CAPÍTULO

7

MODELAGEM DE REQUISITOS: FLUXO, COMPORTAMENTO, PADRÕES E APLICAÇÕES BASEADAS NA WEB (WEBAPP)

CONCEITOS-CHAVE

Após nossa discussão de casos de uso, modelagem de dados e modelos baseados em classes no Capítulo 6, é razoável perguntar: "Não seriam essas representações de modelagem de requisitos suficientes?".

A única resposta razoável é: "Depende". Para alguns tipos de software, o caso de uso poderia ser a única representação de modelagem de requisitos exigida. Para outros, é escolhida uma abordagem orientada a objetos, e modelos baseados em classes poderiam ser desenvolvidos. Porém, em outras situações, requisitos de aplicação complexos poderiam demandar um exame de como os objetos de dados são transformados à medida que fluem por um sistema; como uma aplicação se comporta como consequência de eventos externos; se o conhecimento do domínio existente pode ser adaptado ao problema atual; ou no caso de sistemas e aplicações baseadas na Web (WebApp), como o conteúdo e a funcionalidade combinados poderiam dar a um usuário final a habilidade de navegar com êxito por uma WebApp para atingir essas metas.

PANORAMA

O que é? O modelo de requisitos possui várias dimensões diferentes. No presente capítulo você aprenderá modelos orientados a fluxos, modelos comportamentais e considerações sobre análise de requisitos especiais que entram em cena quando são desenvolvidas aplicações baseadas na Web (WebApps). Cada uma dessas representações de modelagem complementa os casos de uso, os modelos de dados e os modelos baseados em classes discutidos no Capítulo 6.

Quem realiza? Um engenheiro de software (às vezes denominado "analista") constrói o modelo usando os requisitos extraídos de vários interessados.

Por que é importante? Sua visão sobre os requisitos do software aumenta em proporção direta ao número de diferentes dimensões da modelagem de requisitos. Embora talvez você não tenha tempo, recursos ou inclinação para desenvolver cada representação sugerida neste capítulo e no Capítulo 6, deve reconhecer que cada abordagem de modelagem lhe dá uma forma diferente de visualizar o problema. Como consequência, você (e outros interessados) estarão mais bem preparados para avaliar se o que deve ser cumprido foi ou não especificado de maneira apropriada.

Quais são as etapas envolvidas? A modelagem orientada a fluxos dá uma indicação de como os objetos de dados são transformados pelas funções de processamento. A modelagem comportamental representa os estados do sistema e suas classes e o impacto dos eventos sobre esses estados. A modelagem baseada em padrões faz uso do conhecimento do domínio existente para facilitar a análise de requisitos. Os modelos de requisitos para WebApp são especialmente adaptados para a representação das necessidades relacionadas ao conteúdo, interação, função e configuração.

Qual é o artefato? Uma ampla gama de formas textuais e esquemáticas pode ser escolhida para o modelo de requisitos. Cada uma dessas representações dá uma visão de um ou mais dos elementos do modelo.

Como garantir que o trabalho foi realizado corretamente? Os artefatos do modelamento de requisitos devem ser revisados em termos de correção, completude e consistência. Devem refletir os requisitos de todos os interessados e estabelecer uma base a partir da qual o projeto pode ser conduzido.

7.1 Estratégias de Modelagem de Requisitos

Uma visão da modelagem de requisitos, denominada *análise estruturada*, considera os dados e os processos que os transformam entidades distintas. Os objetos de dados são modelados de maneira que defina seus atributos e relacionamentos. Processos que manipulam objetos de dados são modelados para mostrar como transformam os dados à medida que objetos de dados fluem através do sistema. A segunda abordagem para a modelagem de análise denominada *análise orientada a objetos*, enfoca a definição de classes e a maneira pela qual colaboram entre si para atender os requisitos do cliente.

Embora o modelo de análise que propomos neste livro combina recursos de ambas as abordagens, as equipes de software em geral optam por uma abordagem e excluem todas as representações da outra. A questão não é qual a melhor, mas sim, qual combinação de representações irá fornecer aos interessados o melhor modelo de requisitos de software e a ligação mais efetiva para o projeto de software.

7.2 Modelagem Orientada a Fluxos

Alguns poderão sugerir que o DFD seja coisa do passado e que não tem mais lugar na prática moderna. Trata-se de uma visão que exclui um modo de representação potencialmente útil no nível de análise. Se ele for útil para comunicar e documentar, use o DFD.

Embora a modelagem orientada a fluxos de dados seja vista como uma técnica ultrapassada por alguns engenheiros de software, continua a ser uma das notações de análise de requisitos mais largamente usadas hoje em dia.[1] Embora o *diagrama de fluxo de dados* (*data flow diagram, DFD*) e diagramas relacionados não sejam uma parte formal da UML, podem ser usados para complementar os diagramas UML e darem uma visão adicional sobre o fluxo e os requisitos do sistema.

O DFD adota uma visão entrada-processo-saída de um sistema. Isto é, os objetos de dados entram no software, são transformados por elementos de processamento e os objetos de dados resultantes saem do software. Os objetos de dados são representados por setas rotuladas e as transformações, por círculos (também chamados bolhas). O DFD é apresentado de uma forma hierárquica: o primeiro modelo de fluxo de dados (algumas vezes denominado DFD nível 0 ou *diagrama de contexto*) representa o sistema como um todo. Diagramas de fluxo de dados subsequentes refinam o diagrama de contexto, fornecendo detalhamento progressivo em cada nível subsequente.

7.2.1 Criação de um modelo de fluxo de dados

"O propósito dos diagramas de fluxo de dados é fornecer uma ponte semântica entre usuários e desenvolvedores de sistemas."

Kenneth Kozar

O diagrama de fluxo de dados permite que desenvolvamos modelos do domínio de informações e domínio funcional. À medida que o DFD é refinado com níveis de detalhe cada vez maiores, realizamos uma decomposição funcional implícita do sistema. Ao mesmo tempo, o refinamento do DFD resulta em um correspondente refinamento dos dados à medida que se avança nos processos que constituem a aplicação.

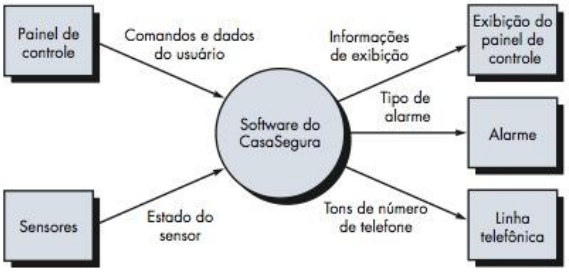
Algumas diretrizes simples podem ajudar muito durante a obtenção do diagrama de fluxo de dados: (1) o diagrama de fluxo de dados nível 0 deve representar o software/sistema como uma única bolha; (2) as entradas e saídas primárias devem ser cuidadosamente indicadas; (3) o refinamento deve começar isolando prováveis processos, objetos de dados e repositórios de dados para ser representados no nível seguinte; (4) todas as setas e bolhas devem ser rotuladas com nomes significativos; (5) deve-se manter a *continuidade do fluxo de informações* de nível para nível,[2] e (6) deve-se refinar uma bolha por vez. Há uma tendência natural de se complicar demais o diagrama de fluxo de dados. Isso acontece quando tentamos mostrar muitos detalhes precocemente ou representar aspectos procedurais do software em vez de fluxo de informações.

---

1 A modelagem de fluxo de dados é uma atividade de modelagem fundamental na *análise estruturada*.

2 Ou seja, os objetos de dados que fluem para dentro do sistema ou qualquer transformação em um nível devem ser os mesmos objetos de dados (ou suas partes constituintes) que fluem para dentro da transformação em um nível mais refinado.

FIGURA 7.1
DFD em nível de contexto para a função de segurança do CasaSegura



PONTO-CHAVE
A continuidade do fluxo de informações deve ser mantida à medida que cada nível DFD seja refinado. Isso significa que entrada e saída em um nível devem ser o mesmo que entrada e saída em um nível refinado.

Para ilustrarmos o uso do DFD e da notação relacionada, consideremos mais uma vez a função de segurança do *CasaSegura*. Um DFD nível 0 para a função de segurança é mostrado na Figura 7.1. As *entidades externas* (retângulos) primárias produzem informações para uso do sistema e consomem informações geradas pelo sistema. As setas rotuladas representam objetos de dados ou hierarquias de objetos de dados. Por exemplo, **dados e comandos de usuário** englobam todos os comandos de configuração, todos os comandos de ativação/desativação, todas as variações de interações e todos os dados introduzidos para qualificar ou expandir um comando.

O DFD nível 0 agora tem de ser expandido para um modelo de fluxo de dados nível 1. Mas como prosseguimos? Seguindo uma abordagem sugerida no Capítulo 6, deveríamos aplicar uma "análise sintática" [Abb83] à narrativa de caso de uso que descreve a bolha no nível de contexto. Isolamos todos os substantivos (e locuções substantivas) e verbos (e locuções verbais) em uma narrativa de processamento do *CasaSegura* obtida durante a primeira reunião para levantamento de necessidades. Recapitulando o texto narrativo de processamento com análise sintática apresentado na Seção 6.5.1:

> A função de segurança domiciliar do *CasaSegura permite* que o proprietário do imóvel *configure* o sistema de segurança quando ele é *instalado, monitora* todos os sensores *conectados* ao sistema de segurança e *interage* com o proprietário do imóvel através da Internet, de um PC ou de um painel de controle.
>
> Durante a instalação, o PC do *CasaSegura* é utilizado para *programar* e *configurar* o sistema. São atribuídos um número e um tipo a cada sensor, é programada uma senha-mestra para *armar* e *desarmar* o sistema e são *introduzidos* número(s) de telefone para *discar* quando ocorre um evento de sensor.
>
> Quando um evento de sensor é *reconhecido*, o software *aciona* um alarme sonoro agregado ao sistema. Após um tempo de espera que é *especificado* pelo proprietário do imóvel durante as atividades de configuração do sistema, o software disca um número de telefone de um serviço de monitoramento, *fornece* informações sobre o local, *relatando* a natureza do evento detectado. O número de telefone será *rediscado* a cada 20 segundos até que seja *completada a ligação telefônica*.
>
> O proprietário do imóvel *recebe* informações de segurança através de um painel de controle, do PC ou de um navegador, coletivamente denominados de interface. A interface *mostra* mensagens de aviso bem como informações sobre o estado do sistema no painel de controle, no PC ou na janela do navegador. A interação com o proprietário do imóvel assume a seguinte forma...

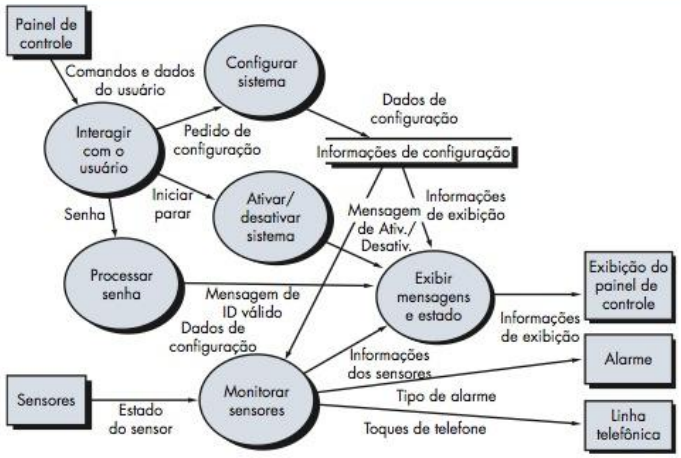
AVISO
A análise sintática não é infalível, mas pode proporcionar um excelente passo inicial, caso esteja havendo dificuldades para definir objetos de dados e as transformações que neles operam.

AVISO
Assegure-se de que a narrativa de processamento que você pretende analisar sintaticamente esteja escrita no mesmo nível de abstração por todo o processo.

Referindo-se à análise sintática, os verbos são processos do *CasaSegura* e podem ser representados como bolhas em um DFD subsequente. Os substantivos podem ser entidades externas (retângulos), objetos de controle ou de dados (setas) ou então repositórios de dados (linhas duplas). Da discussão do Capítulo 6, lembre-se de que os substantivos e verbos podem ser associados entre si (por exemplo, são atribuídos um número e um tipo a cada sensor; dessa forma, **número** e **tipo** são atributos do objeto de dados **sensor**). Consequentemente, realizando uma análise sintática na narrativa de processamento para uma bolha em qualquer nível DFD, podemos gerar muitas informações úteis sobre como prosseguir com o refinamento para o próximo nível. Usando essas informações, é mostrado um DFD nível 1 na Figura 7.2. O processo no nível de contexto mostrado na Figura 7.1 foi expandido em seis processos obtidos do exame da análise sintática. De modo

FIGURA 7.2

DFD nível 1 para a função de segurança do *CasaSegura*



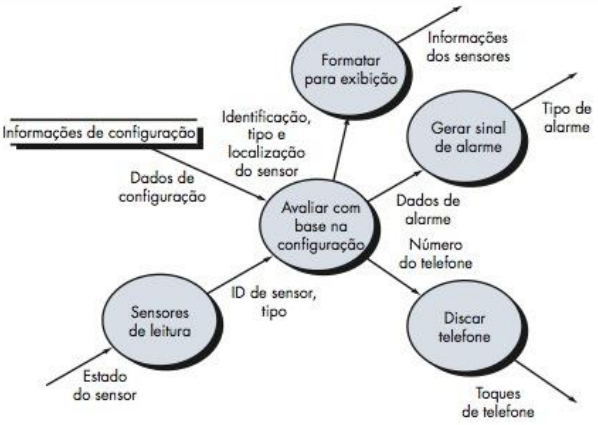
similar, o fluxo de informações entre processos no nível 1 foi extraído da análise sintática. Além disso, a continuidade do fluxo de informações é mantida entre os níveis 0 e 1.

Os processos representados no nível 1 do DFD podem ser mais refinados em níveis mais baixos. Por exemplo, o processo *monitorar sensores* pode ser refinado em um DFD nível 2, conforme mostra a Figura 7.3. Note mais uma vez que a continuidade do fluxo de informações foi mantida entre os níveis.

O refinamento de DFDs prossegue até que cada bolha realize uma simples função. Até que o processo representado pela bolha realize uma função que seria facilmente implementada como componente de um programa. No Capítulo 8, discutiremos um conceito, chamado *coesão*, que pode ser usado para avaliar o foco do processamento de determinada função. Por enquanto, nos esforçaremos para refinar os DFDs até que cada bolha tenha um "único objetivo".

FIGURA 7.3

DFD nível 2 que refina o processo monitorar sensores



7.2.2 Criação de um modelo de fluxo de controle

Para alguns tipos de aplicações, o modelo de dados e o diagrama de fluxo de dados bastam para ter uma visão dos requisitos de software. Entretanto, conforme já citado, um grande grupo de aplicações é "dirigido" por eventos e não por dados, produzem informações de controle em vez de relatórios ou telas e processam informações com grande preocupação com o tempo e desempenho. Tais aplicações requerem o uso da *modelagem de fluxo de controle*, além da modelagem de fluxo de dados.

Já vimos que um item de evento ou controle é implementado como um valor booleano (por exemplo, verdadeiro ou falso, ligado ou desligado, 1 ou 0) ou como uma lista discreta de condições (por exemplo, vazio, bloqueado, cheio). Para escolher eventos possíveis candidatos a eventos, sugerem-se as seguintes diretrizes:

Como escolher eventos potenciais para um diagrama de fluxo de controle, um diagrama de estados ou uma CSPEC?

- Listar todos os sensores "lidos" pelo software.
- Listar todas as condições de interrupção.
- Listar todas as "chaves" acionadas por um operador.
- Listar todas as condições de dados.
- Recapitular a análise sintática de substantivos/verbos aplicada à narrativa de processamento, revisar todos os "itens de controle" como possíveis entradas/saídas de especificação de controle.
- Descrever o comportamento de um sistema por meio da identificação de seus estados, de como cada estado é atingido e definir as transições entre os estados.
- Concentrar-se em possíveis omissões — um erro muito comum na especificação de controles; por exemplo, perguntar: "Existe alguma outra maneira de chegarmos ou sairmos desse estado?".

Entre os muitos itens de eventos e de controle que fazem parte do software *CasaSegura* temos **evento de sensor** (isto é, um sensor foi acionado), **flag piscante** (um sinal para fazer com que a exibição fique piscando) e **iniciar/parar chave** (um sinal para ligar ou desligar o sistema).

7.2.3 A especificação de controles

Uma *especificação de controle* (*control specification, CSPEC*) representa o comportamento do sistema (no nível em que foi referido) de duas maneiras diferentes.[3] A CSPEC contém um diagrama de estados que é uma especificação de comportamento sequencial. Ela também pode conter uma tabela de ativação de programas — uma especificação de comportamento combinatória.

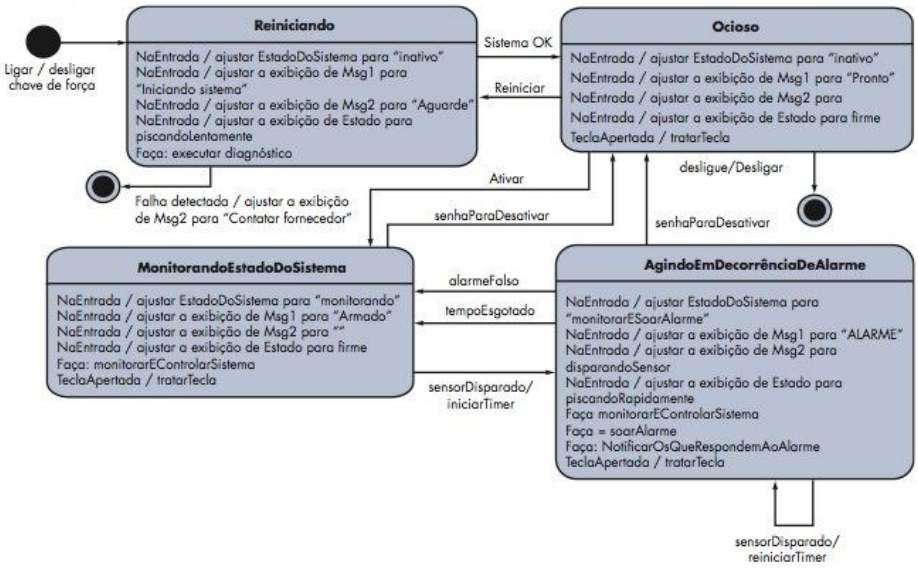
A Figura 7.4 representa um diagrama de estados[4] preliminar para o modelo de fluxo de controle nível 1 para o *CasaSegura*. O diagrama indica como o sistema responde a eventos à medida que passa pelos quatro estados definidos nesse nível. Através da revisão de diagramas de estados, podemos determinar o comportamento do sistema e, mais importante ainda, determinar se há "furos" no comportamento especificado.

Por exemplo, o diagrama de estados (Figura 7.4) indica que as transições do estado **Ocioso** podem ocorrer se o sistema for reinicializado, ativado ou desligado. Se o sistema for ativado (isto é, o sistema de alarme for ligado), a transição para o estado **MonitorandoEstadoDoSistema** ocorre, as mensagens de tela são alteradas como mostrado e o processo *monitorarEControlarSistema* é chamado. Ocorrem duas transições do estado **MonitorandoEstadoDoSistema** — (1) quando o sistema é desativado, ocorre uma transição de volta para o estado **Ocioso**; (2)

3 Outras notações referentes à modelagem comportamental são apresentadas na Seção 7.3.

4 A notação de diagramas de estados aqui usada está em conformidade com a notação da UML. Um "diagrama de transição de estados" se encontra disponível na análise estruturada, porém, o formato UML é superior em termos de representação e conteúdo de informações.

FIGURA 7.4
Diagrama de estados para a função de segurança do *CasaSegura*



quando um sensor é ativado, uma transição para o estado **AgindoEmDecorrenciaDeAlarme**. Todas as transições e o conteúdo de todos os estados são considerados durante a revisão.

Um modo ligeiramente distinto de representação comportamental é a tabela de ativação de processos (*process activation table*, PAT). A PAT representa informações contidas em diagramas de estados no contexto de processos, e não de estados. A tabela indica quais processos (bolhas) no modelo de fluxos serão chamados quando um evento ocorrer. A PAT pode ser usada como guia para um projetista que tem de construir um executável que controla os processos representados nesse nível. Uma PAT para o modelo de fluxos nível 1 do software *CasaSegura* é mostrada na Figura 7.5.

A CSPEC descreve o comportamento do sistema, mas não nos dá nenhuma informação sobre o funcionamento interno dos processos ativados como resultado desse comportamento. A notação de modelagem que fornece essas informações é discutida na Seção 7.2.4.

7.2.4 A especificação de processos

A *especificação de processos (process specification, PSPEC)* é usada para descrever todos os processos do modelo de fluxo que aparecem no nível final de refinamento. O conteúdo da especificação de processos pode incluir texto narrativo, uma descrição[5] em PDL (linguagem de projeto de programas) do algoritmo de processos, equações matemáticas, tabelas ou diagramas

5 A linguagem de projeto de programas (PDL) mistura sintaxe de linguagem de programação com texto narrativo para oferecer um projeto procedural detalhado. A PDL é discutida rapidamente no Capítulo 10.

FIGURA 7.5

Tabela de ativação de processos para a função de segurança do *CasaSegura*

eventos de entrada						
evento de sensor	0	0	0	0	1	0
flag piscante	0	0	1	1	0	0
iniciar/parar chave	0	1	0	0	0	0
exibir status de ação completa	0	0	0	1	0	0
em andamento	0	0	1	0	0	0
tempo esgotado	0	0	0	0	0	1
saída						
sinal de alarme	0	0	0	0	1	0
ativação de processos						
monitorar e controlar sistema	0	1	0	0	1	1
ativar/desativar sistema	0	1	0	0	0	0
exibir mensagens e estado	1	0	1	1	1	1
interagir com o usuário	1	0	0	1	0	1

CASASEGURA

Modelagem de fluxo de dados

Cena: Sala do Jamie, após a última reunião para levantamento de requisitos.

Atores: Jamie, Vinod e Ed — todos membros da equipe de engenharia de software do *CasaSegura*.

Conversa:

(Jamie esboçou os modelos apresentados nas Figuras 7.1 a 7.5 e está mostrando-os para Ed e Vinod.)

Jamie: Tive um curso de engenharia de software na faculdade e eles nos ensinaram essa matéria. O professor disse que é um tanto antigo, mas, quer saber, ajuda-me a esclarecer certos pontos.

Ed: Isso é excelente. Mas não estou vendo nenhuma classe ou objeto aí.

Jamie: Não... Trata-se apenas de um modelo de fluxos com algo sobre seu comportamento no meio dele.

Vinod: Então estes DFDs representam uma visão E-P-S do software, não é mesmo?

Ed: E-P-S?

Vinod: Entrada-processo-saída. Os DFDs são, na verdade, bastante intuitivos... Se você examiná-los por um momento, eles mostram como os objetos de dados fluem através do sistema e são transformados à medida que fluem.

Ed: Parece que poderíamos converter cada bolha em um componente executável... Pelo menos no nível mais baixo do DFD.

Jamie: É interessante, podemos fazer isso. De fato, existe uma maneira de transformarmos os DFDs em uma arquitetura de projeto.

Ed: Verdade?

Jamie: Isso mesmo, mas primeiro temos que desenvolver um modelo de requisitos completo e, no momento, ele não está.

Vinod: Bem, esta é a primeira etapa, mas também teremos que tratar os elementos baseados em classes, bem como os aspectos comportamentais, embora o diagrama de estados e a PAT já façam uma parte disso.

Ed: Temos muito trabalho pela frente e não muito tempo para fazê-lo. (Doug — o gerente de engenharia de software — entra na sala.)

Doug: Portanto, os próximos dias serão gastos no desenvolvimento do modelo de requisitos, não é mesmo?

Jamie (expressando orgulho): Já começamos.

Doug: Muito bem, temos muito trabalho pela frente e não muito tempo para fazê-lo.

(Os três engenheiros de software se entreolham e sorriem.)

PONTO-CHAVE

PSPEC é uma "miniespecificação" para cada transformação no nível mais baixo de refinamento de um DFD.

de atividades UML. Ao juntarmos uma PSPEC a cada bolha no modelo de fluxos, podemos criar uma "miniespecificação" que sirva como guia para o projeto do componente de software que irá implementar a bolha.

Para ilustrarmos o uso da PSPEC, consideremos a transformação processar senha representada no modelo de fluxos do *CasaSegura* (Figura 7.2). A PSPEC para essa função poderia adquirir a seguinte forma:

PSPEC: processar senha (no painel de controle). A transformação *processar senha* realiza a validação de senhas no painel de controle para a função de segurança do *CasaSegura*. *Processar senha* re-

cebe uma senha de quatro dígitos da função *interagir com o usuário*. Primeiro, a senha é comparada com a senha-mestra armazenada no sistema. Se a senha-mestra coincidir, <valid id message = true> é passada para a função *mostrar mensagem e status*. Se a senha-mestra não coincidir, os quatro dígitos são comparados com uma tabela de senhas secundárias (essas poderiam ser atribuídas a convidados e/ou empregados que precisam ter acesso à casa quando o proprietário não está presente). Se a senha coincidir com uma entrada da tabela, <valid id message = true> é passado para a *função mostrar mensagem e status*. Caso não coincida, <valid id message = false> é passado para a função *mostrar mensagem e status*.

Caso sejam necessários mais detalhes sobre o algoritmo nesse estágio, uma representação PDL também poderia ser incluída como parte da PSPEC. Entretanto, muitos acreditam que a versão PDL deva ser postergada até que se inicie o projeto de componentes.

FERRAMENTAS DO SOFTWARE

Análise estruturada

Objetivo: As ferramentas de análise estruturada permitem a um engenheiro de software criar modelos de dados, modelos de fluxos e modelos comportamentais para possibilitar a verificação de consistência e continuidade, bem como fácil edição e extensão. Os modelos criados empregando-se tais ferramentas fornecem ao engenheiro de software uma visão da representação de análise e podem ajudar a eliminar erros antes que se propaguem pelo projeto ou, pior ainda, na própria implementação.

Mecânica: Ferramentas nessa categoria usam um "dicionário de dados" como banco de dados central para a descrição de todos os objetos de dados. Uma vez que as entradas no dicionário tenham sido definidas, os diagramas entidade-relacionamento podem ser criados e as hierarquias de objetos podem ser desenvolvidas. Os recursos dos diagramas de fluxo de dados permitem criar facilmente esse modelo gráfico, bem como fornecem recursos para a criação de PSPECs e CSPECs.

As ferramentas de análise também possibilitam que o engenheiro de software crie modelos comportamentais usando o diagrama de estados como notação efetiva.

Ferramentas representativas:[6]

MacA&D, WinA&D, desenvolvidas pela Excel software (**www.excelsoftware.com**), oferece um conjunto de ferramentas simples e baratas para análise e projeto tanto para máquinas Macs quanto Windows.

MetaCASE Workbench, desenvolvida pela MetaCase Consulting (**www.metacase.com**), é uma metaferramenta usada para definir um método de análise ou projeto (inclusive análise estruturada), bem como seus conceitos, regras, notações e geradores.

System Architect, desenvolvida pela Popkin Software (**www.popkin.com**) oferece uma ampla gama de ferramentas de análise e projeto, inclusive ferramentas para modelagem de dados e análise estruturada.

7.3 CRIAÇÃO DE UM MODELO COMPORTAMENTAL

Como modelar a reação do software a algum evento externo?

A notação de modelagem discutida aqui até então representa elementos estáticos do modelo de requisitos. É chegado o momento de fazermos uma transição para o comportamento dinâmico do sistema ou produto. Para tanto, precisamos representar o comportamento do sistema em função do tempo e eventos específicos.

O *modelo comportamental* indica como o software irá responder a estímulos ou eventos externos. Para criá-lo, devemos executar as seguintes etapas:

1. Avaliar todos os casos de uso para entender completamente a sequência de interação dentro do sistema.
2. Identificar eventos que dirigem a sequência de interações e compreender como esses eventos se relacionam com objetos específicos.
3. Criar uma sequência para cada caso de uso.
4. Construir um diagrama de estados para o sistema.
5. Revisar o modelo comportamental para verificar precisão e consistência.

Cada uma dessas etapas é discutida nas seções a seguir.

6 As ferramentas aqui apresentadas não significam um aval, mas sim uma amostra dessa categoria. Na maioria dos casos, seus nomes são marcas registradas pelos respectivos desenvolvedores.

7.3.1 Identificação de eventos com o caso de uso

No Capítulo 6 vimos que o caso de uso representa uma sequência de atividades que envolve atores e o sistema. Em geral, um evento ocorre toda vez que o sistema e um ator trocam informações. Na Seção 7.2.3, indicamos que um evento *não* é a informação que foi trocada, mas sim o fato de que as informações foram trocadas.

Um caso de uso é examinado para encontrar pontos de troca de informação. Para ilustrarmos, reconsideremos o caso de uso de uma pequena parte da função de segurança do *CasaSegura*.

> O proprietário usa o teclado numérico para digitar uma senha de quatro dígitos. A senha é comparada com a senha válida armazenada no sistema. Se a senha for incorreta, o painel de controle emitirá um bipe e se reiniciará para receber novas entradas. Se a senha for correta, o painel de controle aguarda as próximas ações.

Os trechos sublinhados do cenário do caso de uso indicam eventos. Deve se identificar um ator para cada evento; as informações trocadas devem ser indicadas e quaisquer condições ou restrições devem ser enumeradas.

Como um exemplo típico de evento, consideremos o trecho sublinhado do caso de uso "proprietário usa o teclado numérico para digitar uma senha de quatro dígitos". No contexto do modelo de requisitos, o objeto, **Proprietário**,[7] transmite um evento para o objeto **PainelDeControle**. O evento poderia ser chamado *entrada de senha*. As informações transferidas são os quatro dígitos que constituem a senha, mas essa não é parte essencial do modelo comportamental. É importante notar que alguns eventos têm um impacto explícito no fluxo de controle do caso de uso, ao passo que outros não têm impacto direto no fluxo de controle. Por exemplo, o evento *entrada de senha* não muda explicitamente o fluxo de controle do caso de uso, mas os resultados do evento *entrada de senha* (derivado da interação "senha é comparada com a senha válida armazenada no sistema") terá um impacto explícito no fluxo de controle e informações do software *CasaSegura*.

Uma vez que todos os eventos tenham sido identificados, são alocados aos objetos envolvidos. Os objetos podem ser responsáveis pela geração de eventos (por exemplo, **Proprietário** gera um evento *entrada de senha*) ou reconhece eventos que ocorreram em algum outro ponto (por exemplo, **PainelDeControle** reconhece o resultado binário do evento *comparação de senha*).

7.3.2 Representações de estados

PONTO-CHAVE

O sistema possui estados que representam comportamento específico externamente observável; uma classe possui estados que representam seu comportamento à medida que o sistema executa suas funções.

No contexto da modelagem comportamental, devem ser consideradas duas caracterizações de estados distintas: (1) o estado de cada classe à medida que o sistema executa sua função e (2) o estado do sistema como observado de fora à medida que o sistema executa sua função.[8]

O estado de uma classe pode assumir tanto características passivas quanto ativas [Cha93]. *Estado passivo* é o estado atual de todos os atributos de um objeto. Por exemplo, o estado passivo da classe **Jogador** (na aplicação de videogame discutida no Capítulo 6) incluiria atributos referentes à **posição** e **orientação** atual do **Jogador,** bem como outros recursos do **Jogador** relevantes ao jogo (por exemplo, um atributo que indique **permanência de desejos mágicos**). O *estado ativo* de um objeto indica o estado atual do objeto à medida que passa por uma transformação ou processamento contínuo. A classe **Jogador** poderia ter os seguintes estados ativos: *em movimento, em repouso, contundido, recuperando-se no departamento médico; preso, perdido* e assim por diante. Deve acontecer um evento (algumas vezes denominado *gatilho*). Para forçar um objeto a fazer uma transição de um estado ativo para outro.

---

7 Neste exemplo, partimos do pressuposto de que cada usuário (proprietário do imóvel) que interage com o *CasaSegura* possui uma senha de identificação e é, consequentemente, um objeto legítimo.

8 Os diagramas de estados apresentados no Capítulo 6 e na Seção 7.3.2 representam o estado do sistema. Nossa discussão nessa seção irá se concentrar no estado de cada classe do modelo de análise.

Nos próximos parágrafos serão discutidas duas representações comportamentais distintas. A primeira indica como determinada classe muda de estado baseada em eventos externos e a segunda mostra o comportamento do software em função do tempo.

Diagramas de estados para classes de análise. O componente de um modelo comportamental é um diagrama de estados[9] UML que representa estados ativos para cada uma das classes e para os eventos (*gatilhos*) que provocam mudanças entre esses estados ativos. A Figura 7.6 ilustra um diagrama de estados para o objeto **PainelDeControle** na função de segurança *CasaSegura*.

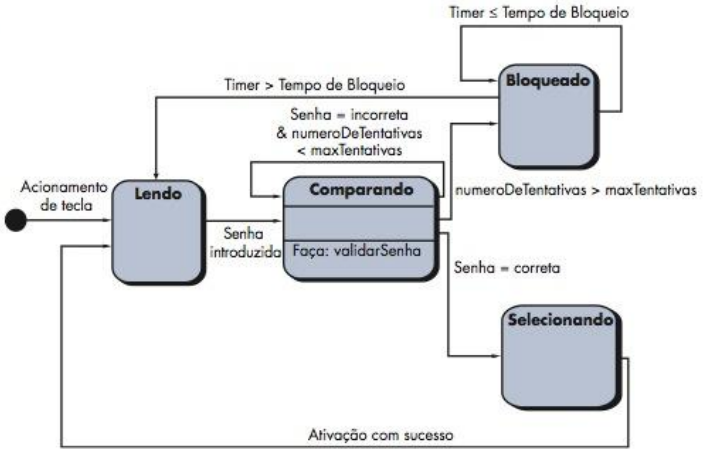
Cada seta da Figura 7.6 representa a transição de um estado ativo de um objeto para outro. As identificações em cada seta representam o evento que dispara a transição. Embora o modelo de estados ativos dê uma visão proveitosa sobre a "história de vida" de um objeto, é possível especificar informações adicionais para uma maior profundidade no entendimento do comportamento de um objeto. Além de especificarmos o evento que faz com que a transição ocorra, podemos especificar um guarda e uma ação [Cha93]. *Guarda* é uma condição booleana que deve ser satisfeita de modo que a transição ocorra. Por exemplo, o guarda para a transição do estado "lendo" para o estado "comparando" na Figura 7.6 pode ser determinado examinando-se o caso de uso:

if (entrada de senha = 4 dígitos) então compare com a senha armazenada

Em geral, o guarda para uma transição normalmente depende do valor de um ou mais atributos de um objeto. Em outras palavras, o guarda depende do estado passivo do objeto.

Uma *ação* ocorre concorrentemente com a transição de estado ou como consequência dele e geralmente envolve uma ou mais operações (responsabilidades) do objeto. Por exemplo, a ação ligada ao evento *entrada de senha* (Figura 7.6) é uma operação chamada *validarSenha()* que acessa um objeto **senha** e realiza uma comparação dígito por dígito para validar a senha introduzida.

FIGURA 7.6
Diagrama de estados para a classe PainelDeControle



9 Caso não esteja familiarizado com a UML, apresentamos uma breve introdução a essa importante notação de modelagem no Apêndice 1.

Diagramas de sequência. O segundo tipo de representação comportamental, denominado *diagrama de sequência* em UML, indica como os eventos provocam transições de objeto para objeto. Uma vez que os eventos tenham sido identificados pelo exame de um caso de uso, o modelador cria um diagrama de sequência — uma representação de como os eventos provocam o fluxo de um objeto para outro em função do tempo. Em resumo, o diagrama de sequência é uma versão abreviada do caso de uso. Ele representa classes-chave e os eventos que fazem com que o comportamento flua de classe para classe.

PONTO-CHAVE

Diferentemente de um diagrama de estados que representa comportamento sem citar as classes envolvidas, um diagrama de sequência representa comportamento descrevendo como as classes passam de um estado para outro.

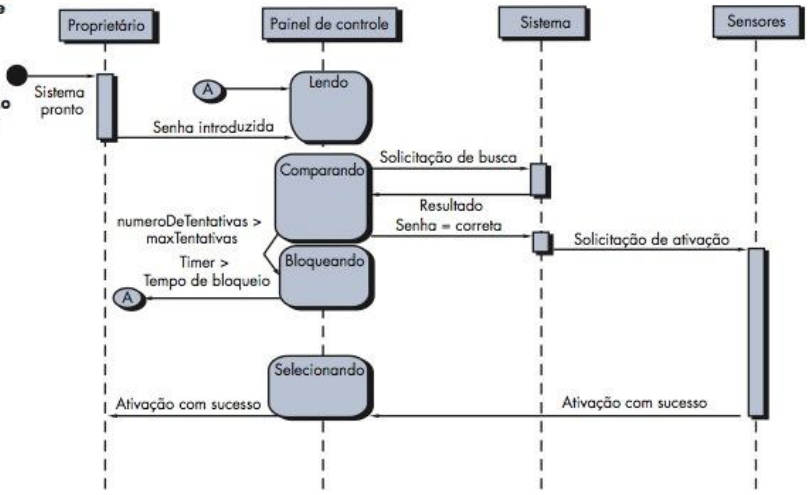
A Figura 7.7 ilustra um diagrama de sequência parcial para a função de segurança do *CasaSegura*. Cada uma das setas representa um evento (derivado de um caso de uso) e indica como o evento canaliza o comportamento entre os objetos do *CasaSegura*. O tempo é medido verticalmente (de cima para baixo), e os retângulos estreitos na vertical representam o tempo gasto no processamento de uma atividade. Os estados poderiam ser mostrados ao longo de uma linha do tempo vertical.

O primeiro evento, *sistema pronto*, é derivado do ambiente externo e canaliza comportamento para o objeto **Proprietário**. O proprietário do imóvel introduz uma senha. Um evento *de solicitação de busca* é passado para **Sistema,** que busca uma senha em um banco de dados simples e retorna um *resultado* (*encontrada* ou *não encontrada*) para **PainelDeControle** (agora no estado *comparando*). Uma senha válida resulta em um evento *senha=correta* para **Sistema,** que ativa **Sensores** com um evento *solicitação de ativação*. Por fim, o controle retorna ao proprietário com o evento *ativação com sucesso*.

Assim que um diagrama de sequência completo tiver sido desenvolvido, todos os eventos que provocam transições entre objetos do sistema podem ser reunidos em um conjunto de eventos de entrada e eventos de saída (de um objeto). Essas informações são úteis na criação de um projeto efetivo para o sistema a ser construído.

FIGURA 7.7

Diagrama de sequência (parcial) para a função de segurança do *CasaSegura*



FERRAMENTAS DO SOFTWARE

Modelagem de análise generalizada em UML

Objetivo: As ferramentas de modelagem de análise fornecem a capacidade de desenvolver modelos baseados em cenários, modelos baseados em classes e modelos comportamentais usando a notação UML.

Mecânica: As ferramentas nesta categoria suportam todo o conjunto de diagramas UML necessários para construir um modelo de análise (as ferramentas também dão suporte à modelagem de projeto). Além da diagramação, elas (1) realizam verificação de consistência e correção para todos os diagramas UML, (2) fornecem links para projeto e geração de código, (3) constroem um banco de dados que permite o gerenciamento e a avaliação de grandes modelos UML necessários para sistemas complexos.

Ferramentas representativas:[10]

As seguintes ferramentas suportam todo o conjunto de diagramas UML necessário para a modelagem de análise:

ArgoUML é uma ferramenta com código-fonte aberto disponível em **argouml.tigris.org**.

Enterprise Architect, desenvolvida pela Sparx Systems (**www.sparxsystems.com.au**).

PowerDesigner, desenvolvida pela Sybase (**www.sybase.com**).

Rational Rose, desenvolvida pela IBM (Rational) (**www01.ibm.com/software/rational/**).

System Architect, desenvolvida pela Popkin Software (**www.popkin.com**).

UML Studio, desenvolvida pela Pragsoft Corporation (**www.pragsoft.com**).

Visio, desenvolvida pela Microsoft (**www.microsoft.com**).

Visual UML, desenvolvida pela Visual Object Modelers (**www.visualuml.com**).

7.4 PADRÕES PARA A MODELAGEM DE REQUISITOS

Padrões de software constituem um mecanismo para capturar conhecimento do domínio acumulado para permitir que seja reaplicado quando um novo problema é encontrado. Em alguns casos, o conhecimento do domínio é aplicado a um novo problema no mesmo domínio de aplicação. Em outros casos, o conhecimento do domínio capturado por um padrão pode ser aplicado por analogia a um domínio de aplicação completamente diferente.

O autor original de um padrão de análise não "cria" o padrão, mas sim, o *descobre* à medida que o fluxo de trabalho de levantamento de requisitos é conduzido. Assim que o padrão tiver sido descoberto, é documentado descrevendo-se "explicitamente o problema geral ao qual o padrão se aplica, a solução recomendada, hipóteses e restrições do emprego do padrão na prática e em geral algumas outras informações sobre o padrão, como a motivação e as forças que orientam o uso do padrão, discussão das vantagens e desvantagens do padrão, bem como referências para alguns exemplos conhecidos do uso desse padrão em aplicações práticas". [Dev01].

No Capítulo 5, introduzimos o conceito de padrões de análise e indicamos que padrões representam uma solução que frequentemente incorpora uma classe, função ou comportamento em um campo de aplicação. O padrão pode ser reutilizado ao se realizar a modelagem de requisitos para uma aplicação em um domínio.[11] Os padrões de análise são armazenados em um repositório de modo que os membros da equipe de software possam usar recursos de pesquisa para encontrá-los e reutilizá-los. Assim que um padrão apropriado tiver sido selecionado, é integrado ao modelo de requisitos por meio de referência ao nome do padrão.

7.4.1 Descoberta de padrões de análise

O modelo de requisitos é formado por uma ampla gama de elementos: baseados em cenários (casos de uso), orientados a dados (o modelo de dados), baseados em classes, orientados a fluxos e comportamentais. Cada um deles examina o problema segundo uma perspectiva e cada uma delas proporciona a descoberta de padrões que poderiam ocorrer em um domínio de aplicação, ou por analogia, em domínios de aplicação diferentes.

10 As ferramentas aqui apresentadas não significam um aval, mas sim uma amostra dessa categoria. Na maioria dos casos, seus nomes são marcas registradas pelos respectivos desenvolvedores.

11 Uma discussão aprofundada do uso de padrões durante o projeto de software é apresentada no Capítulo 12.

O elemento mais básico na descrição de um modelo de requisitos é o caso de uso. No contexto desta discussão, um conjunto coerente de casos de uso poderia servir como base para descobrir um ou mais padrões de análise. *Padrão de análise semântica* (*semantic analysis pattern, SAP*) "descreve um pequeno conjunto de casos de uso coerentes que juntos descrevem uma aplicação genérica básica" [Fer00].

Consideremos o seguinte caso de uso preliminar para um software necessário para controlar e monitorar um sensor de proximidade e câmera de visualização real para um automóvel:

> **Caso de uso:** ***Monitorar deslocamento em marcha a ré***
>
> **Descrição:** Quando o veículo é colocado em *marcha a ré*, o software de controle habilita um alimentador de vídeo por meio de uma câmera de vídeo colocada atrás do painel de comandos. O software de controle sobrepõe uma variedade de linhas de orientação e distâncias na exibição do painel de comandos de modo que o condutor do veículo possa ser orientado à medida que se desloca em marcha a ré. O software de controle também monitora um sensor de proximidade para determinar se um objeto se encontra ou não a 3 m da traseira do veículo. Ele irá frear o veículo automaticamente se o sensor de proximidade indicar um objeto a *x* metros da traseira do veículo, em que *x* é determinado com base na velocidade do veículo.

Esse caso de uso implica uma funcionalidade muito variada que poderia ser refinada e elaborada (em um conjunto de casos de uso coerentes) durante o levantamento de requisitos e a modelagem. Independentemente do nível de elaboração alcançado, os casos de uso sugerem um SAP simples, porém de larga aplicação — o monitoramento e controle de sensores e atuadores baseado em software em um sistema físico. Nesse caso, os "sensores" fornecem informações sobre proximidade e informações de vídeo. O "atuador" é o sistema de frenagem do veículo (acionado caso um objeto esteja muito próximo do veículo). Porém, em um caso mais genérico, se descobre um padrão de larga aplicação.

Diferentes domínios de aplicação para monitorar sensores e controlar atuadores físicos necessitam de software. Um padrão de análise que descreva requisitos genéricos para essa capacidade poderia ser utilizado largamente. O padrão, denominado **Atuador-Sensor**, seria aplicável como parte do modelo de requisitos do *CasaSegura* e é discutido na Seção 7.4.2, a seguir.

7.4.2 Exemplo de padrão de requisitos: atuador-sensor[12]

Um dos requisitos da função de segurança do *CasaSegura* é a habilidade de monitorar sensores de segurança (por exemplo, sensores contra roubos, sensores de fogo, fumaça ou CO, sensores de água).

Extensões para o *CasaSegura baseadas na Internet exigiram a capacidade de* controlar o movimento (por exemplo, deslocamento horizontal e vertical, ampliação/redução) de uma câmera de segurança no interior de uma residência. A implicação — o software *CasaSegura* deve gerenciar vários sensores e "atuadores" (por exemplo, mecanismos de controle de câmera).

Konrad e Cheng [Kon02] sugeriram um padrão de requisitos chamado **Atuador-Sensor** que fornece uma útil orientação para modelar esses requisitos no software *CasaSegura*. Uma versão abreviada do padrão **Atuador-Sensor**, originalmente desenvolvido para aplicações na indústria automobilística, é apresentada a seguir.

Nome do Padrão. **Atuador-Sensor**

Intuito. Especificar vários tipos de sensores e atuadores em um sistema embutido.

Motivo. Os sistemas embarcados normalmente possuem vários tipos de sensores e atuadores. Esses sensores e atuadores estão todos direta ou indiretamente conectados a uma unidade de controle. Embora muitos sensores e atuadores pareçam bem diferentes, seu comportamento é suficientemente similar para estruturá-los em um padrão. O padrão mostra como especificar os sensores e atuadores para um sistema, inclusive atributos e operações. O padrão **Atuador-Sensor** usa um mecanismo *pull* (solicitação explícita de informação)

12 Essa seção foi adaptada de [Kon02] com a permissão dos autores.

para **PassiveSensors (SensoresPassivos)** e um mecanismo *push* (difusão de informação) para os **ActiveSensors (SensoresAtivos).**

Restrições

- Cada sensor passivo deve ter algum método para ler entrada de sensores e atributos que representam o valor do sensor.
- Cada sensor ativo deve ter capacidades para transmitir mensagens de atualização quando seu valor muda.
- Cada sensor ativo deve enviar um *sinal de vida*, uma mensagem de estado emitida em determinado intervalo de tempo, para detectar defeitos.
- Cada atuador deve ter algum método para chamar a resposta apropriada determinada por **ComputingComponent (ComponenteDeCálculo).**
- Cada sensor e atuador deve ter uma função implementada para verificar seu próprio estado de operação.
- Cada sensor e atuador deve ser capaz de testar a validade dos valores recebidos ou enviados e estabelecer seu estado de operação caso os valores se encontrem fora das especificações.

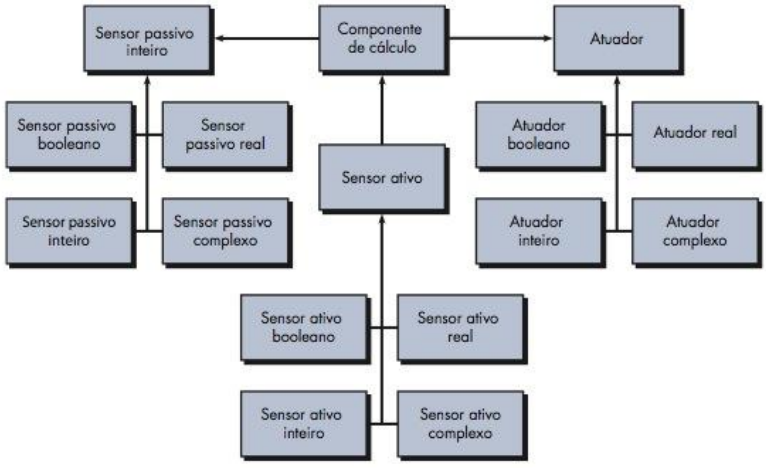
Aplicabilidade. Útil em qualquer sistema em que vários sensores e atuadores estão presentes.

Estrutura. Um diagrama de classes UML para o padrão **Atuador-Sensor** é mostrado na Figura 7.8. **Atuador, PassiveSensor e ActiveSensor** são classes abstratas e grafadas em itálico. Existem quatro tipos diferentes de sensores e atuadores nesse padrão.

As classes **Boolean (Booleana), Integer (Inteira) e Real (Real)** representam os tipos mais comuns de sensores e atuadores. As classes complexas são sensores ou atuadores que usam valores que não podem ser facilmente representados em termos de tipos de dados primitivos, como um dispositivo de radar. Não obstante, tais dispositivos ainda deveriam herdar a interface das classes abstratas já que elas deveriam ter funcionalidades básicas como consulta aos estados de operação.

FIGURA 7.8

Diagrama de sequência UML para o padrão Atuador-Sensor. ***Fonte:*** **adaptado de [Kon02] com permissão**

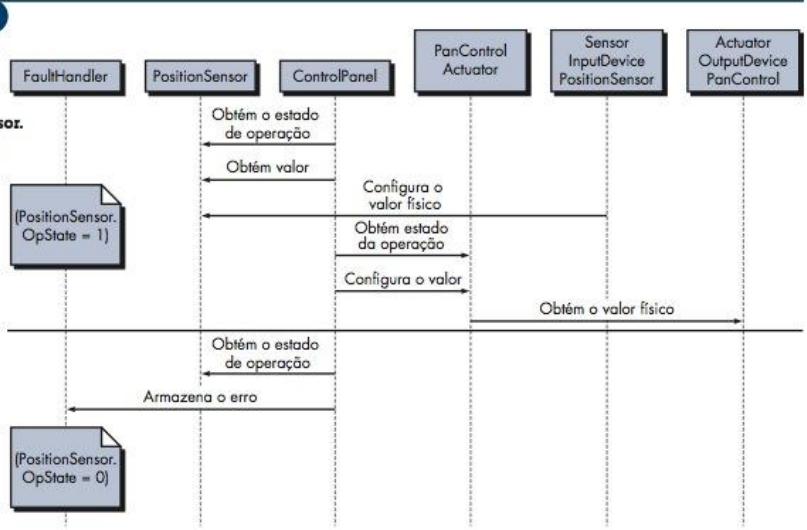


Comportamento. A Figura 7.9 apresenta um diagrama de sequência UML para um exemplo do padrão **Atuador-Sensor** já que ele poderia ser aplicado à função do *CasaSegura* que controla o posicionamento (por exemplo, deslocamentos, ampliação/redução) de uma câmera de segurança. Nesse caso, **ControlPanel (PainelControle)**[13] consulta um sensor (um sensor de posição passivo) e um atuador (controle de deslocamentos) para verificar o estado de operação para fins de diagnóstico antes de ler ou configurar um valor. As mensagens *Set Physical Value* (*AjustarValorFísico*) e *Get Physical Value* (*ObterValorFísico*) não são mensagens entre objetos. Em vez disso, descrevem a interação entre os dispositivos físicos do sistema e seus equivalentes em software. Na parte inferior do diagrama, abaixo da linha horizontal, **PositionSensor** (**SensorPosicional**) informa que o estado de operação é zero. **ComputingComponent** (representada por **ControlPanel**) envia então o código de erro devido a uma falha no sensor de posição para **FaultHandler** (**ControleFalhas**) que decidirá como esse erro afeta o sistema e que medidas são necessárias. Ela obtém os dados dos sensores e calcula a resposta necessária para os atuadores.

Participantes. Essa seção de descrição dos padrões "elenca as classes/objetos incluídos no padrão de requisitos" [Kon02] e descreve as responsabilidades de cada classe/objeto (Figura 7.8). A seguir, temos uma lista resumida:

- **PassiveSensor abstract:** Define uma interface para sensores passivos.
- **PassiveBooleanSensor:** Define sensores passivos booleanos.
- **PassiveIntegerSensor:** Define sensores passivos inteiros.
- **PassiveRealSensor:** Define sensores passivos reais.
- **ActiveSensor abstract:** Define uma interface para sensores ativos.
- **ActiveBooleanSensor:** Define sensores ativos booleanos.
- **ActiveIntegerSensor:** Define sensores ativos inteiros.

FIGURA 7.9
Diagrama de classes UML para o padrão Atuador-Sensor.
***Fonte:* reimpresso de [Kon02] com permissão**



13 O padrão original usa o termo genérico **ComputingComponent.**

- **ActiveRealSensor:** Define sensores ativos reais.
- **Actuator abstract:** Define uma interface para atuadores.
- **BooleanActuator:** Define atuadores booleanos.
- **IntegerActuator:** Define atuadores inteiros.
- **RealActuator:** Define atuadores reais.
- **ComputingComponent:** A parte principal do controlador; ele obtém os dados dos sensores e calcula a resposta necessária para os atuadores.
- **ActiveComplexSensor:** Os sensores ativos complexos possuem a funcionalidade básica da classe abstrata **ActiveSensor,** porém, métodos e atributos adicionais mais elaborados precisam ser especificados.
- **PassiveComplexSensor:** Os sensores passivos complexos possuem a funcionalidade básica da classe abstrata **PassiveSensor**, porém, métodos e atributos adicionais mais elaborados precisam ser especificados.
- **ComplexActuator:** Os atuadores complexos também possuem a funcionalidade básica da classe abstrata **Actuator**, porém, métodos e atributos adicionais mais elaborados precisam ser especificados.

Colaborações. Essa seção descreve como os objetos e as classes interagem entre si e como cada uma delas cumpre suas obrigações.

- Quando **ComputingComponent** precisar atualizar o valor de um **PassiveSensor,** ela consulta os sensores, solicitando o valor por meio do envio de uma mensagem apropriada.
- **ActiveSensors** não são consultados. Eles iniciam a transmissão de valores de sensor para a unidade de cálculo, usando o método apropriado para configurar o valor no **ComputingComponent.** Eles enviam um sinal de vida pelo menos uma vez durante um intervalo de tempo especificado para atualizar seus registros de horas com o horário do relógio do sistema.
- Quando **ComputingComponent** precisar configurar o valor de um atuador, ela envia o valor para o atuador.
- **ComputingComponent** pode consultar e configurar o estado de operação dos sensores e atuadores usando os métodos apropriados. Se for constatado que um estado de operação é zero, o erro é enviado a **FaultHandler,** uma classe que contém métodos para tratar mensagens de erro como, por exemplo, acionar um mecanismo de recuperação mais elaborado ou um dispositivo de backup. Se a recuperação não for possível, o sistema poderá usar apenas o último valor conhecido do sensor ou o valor default .
- **ActiveSensors** oferece métodos para acrescentar ou remover os endereços ou intervalos de endereços dos componentes que querem receber as mensagens no caso de mudança de um valor.

Consequências

1. As classes de sensores e atuadores possuem uma interface comum.
2. Os atributos de classes podem ser acessados apenas através de mensagens e a classe decide se aceita ou não uma mensagem. Por exemplo, se o valor de um atuador estiver configurado acima de seu valor máximo, a classe do atuador talvez não aceite a mensagem ou então use um valor máximo padrão.
3. A complexidade do sistema é potencialmente reduzida devido à uniformidade das interfaces para atuadores e sensores.

A descrição dos padrões de requisitos também poderia fornecer referências a outros padrões de projeto e de requisitos relacionados.

7.5 MODELAGEM DE REQUISITOS PARA WEBAPPS[14]

Os desenvolvedores Web normalmente são céticos quando lhes é sugerida a ideia de análise de requisitos para WebApps. "Afinal de contas", argumentam eles, "o processo de desenvolvimento para Web deve ser ágil e a análise toma muito tempo. Ela irá nos retardar quando simplesmente precisamos projetar e construir uma WebApp."

O levantamento de requisitos realmente toma tempo, porém resolver o problema errado leva mais tempo ainda. A pergunta para cada desenvolvedor de WebApp é simples — você tem certeza de que entendeu os requisitos do problema? Se a resposta for um inequívoco "sim", poderia ser possível pular a fase de modelagem de requisitos, porém se a resposta for "não", a modelagem de requisitos deve ser realizada.

7.5.1 Que nível de análise é suficiente?

O grau com o qual a modelagem de requisitos para WebApps é enfatizado depende dos seguintes fatores:

- Tamanho e complexidade do incremento de WebApp.
- Número de interessados (a análise pode ajudar a identificar requisitos conflitantes provenientes de várias fontes).
- Tamanho da equipe de desenvolvimento de WebApps.
- Nível em que os membros da equipe de desenvolvimento de WebApps trabalharam juntos antes (a análise pode ajudar a desenvolver um entendimento comum do projeto).
- Nível de êxito da organização é diretamente dependente do êxito das WebApps.

O contrário dos pontos anteriores é que à medida que um projeto se torna menor, que o número de interessados diminui, que a equipe se torne mais coesa e a aplicação seja menos crítica, é razoável aplicar uma abordagem de análise menos rígida.

Embora seja uma boa ideia analisar o problema *antes* de começar o projeto, não é verdade que *toda* análise deve preceder *todo o* projeto. De fato, o projeto de uma parte específica da WebApp exige apenas uma análise daqueles requisitos que afetam apenas essa parte da WebApp. Um exemplo para o *CasaSegura*: poderíamos projetar de forma válida toda a estética do site (layouts, esquemas de cores etc.) sem ter analisado as necessidades funcionais para recursos de comércio eletrônico. Precisaríamos analisar apenas aquela parte do problema relevante ao trabalho de projeto do incremento a ser entregue.

7.5.2 Entrada da modelagem de requisitos

Uma versão ágil do processo de software genérico discutido no Capítulo 2 pode ser aplicada quando WebApps forem criadas. O processo incorpora uma atividade de comunicação que identifica interessados e categorias de usuário, o contexto de negócio, metas de aplicação e de informação definidas, requisitos gerais da WebApp e cenários de uso — as informações se tornam entrada para a modelagem de requisitos. Estas são representadas na forma de descrições de linguagem natural, descrições gerais, esboços e outras representações de informação.

A análise captura essas informações, as estruturam usando um esquema de representação formalmente definido (onde apropriado) e depois produz modelos mais rigorosos como saída. O modelo de requisitos fornece uma indicação detalhada da verdadeira estrutura do problema e dá uma visão da forma da solução.

A função **ACS-EVC** (vigilância através de câmeras) do *CasaSegura* foi introduzida no Capítulo 6. Quando foi introduzida, essa função parecia relativamente clara e foi descrita com certo nível de detalhamento como parte de um caso de uso (Seção 6.2.1). Entretanto, um

14 Essa seção foi adaptada de Pressman e Lowe [Pre08] com permissão.

reexame do caso de uso poderia revelar informações que estão faltando, ambíguas ou não claras. Alguns aspectos de informações faltantes emergiriam naturalmente durante o projeto. Exemplos poderiam incluir o layout específico dos botões de função, seu aspecto estético, o tamanho das visões instantâneas, a colocação de visões das câmeras e planta do imóvel ou até mesmo minúcias como o comprimento máximo e mínimo das senhas. Alguns desses aspectos são decisões de projeto (como o layout dos botões) e outros são requisitos (como o tamanho das senhas) que, fundamentalmente, não influenciam os trabalhos iniciais de projeto.

Porém, algumas informações faltantes poderiam realmente influenciar o próprio projeto como um todo e estarem mais relacionadas a um real entendimento dos requisitos. Por exemplo:

P1: Qual a resolução de vídeo fornecida pelas câmeras do *CasaSegura*?

P2: O que acontece se uma condição de alarme for encontrada enquanto a câmera estiver sendo monitorada?

P3: Como o sistema manipula câmeras que possam ser deslocadas e ampliadas/reduzidas?

P4: Que informações deveriam ser fornecidas juntamente com a visão das câmeras? (Por exemplo, posição? hora/data? último acesso anterior?)

Nenhuma dessas perguntas foram identificadas ou consideradas no desenvolvimento inicial do caso de uso, muito embora as respostas poderiam ter um efeito substancial em diferentes aspectos do projeto.

Consequentemente, é razoável concluir que embora a atividade de comunicação forneça uma boa fundamentação para o entendimento, a análise de requisitos refina esse entendimento através de interpretações adicionais. À medida que a estrutura do problema é delineada como parte do modelo de requisitos, invariavelmente surgem perguntas. São essas perguntas que preenchem as lacunas — ou, em alguns casos, realmente nos ajudam, em primeiro lugar, a encontrar as lacunas.

Em suma, as entradas para o modelo de requisitos serão as informações coletadas durante a atividade de comunicação — qualquer uma, desde um e-mail informal até uma descrição de projeto detalhado contendo cenários de uso e especificações de produto completas.

7.5.3 Saída da modelagem de requisitos

A análise de requisitos fornece um mecanismo disciplinado para representar e avaliar o conteúdo e função de uma WebApp, os modos de interação que os usuários irão encontrar e o ambiente e a infraestrutura em que a WebApp reside.

Cada uma dessas características pode ser representada como um conjunto de modelos que permitem que os requisitos da WebApp sejam analisados de maneira estruturada. Embora os modelos específicos dependam em grande parte da natureza da WebApp, há cinco classes principais de modelos:

- **Modelo de conteúdo** — identifica o espectro completo do conteúdo a ser fornecido pela WebApp. Conteúdo pode ser texto, gráficos, imagens, vídeo e áudio.
- **Modelo de interações** — descreve a maneira através da qual os usuários interagem com a WebApp.
- **Modelo funcional** — define as operações que serão aplicadas ao conteúdo da WebApp e descreve outras funções de processamento independentes do conteúdo, mas necessárias para o usuário final.
- **Modelo de navegação** — define a estratégia geral de navegação para a WebApp.
- **Modelo de configuração** — descreve o ambiente e a infraestrutura na qual a WebApp reside.

Podemos desenvolver cada um desses modelos usando um esquema de representação (em geral chamado "linguagem") que permite que seu intuito e estrutura sejam comunicados e avaliados facilmente entre os membros da equipe de engenharia de Web e outros interessados. Como consequência, várias questões fundamentais (por exemplo, erros, omissões, inconsistências, sugestões para aperfeiçoamento ou modificações, pontos a ser esclarecidos) são identificadas e posteriormente trabalhadas.

7.5.4 Modelo de conteúdo para WebApps

O modelo de conteúdo contém elementos estruturais que dão uma visão importante dos requisitos de conteúdo para uma WebApp. Estes englobam objetos de conteúdo e todas as classes de análise — entidades visíveis aos usuários criadas ou manipuladas à medida que o usuário interage com a WebApp.[15]

O conteúdo pode ser desenvolvido antes da implementação da WebApp, enquanto a WebApp está sendo construída ou bem depois de estar em funcionamento. Em todos os casos, ela é incorporada via referência navegacional na estrutura geral da WebApp. Um *objeto de conteúdo* poderia ser uma descrição textual de um produto, um artigo descrevendo um evento de noticiário, uma fotografia de ação tirada durante um evento esportivo, a resposta de um usuário em um fórum de discussão, uma animação do logotipo de uma empresa, um breve vídeo de apresentação ou um áudio agregado a um conjunto de slides de uma apresentação. Os objetos de conteúdo poderiam ser armazenados como arquivos distintos, incorporados diretamente em páginas Web ou obtidos dinamicamente de um banco de dados. Em outras palavras, um objeto de conteúdo é qualquer informação coesa que deve ser apresentada a um usuário final.

Os objetos de conteúdo podem ser determinados diretamente dos casos de uso, examinando-se a descrição do cenário para referências diretas e indiretas ao conteúdo. Por exemplo, uma WebApp que oferecesse suporte ao *CasaSegura* seria estabelecida em **CasaSeguraGarantida.com**. Um caso de uso, *Comprando Componentes Específicos do CasaSegura,* descreve o cenário necessário para adquirir um componente do *CasaSegura* e contém a seguinte sentença:

> Serei capaz de obter informações descritivas e de preço para cada componente do produto.

O modelo de conteúdo deve ser capaz de descrever o objeto de conteúdo **Componente.** Em muitos casos, uma simples lista dos objetos de conteúdo, com uma breve descrição de cada objeto, é suficiente para definir os requisitos para o conteúdo a ser projetados e implementados. Entretanto, em alguns casos, o modelo de conteúdo pode se beneficiar de uma análise mais rica que ilustre graficamente os relacionamentos entre os objetos de conteúdo e/ou a hierarquia do conteúdo mantido por uma WebApp.

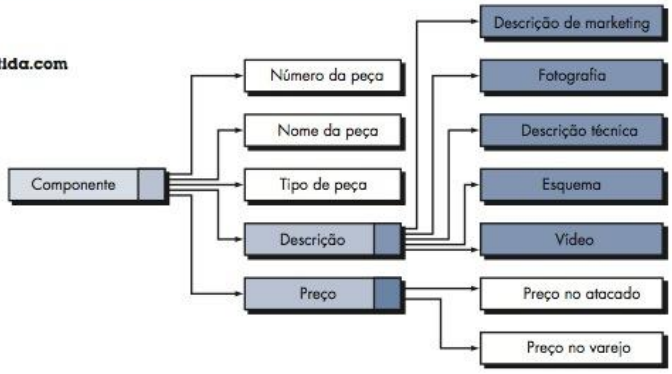
Por exemplo, consideremos a *árvore de dados* [Sri01] criada para um componente de **CasaSeguraGarantida.com** e ilustrada na Figura 7.10. A árvore representa uma hierarquia de informações usadas para descrever um componente. Dados simples ou compostos (um ou mais valores de dados) são representados com retângulos de fundo branco. Os objetos de conteúdo são representados como retângulos chapados. Na figura, a **descrição** é definida por cinco objetos de conteúdo (os retângulos chapados). Em alguns casos, um ou mais desses objetos seriam ainda mais refinados à medida que a árvore de dados fosse se expandindo.

Uma árvore de dados pode ser criada para qualquer conteúdo composto de vários objetos de conteúdo e dados. A árvore de dados é desenvolvida em uma tentativa de definir relações hierárquicas entre os objetos de conteúdo e de fornecer um meio para revisar o conteúdo de modo que omissões e inconsistências sejam reveladas antes de o projeto começar. Além disso, a árvore de dados serve como base para o projeto do conteúdo.

---

15 As classes de análise foram discutidas no Capítulo 6.

FIGURA 7.10
Árvore de dados para um componente de CasaSeguraGarantida.com



7.5.5 Modelo de interações para WebApps

A grande maioria das WebApps possibilita um "diálogo" entre o usuário final e a funcionalidade da aplicação, o conteúdo e o comportamento de uma aplicação. Esse diálogo pode ser descrito por meio de um *modelo de interações* que pode ser composto de um ou mais dos seguintes elementos: (1) casos de uso, (2) diagramas de sequência, (3) diagramas de estados,[16] e/ou (4) protótipos de interfaces do usuário.

Em muitos casos, um conjunto de casos de uso já basta para descrever a interação em um nível de análise (um maior refinamento e detalhes serão introduzidos durante a fase de projeto). Entretanto, quando a sequência de interação for complexa e envolver várias classes de análise ou muitas tarefas, às vezes vale a pena representá-la usando uma forma esquemática mais rigorosa.

O layout da interface do usuário, o conteúdo que ela apresenta, os mecanismos de interação que implementa e a estética geral das conexões de WebApps para usuários têm muito a ver com a satisfação do usuário e com o êxito geral da WebApp. Embora possa se argumentar que a criação de um protótipo de interface do usuário seja uma atividade de projeto, é uma boa ideia realizá-la durante a criação do modelo de análise. O quanto antes uma representação física de uma interface do usuário puder ser revisada, maior a probabilidade de que os usuários finais obterão aquilo que realmente desejam. O projeto de interfaces do usuário é discutido em detalhes no Capítulo 11.

Pelo fato de as ferramentas para construção de WebApps serem abundantes, relativamente baratas e funcionalmente poderosas, é melhor criar o protótipo de interface usando-as. O protótipo deve implementar os principais links de navegação e representar o layout geral da tela em grande parte da mesma forma que será construído. Por exemplo, se o intuito é oferecer cinco funções principais de sistema para o usuário final, o protótipo deve representá-las já que o usuário irá vê-las assim que entrar na WebApp. Serão fornecidos links gráficos? Onde será exibido o menu de navegação? Que outras informações o usuário verá? Perguntas como essas devem ser respondidas pelo protótipo.

7.5.6 Modelo funcional para WebApps

Muitas WebApps oferecem uma ampla gama de funções computacionais e manipuladoras que podem ser associadas diretamente ao conteúdo (seja usando-o, seja produzindo-o) e que

16 Diagramas de sequência e diagramas de estados são modelados usando-se a notação da UML. Os diagramas de estados são descritos na Seção 7.3. Veja o Apêndice 1 para mais detalhes.

frequentemente são sua meta principal de interação com o usuário. Por essa razão, os requisitos funcionais têm de ser analisados e, quando necessário, modelados.

O *modelo funcional* lida com dois elementos de processamento da WebApp, cada um dos quais representando um diferente nível de abstração procedural: (1) funcionalidade observável pelo usuário fornecida pela WebApp aos usuários finais e (2) as operações contidas nas classes de análise que implementam comportamentos associados à classe.

A funcionalidade observável pelos usuários engloba quaisquer funções de processamento iniciadas diretamente pelo usuário. Uma WebApp financeira poderia implementar uma série de funções financeiras (por exemplo, uma calculadora de crédito educativo para o ensino superior ou uma calculadora para planos de aposentadoria). Essas funções poderiam, na verdade, ser implementadas usando-se operações dentro das classes de análise, porém, do ponto de vista do usuário final, a função (mais precisamente, os dados fornecidos pela função) é o resultado visível.

Em um nível de abstração procedural mais baixo, o modelo de requisitos descreve o processamento a ser realizado pelas operações das classes de análise. Essas operações manipulam atributos de classes e estão envolvidas, já que as classes colaboram entre si para cumprir determinado comportamento exigido.

Independentemente do nível de abstração procedural, o diagrama de atividades UML pode ser utilizado para representar detalhes do processamento. No nível de análise, os diagramas de atividades devem ser usados apenas onde a funcionalidade é relativamente complexa. Grande parte da complexidade de muitas WebApps ocorre não na funcionalidade fornecida, mas sim na natureza das informações que podem ser acessadas e nas maneiras pelas quais podem ser manipuladas.

Um exemplo de funcionalidade relativamente complexa do **CasaSeguraGarantida.com** é tratado por um caso de uso intitulado *Obter recomendações para a disposição dos sensores no espaço disponível*. O usuário já desenvolveu um layout para o espaço a ser monitorado e, nesse caso de uso, escolhe esse layout e solicita posições recomendadas para os sensores de acordo com o layout. **CasaSeguraGarantida.com** responde com uma representação gráfica do layout com informações adicionais sobre os pontos recomendados para posicionamento dos sensores. A interação é bastante simples, o conteúdo é ligeiramente mais complexo, porém a funcionalidade subjacente é muito sofisticada. O sistema deve empreender uma análise relativamente complexa do layout do andar para determinar o conjunto ótimo de sensores. Ele tem de examinar as dimensões dos ambientes, a posição das portas e janelas e coordená-las com as capacidades e especificações dos sensores. Nada trivial! Um conjunto de diagramas de atividades pode ser usado para descrever o processamento para esse caso de uso.

O segundo exemplo é o caso de uso *Controlar câmeras*. Neste, a interação é relativamente simples, porém há grande risco de funcionalidade complexa, dado que essa "simples" operação requer comunicação complexa com os dispositivos localizados remotamente e acessíveis via Internet. Uma outra possível complicação está relacionada com a negociação do controle, quando várias pessoas autorizadas tentam monitorar e/ou controlar um único sensor ao mesmo tempo.

A Figura 7.11 ilustra um diagrama de atividades para a operação *assumirControleDaCamera()* que faz parte da classe de análise **Camera** usada no caso de uso *Controlar cameras*. Deve-se notar que duas operações adicionais são chamadas dentro do fluxo procedural: *solicitarBloqueioCamera()*, que tenta bloquear a câmera para esse usuário e *obterUsuarioCameraAtual()*, que recupera o nome do usuário que está controlando a câmera no momento. Os detalhes construtivos indicando como essas operações são chamadas, bem como os detalhes da interface para cada operação, não são considerados até que o projeto da WebApp seja iniciado.

7.5.7 Modelos de configuração para WebApps

Em alguns casos, o modelo de configuração nada mais é que uma lista de atributos no servidor e no cliente. Entretanto, para WebApps mais complexas, uma série de detalhes de configuração

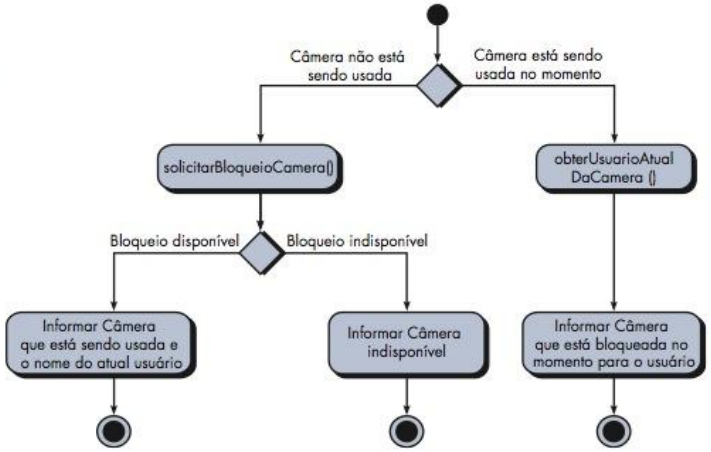


FIGURA 7.11
Diagrama de atividades para a operação *assumirControleDaCamera()*

(por exemplo, distribuição da carga entre vários servidores, arquiteturas de caching, bancos de dados remotos, vários servidores atendendo vários objetos na mesma página Web) poderiam ter um impacto na análise e no projeto. O *diagrama de disponibilização da* UML pode ser utilizado em situações em que complexas arquiteturas de configuração têm de ser consideradas.

Para **CasaSeguraGarantida.com,** a funcionalidade e conteúdo público deveriam ser especificados como acessíveis a todos os principais clientes Web (isto é, aqueles com mais do que 1% de participação no mercado[17]). Ao contrário, pode ser aceitável restringir a funcionalidade de monitoramento e controle mais complexa (que poderia ser acessado somente pelos usuários **Proprietário**) para um conjunto de clientes menor. O modelo de configuração para **CasaSeguraGarantida.com** também especificará a interoperabilidade com aplicações de monitoramento e bancos de dados de produtos existentes.

7.5.8 Modelo de navegação

O modelo de navegação considera a maneira como cada categoria de usuário irá navegar de um elemento da WebApp (por exemplo, objeto de conteúdo) para outro. A mecânica de navegação é definida como parte do projeto. Nesse estágio, devemos nos concentrar nos requisitos gerais de navegação. As seguintes perguntas devem ser consideradas:

- Deveriam certos elementos ser mais fáceis de ser acessados (exigir um número de passos de navegação menor) do que outros? Qual a prioridade para a apresentação?
- Deveriam certos elementos ser enfatizados para forçar os usuários a navegar em sua direção?
- Como os erros de navegação deveriam ser tratados?
- Deveria a navegação para grupos de elementos relacionados ter maior prioridade em relação à navegação para um elemento específico?
- A navegação deveria ser obtida via links, via acesso baseado em pesquisa ou por outros meios?

17 Determinar a participação de mercado para navegadores é notoriamente problemático e varia, dependendo de qual tipo de pesquisa for utilizada. Não obstante, na época em que este livro foi escrito, o Internet Explorer e o Firefox eram os únicos navegadores que ultrapassavam os 30%, enquanto o Mozilla, o Opera e o Safari eram os únicos consistentemente acima de 1%.

- Deveriam certos elementos ser apresentados aos usuários no contexto de ações de navegação prévias?
- Deveria o log de navegação ser mantido para os usuários?
- Deveria existir um menu ou mapa de navegação completo (em vez de um simples link "de retorno" ou ponteiro indicando direção) em qualquer ponto da interação de um usuário?
- Deveria o projeto de navegação ser dirigido pelos comportamentos de usuário mais comumente esperados ou pela importância percebida dos elementos definidos da WebApp?
- Poderia um usuário "armazenar" sua navegação prévia pela WebApp para agilizar seus uso futuro?
- Para qual categoria de usuário a navegação otimizada deveria ser desenvolvida?
- Como os links externos à WebApp deveriam ser tratados? Sobrepondo a janela do navegador existente? Como uma nova janela do navegador? Como um quadro separado?

Essas e muitas outras perguntas devem ser feitas e respondidas como parte da análise de navegação.

Você e outros interessados também devem determinar os requisitos gerais para navegação. Por exemplo, será fornecido um "mapa do site" para dar aos usuários uma visão geral de toda a estrutura da WebApp? Será possível um usuário fazer um "tour guiado" que destacará os elementos mais importantes (objetos de conteúdo e funções) disponíveis? Um usuário será capaz de acessar objetos de conteúdo ou funções baseadas em atributos definidos daqueles elementos (por exemplo, um usuário poderia querer ter acesso a todas as fotografias de determinado prédio ou a todas as funções que possibilitam o cálculo de peso)?

7.6 Resumo

Os modelos orientados a fluxos se concentram no fluxo de objetos de dados à medida que são transformados por funções de processamento. Extraídos da análise estruturada, os modelos orientados a fluxos usam o diagrama de fluxo de dados, uma notação de modelagem que representa como a entrada é transformada em saída à medida que os objetos de dados se deslocam através de um sistema. Cada função de software que transforma dados é descrita por uma narrativa ou especificação de processos. Além do fluxo de dados, esse elemento de modelagem também representa o fluxo de controle — uma representação que ilustra como os eventos afetam o comportamento de um sistema.

A modelagem comportamental representa o comportamento dinâmico. O modelo comportamental usa entrada de elementos baseados em cenários, orientados a fluxos e baseados em classes para representar os estados das classes de análise e do sistema como um todo. Para tanto, são identificados os estados, são definidos os eventos que fazem com que uma classe (ou o sistema) passe por uma transição de um estado para outro e também são identificadas as ações que ocorrem à medida que acontece a transição. Os diagramas de estados e os diagramas de sequência são a notação utilizada para modelagem comportamental.

Os padrões de análise possibilitam a um engenheiro de software usar conhecimento do domínio existente para facilitar a criação de um modelo de requisitos. Um padrão de análise descreve uma função ou recurso de software específico que pode ser descrito por meio de um conjunto de casos de uso coerente. Ele especifica o intuito do padrão, o motivo para seu emprego, restrições que limitam seu uso, sua aplicabilidade em vários domínios de problemas, a estrutura geral do padrão, seu comportamento e colaborações, bem como outras informações complementares.

A modelagem de requisitos para WebApps pode usar a maioria, se não todos, dos elementos de modelagem discutidos neste livro. Entretanto, tais elementos são aplicados em um conjunto de modelos especializados que tratam o conteúdo, interação, função, navegação e a configuração cliente/servidor em que a WebApp reside.

Problemas e Pontos a Ponderar

7.1. Qual a diferença fundamental entre as estratégias de análise estruturada e orientadas a objetos para a análise de requisitos?

7.2. Em um diagrama de fluxo de dados, uma seta representa um fluxo de controle ou algo mais?

7.3. O que é "continuidade de fluxo de informações" e como se aplica à medida que um diagrama de fluxo de dados é refinado?

7.4. Como a análise sintática é usada na criação de um DFD?

7.5. O que é uma especificação de controle?

7.6. Uma PSPEC e um caso de uso são a mesma coisa? Se não forem, explique as diferenças.

7.7. Existem dois tipos de "estados" que os modelos comportamentais são capazes de representar. Quais são eles?

7.8. Como um diagrama de sequência difere de um diagrama de estado? Em que são similares?

7.9. Sugira três padrões de requisitos para um celular moderno e redija uma breve descrição de cada um deles. Poderiam esses padrões ser utilizados para outros dispositivos? Cite um exemplo.

7.10. Escolha um dos padrões que você desenvolveu no Problema 7.9 e faça uma descrição de padrão relativamente completa similar em conteúdo e estilo àquela apresentada na Seção 7.4.2.

7.11. Que nível de modelagem de análise você acredita que seria necessária para **CasaSeguraGarantida.com?** Seria necessário cada um dos tipos de modelos descritos na Seção 7.5.3?

7.12. Qual o objetivo do modelo de interações para uma WebApp?

7.13. Poder-se-ia argumentar que um modelo funcional para WebApp deveria ser postergado até a fase de projeto. Apresente os prós e os contras desse argumento.

7.14. Qual o objetivo de um modelo de configuração?

7.15. Em que o modelo de navegação difere do modelo de interações?

Leituras e Fontes de Informação Complementares

Foram publicados dezenas de livros sobre análise estruturada. Todos cobrem o tema de forma adequada, porém poucos o fazem de maneira realmente excelente. DeMarco e Plauger (*Structured Analysis and System Specification,* Pearson, 1985) é um clássico que ainda se mantém como uma boa introdução à notação básica. Livros como os de Kendall e Kendall (*Systems Analysis and Design*, 5. ed., Prentice-Hall, 2002), Hoffer et al. (*Modern Systems Analysis and Design,* Addison-Wesley, 3. ed., 2001), Davis e Yen (*The Information System Consultant's Handbook: Systems Analysis and Design,* CRC Press, 1998) e Modell (*A Professional's Guide to Systems Analysis,* 2. ed., McGraw-Hill, 1996) são referências proveitosas. O livro de Yourdon (*Modern Structured Analysis,* Yourdon-Press, 1989) sobre o tema ainda se encontra entre os mais completos publicados até hoje.

A modelagem comportamental apresenta uma visão dinâmica importante do comportamento de um sistema. Livros como os de Wagner e seus colegas (*Modeling Software with Finite State Machines: A Practical Approach,* Auerbach, 2006) e Boerger e Staerk (*Abstract State Machines,* Springer, 2003) apresentam uma discussão abrangente sobre diagramas de estado e outras representações comportamentais.

A maioria dos livros escritos que abordam o tema padrões de software enfoca o projeto de software. Entretanto, livros como os de Evans (*Domain-Driven Design,* Addison-Wesley, 2003) e Fowler ([Fow03] e [Fow97]) examinam especificamente os padrões de análise.

Um tratado aprofundado sobre a modelagem de análise para WebApps é apresentado por Pressman e Lowe [Pre08]. Artigos contidos em uma antologia editada por Murugesan e Desphande (*Web Engineering: Managing Diversity and Complexity of Web Application Development,* Springer, 2001) discutem vários aspectos dos requisitos de uma WebApp. Além destes, o *Proceedings of the International Conference on Web Engineering* publicado anualmente trata regularmente de questões referentes à modelagem de requisitos.

Uma ampla gama de fontes de informação sobre modelagem de requisitos se encontra à disposição na Internet. Uma lista atualizada de referências na Web, relevante para a modelagem de análise, pode ser encontrada no site **www.mhhe.com/engcs/compsci/pressman/professional/olc/ser.htm**.