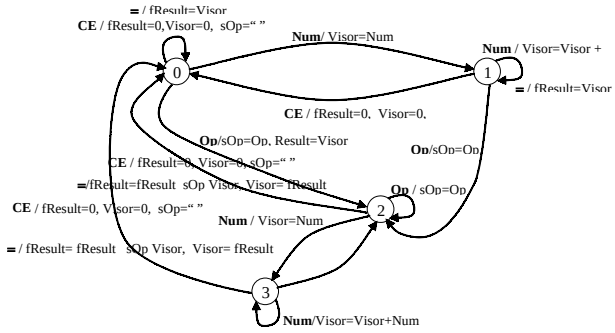

Figura 6 – Modelo da Calculadora em Diagrama de Estado

USUÁRIO CLICA NO NÚMERO 1

O controle está no estado inicial 0 e o usuário clica no número 1. A linha 3 da Tabela 1 diz o seguinte:

1. Mostrar o número 1 no Visor da calculadora.

2. Levar a calculadora para o estado 1.

Em termos de programação, os procedimentos acima poderiam ser mapeados para o código *visual basic* a seguir, onde *txtVisor* é o nome da caixa de texto que forma o visor da calculadora mostrada na Figura 5.

```
' verifica se o estado corrente é o estado 0
If (iState = 0) then
    txtVisor.text = "1"
    iState = 1
end if
```

USUÁRIO CLICA NO OPERADOR DE ADIÇÃO, "+"

Com o controle no estado 1, o usuário clica no operador "+". Neste caso, a linha 8 diz que :

1. sOp = "+", salva o operador pressionado pelo usuário.

2. fResult = 1, fResult recebe o número mostrado no Visor.

3. o controle da calculadora vai para o estado 2.

```
' verifica se o estado corrente é o estado 1
If (iState = 1) then
    sOp = "+"
    fResult = CDbl(txtVisor.Text)
    iState = 2
end if
```

USUÁRIO CLICA NO NÚMERO 3

A partir do estado 2, o usuário clica no número 3. A linha 11 da Tabela 1 diz que:

1. O visor da calculadora deve apresentar o número 3.

2. O controle passa para o estado 3.

```
' verifica se o estado corrente é o estado 2
If (iState = 2) then
    txtVisor.text = "3"
    iState = 3
end if
```

É importante frisar que o valor anteriormente mostrado pelo visor da calculadora foi salvo em *fResult* durante a transição do estado 1 (um) para o estado 2 (dois), como mostrado no passo anterior. O valor de *fResult* será utilizado para o cálculo do valor final da operação, como pode ser visto no próximo item.

USUÁRIO CLICA NO SINAL DE IGUALDADE, "="

No estado 3 o usuário clica no sinal de "=". A linha 14 indica que:

1. fResult = fResult sOp Visor, como fResult = 1 , sOp = "+" e Visor = 3, isso significa que fResult = 1 + 3, ou seja, fResult = 4.

2. A calculadora volta ao estado inicial 0.

3. Visor = fResult, mostra o resultado da operação, o valor 4, no visor da calculadora.

```
If (iState = 3) then
    iState = 0
    'Lê o conteúdo do visor, convertendo-o para um valor numérico
    Temp = CDbl(txtVisor.text)
    'Faz cálculo de acordo com a operação contida em sOp
    If (sOp = "+") then
        fResult = fResult + Temp
    elseif (sOp = "-") then
        fResult = fResult – Temp
    elseif (sOp = "*") then
        fResult = fResult * Temp
    elseif (Temp <> 0) then
        fResult = fResult / Temp
    else
        txtVisor.Text = "Erro de divisão por zero"
        Exit sub
    End if
    txtVisor.Text = fResult
end if
```

Conclusão e Considerações Finais

Os métodos e as técnicas formais, ou mesmo semiformais, auxiliam de modo mais preciso e eficiente o desenvolvimento de sistemas de software. Isto pode ser observado nos diferentes projetos de sistemas, utilizando as diferentes metodologias e modelos de acordo com as necessidades do projeto.

Neste trabalho é abordado o modelo de máquina de estado, uma das formas de se especificar sistemas. O estudo de caso apresentou a especificação do projeto de uma calculadora simplificada. O modelo descrito na Tabela 1 e representado pela FSM da Figura 6, permite, de maneira precisa, verificar o que deve ser feito em resposta aos eventos gerados pelo usuário. Isso torna possível e/ou facilita muito o trabalho de implementação de um sistema, como pôde ser observado no código VB gerado a partir da FSM apresentada na seção 4. A especificação, usando uma FSM, constitui uma importante ferramenta de modelagem para a criação de um sistema confiável, diminuindo as perdas com relação a tempo, dinheiro, ou riscos para as pessoas que dependem do sistema, em caso de sistemas críticos.

Referências

BONIFÁCIO, A. L.; MOURA, A. V. Modeling and Parameters Synthesis for an Air Traffic Management System. In: INTERNATIONAL CONFERENCE ON FORMAL METHODS IN COMPUTER AIDED DESIGN, 3rd., 2000, Austin. *Proceedings* .... Austin: Springer-Verlag, 2000. p. 316–334.

BONIFÁCIO, A. L. et al. Formal Parameters Synthesis for Track Segments of the Subway Mesh. In: ANNUAL IEEE INTERNATIONAL CONFERENCE AND WORKSHOP ON THE ENGINEERING OF COMPUTER BASED SYSTEMS, 7th., 2000, Edinburgh. *Proceedings*.... Edinburgh: IEEE Computer Society Press, 2000. p. 263–272.

CARROLL, J.; LONG, D. *Theory of Finite Automata*. Englewood Cliffs, NJ: Prentice Hall, 1989.

HAREL, D. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, Amsterdam, v. 8, n. 3, p. 231-274, June, 1987.

HOPCROFT, J. E.; ULLMAN, J. D. *Introduction to Automata Theory, Languages and Computation*. Reading, MA: Addison–Wesley, 1979.

LARMAN, C. *Utilizando UML e padrões*: uma introdução à análise e ao projeto Orientada a Objeto. Porto Alegre: Bookman, 2000.

MELO, A. C. *Desenvolvendo aplicações com UML*. Rio de Janeiro: Brasport, 2002.

MURATA, T. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, New York, v.77, n.4, p.541-580, 1989.

REED, J. P. R. *Desenvolvendo aplicativos com Visual Basic e UML*. São Paulo: Makron Books, 2000.