Capítulo 7 – Modelagem Dinâmica

Um sistema de informação realiza ações. O efeito de uma ação pode ser uma alteração em sua base de informação e/ou a comunicação de alguma informação ou comando para um ou mais destinatários. Um evento de requisição de ação (ou simplesmente uma requisição) é uma solicitação para o sistema realizar uma ação. O esquema comportamental de um sistema visa especificar essas ações (OLIVÉ, 2007).

Uma parte importante do modelo comportamental de um sistema é o modelo de casos de uso, o qual fornece uma visão das funcionalidades que o sistema deve prover. O modelo conceitual estrutural define os tipos de entidades (classes) e de relacionamentos (atributos e associações) do domínio do problema que o sistema deve representar para poder prover as funcionalidades descritas no modelo de casos de uso. Durante a realização de um caso de uso, atores geram eventos de requisição de ações para o sistema, solicitando a execução de alguma ação. O sistema realiza ações e ele próprio pode gerar outras requisições de ação. É necessário, pois, modelar essas requisições de ações, as correspondentes ações a serem realizadas pelo sistema e seus efeitos. Este é o propósito da modelagem dinâmica.

Em uma abordagem orientada a objetos, requisições de ação correspondem a mensagens trocadas entre objetos. As ações propriamente ditas e seus efeitos são tratados pelas operações das classes[19]. Assim, a modelagem dinâmica está relacionada com as trocas de mensagens entre objetos e a modelagem das operações das classes.

Os diagramas de classes gerados pela atividade de modelagem conceitual estrutural representam apenas os elementos estáticos de um modelo de análise orientada a objetos. É preciso, ainda, modelar o comportamento dinâmico da aplicação. Para tal, é necessário representar o comportamento do sistema como uma função do tempo e de eventos específicos. Um modelo de dinâmico indica como o sistema irá responder a eventos ou estímulos externos e auxilia o processo de descoberta das operações das classes do sistema.

Para apoiar a modelagem da dinâmica de sistemas, a UML oferece três tipos de diagramas (BOOCH; RUMBAUGH; JACOBSON, 2006):

- *Diagrama de Gráfico de Estados*: mostra uma máquina de estados que consiste dos estados pelos quais objetos de uma particular classe podem passar ao longo de seu ciclo de vida e as transições possíveis entre esses estados, as quais são resultados de eventos que atingem esses objetos. Diagramas de gráfico de estados (ou diagramas de transição de estados) são usados principalmente para modelar o comportamento de uma classe, dando ênfase ao comportamento específico de seus objetos.
- *Diagrama de Interação*: descreve como grupos de objetos colaboram em certo comportamento. Diagramas de interação podem ser de dois tipos: diagramas de

[19] De fato, abordagens distintas podem ser usadas, tal como representar tipos de requisições como classes, ditas classes de evento, e os seus efeitos como operações das correspondentes classes de evento, tal como faz Olivé (2007).

comunicação e diagramas de sequência. Um diagrama de sequência é um diagrama de interação que dá ênfase à ordenação temporal das mensagens trocadas por objetos. Já um diagrama de comunicação dá ênfase à organização estrutural dos objetos que enviam e recebem mensagens. Ambos são usados para ilustrar a visão dinâmica de um sistema.

- *Diagrama de Atividades*: mostra o fluxo de uma atividade para outra em um sistema, incluindo sequências e ramificações de fluxo, subatividades e objetos que realizam e sofrem ações. Diagramas de atividades são usados principalmente para a modelagem das funções de um sistema, dando ênfase ao fluxo de controle na execução de um comportamento.

Diagramas de estados focalizam o comportamento de objetos de uma classe específica. Diagramas de interação e de atividades enfocam o fluxo de controle entre vários objetos e atividades. Enquanto os diagramas de interação dão ênfase ao fluxo de controle de um objeto para o outro, os diagramas de atividade dão ênfase ao fluxo de controle de uma etapa (atividade) para outra. Um diagrama de interação observa os objetos que passam mensagens; um diagrama de atividades focaliza as atividades e suas entradas e saídas, i.e., objetos passados de uma atividade para outra (BOOCH; RUMBAUGH; JACOBSON, 2006).

Este capítulo aborda a modelagem dinâmica, discutindo os principais aspectos dessa importante tarefa, quando realizada segundo o paradigma orientado a objetos. São abordados os diagramas de estados e os diagramas de atividades. Diagramas de interação não são discutidos neste texto. A seção 7.1 discute os diferentes tipos de requisições de ações e como os diagramas dinâmicos da UML podem ser usados para modelá-los. A seções 7.2, 7.3 e 7.4 tratam, respectivamente, dos diagramas de transição de estados, diagramas de sequência e diagramas de atividades. A seção 7.4 aborda a especificação de operações. Finalmente, a seção 7.5 explora as relações existentes entre os vários modelos elaborados durante a análise de requisitos, as quais devem ser atentamente avaliadas durante a atividade de verificação de requisitos.

7.1 – Tipos de Requisições de Ação

Um sistema de informação mantém uma representação do estado do domínio em sua base de informações. Esse estado de coisas do domínio em um dado ponto no tempo corresponde ao conjunto de instâncias dos tipos de entidades (classes) e de relacionamentos (associações) relevantes que existem no domínio naquele momento. O esquema estrutural se preocupa com essa perspectiva. Entretanto, o sistema também realiza ações. O efeito de uma ação pode ser uma alteração em sua base de informação e/ou a comunicação de alguma informação ou comando para um ou mais destinatários. Um evento de requisição de ação (ou simplesmente uma requisição) é uma solicitação para que o sistema realize uma ação. Na análise de requisitos, assume-se que a tecnologia é perfeita e, por conseguinte, que o sistema executa as ações requisitadas instantaneamente (OLIVÉ, 2007).

Dependendo de como são iniciadas, requisições podem ser explícitas, temporais ou geradas. Uma *requisição explícita* é iniciada explicitamente por um ator (*requisição externa*) ou por uma outra ação (*requisição induzida*), como parte de seu efeito. Uma *requisição temporal* é iniciada pela passagem do tempo, ocorrendo independentemente do sistema. Por fim, uma *requisição gerada* é iniciada quando uma condição de geração da requisição é satisfeita. O sistema detecta que a condição foi satisfeita e gera a correspondente requisição.

Por exemplo, em um sistema de controle de estoque, uma requisição de compra pode ser gerada quando a quantidade mínima de um produto for atingida (OLIVÉ, 2007).

A maioria das requisições é externa. Dois importantes tipos de requisições externas são notificações de eventos de domínio e consultas. Uma *consulta* é uma requisição externa que provê alguma informação para o ator que iniciou a requisição. Consultas não alteram a base de informações do sistema. Uma *notificação de evento de domínio* é uma requisição externa cujo efeito é uma mudança na base de informações do sistema, correspondendo a um evento de domínio. Nem todas as mudanças na base de informações de um sistema são admissíveis. Os fatos nessa base mudam ao longo do tempo, mas não de maneira arbitrária. Os eventos de domínio definem, exatamente, as mudanças admissíveis. Por meio de notificações de eventos de domínio, atores dizem para o sistema que um evento de domínio ocorreu (OLIVÉ, 2007). Por exemplo, quando ocorre no mundo real o evento de reserva de um carro em uma locadora de automóveis, um usuário, ao realizar o caso de uso correspondente (p.ex., Reservar Carro), está notificando o sistema que esse evento ocorreu, o que inicia uma sequência de ações (os passos do caso de uso), por meio da qual o sistema sabe que o evento ocorreu no domínio.

Um evento de domínio corresponde a um conjunto não vazio de eventos estruturais, percebido ou considerado como uma alteração única no domínio. Um evento estrutural, por sua vez, é uma ação elementar que insere ou remove um fato na base de informações do sistema. Há quatro tipos básicos de eventos estruturais: inserção de entidade, remoção de entidade, inserção de relacionamento e remoção de relacionamento. Esses eventos são ditos estruturais, porque eles são completamente determinados pelo esquema conceitual estrutural e não são explicitamente mostrados no esquema comportamental (OLIVÉ, 2007).

No desenvolvimento orientado a objetos, os eventos estruturais correspondem a operações básicas das classes. Assim, toda classe tem, implicitamente, operações para: criar objetos da classe (evento estrutural de inserção de entidade), dita operação construtora da classe; eliminar objetos (evento estrutural de remoção de entidade), dita operação destruidora da classe; estabelecer ligações e atribuir valores para atributos (eventos estruturais de inserção de relacionamentos); e remover ligações ou excluir valores de atributos (eventos estruturais de remoção de relacionamentos). Essas operações são consideradas básicas e não precisam ser mostradas explicitamente no modelo conceitual.

Seja o exemplo de um sistema de controle de produtos, cujo modelo estrutural é parcialmente apresentado na Figura 7.1. Nesse exemplo, quando a companhia começa a trabalhar com um novo produto (p.ex., *Prod1*), o estado do domínio se altera, caracterizando um evento de domínio. Esse evento de domínio corresponde a cinco eventos estruturais, a saber: (1) a criação do objeto *Prod1*, (2) a atribuição de um valor para o atributo *codigo*, (3) a atribuição de um valor para o atributo *valor*, (4) a atribuição de um valor para o atributo *quantidadeEstoque* e (5) o estabelecimento da associação *fornece* com seu fornecedor. Esse novo produto é considerado um evento de domínio, porque o conjunto de cinco eventos estruturais que o compõe é visto como um evento único. É muito mais fácil para um usuário dizer ao sistema que o evento de domínio ocorreu do que dizer explicitamente cada um dos eventos estruturais.

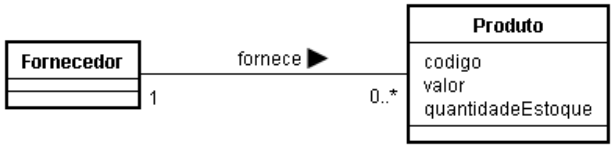

Figura 7.1 – Fragmento do Modelo Estrutural de um Sistema de Controle de Produtos.

Cada evento de domínio possui um conjunto de eventos estruturais, chamado de efeito do evento. A correspondência entre eventos e seus efeitos é dada por uma expressão de mapeamento. O efeito do evento pode ser definido segundo duas abordagens: a abordagem de pós-condição e a abordagem procedimental. Na abordagem de pós-condição, o efeito de um evento é definido por uma condição da base de informações do sistema que deve ser satisfeita após a aplicação do efeito do evento. Na abordagem procedimental, o efeito de um evento é definido por um procedimento, indicando os eventos estruturais que compõem o evento de domínio (OLIVÉ, 2007). Neste texto, enfocamos apenas a abordagem procedimental.

Na abordagem procedimental, quando o paradigma orientado a objetos é adotado, o efeito de um evento é definido como uma operação de uma classe. O corpo dessa operação deve ser tal que sua execução produza o conjunto de eventos estruturais que compõe o evento de domínio. A execução dessa operação vai deixar a base de informações em um novo estado, o qual deve satisfazer a todas as restrições estáticas (definidas no modelo estrutural). Assim, durante a definição da operação correspondente ao efeito de um evento de domínio, devem-se levar em conta as restrições capturadas no modelo estrutural e garantir que o novo estado da base de informações do sistema vai satisfazer a todas elas. Em outras palavras, os eventos de domínio do esquema comportamental devem estar consistentes com o esquema estrutural (OLIVÉ, 2007).

A ideia de efeito de um evento de domínio estende-se, na verdade, para quaisquer requisições. Ou seja, o efeito de uma requisição pode ser representado por meio de uma operação, de maneira análoga ao descrito anteriormente para eventos de domínio.

No que concerne a consultas, na modelagem conceitual define-se apenas o conteúdo de informação das respostas, abstraindo-se detalhes que dizem respeito ao formato e a características de dispositivos de saída (OLIVÉ, 2007). Para que as informações de atributos e associações possam ser recuperadas para serem mostradas como parte das respostas a consultas, as classes precisam prover operações básicas para obter essas informações. Assim como as demais operações básicas, assume-se que toda classe possui implicitamente operações para se obter os valores correntes de seus atributos e associações.

Durante a realização de um caso de uso, atores geram requisições para o sistema, solicitando a execução de ações. O sistema realiza ações e ele próprio pode gerar outras requisições de ação. O conjunto de casos de uso tem de ser consistente com o conjunto de requisições definidas no esquema comportamental do sistema.

7.2 - Diagramas de Gráfico de Estados

Todo objeto tem um tempo de vida. Na criação, o objeto nasce; na destruição, ele deixa de existir. Entre esses dois momentos, um objeto poderá interagir com outros objetos, enviando e recebendo mensagens (BOOCH; RUMBAUGH; JACOBSON, 2006). Essas interações representam o comportamento do objeto e ele pode ser variável ao longo do ciclo de vida do objeto. Ou seja, muitas vezes, o comportamento dos objetos de uma classe depende do estado em que o objeto se encontra em um dado momento. Nestes casos, é útil especificar o comportamento usando uma máquina de estados.

Classes com estados (ou classes modais) são classes cujas instâncias podem mudar de um estado para outro ao longo de sua existência, mudando possivelmente sua estrutura, seus valores de atributos ou comportamento dos métodos (WAZLAWICK, 2004).

Classes modais podem ser modeladas como máquinas de estados finitos. Uma máquina de estados finitos é uma máquina que, em um dado momento, está em um e somente um de um número finito de estados (OLIVÉ, 2007). Os estados de uma máquina de estados de uma classe modal correspondem às situações relevantes em que as instâncias dessa classe podem estar durante sua existência. Um estado é considerado relevante quando ele ajuda a definir restrições ou efeitos dos eventos.

Em qualquer estado, uma máquina de estados pode receber estímulos. Quando a máquina recebe um estímulo, ela pode realizar uma transição de seu estado corrente (dito estado origem) para outro estado (dito estado destino), sendo que se assume que as transições são instantâneas. A definição do estado destino depende do estado origem e do estímulo recebido. Além disso, os estados origem e destino em uma transição podem ser o mesmo. Neste caso, a transição é dita uma autotransição (OLIVÉ, 2007).

Diagramas de Transições de Estados são usados para modelar o comportamento de instâncias de uma classe modal na forma de uma máquina de estados. Todas as instâncias da classe comportam-se da mesma maneira. Em outras palavras, cada diagrama de estados é construído para uma única classe, com o objetivo de mostrar o comportamento ao longo do tempo de vida de seus objetos. Diagramas de estados descrevem os possíveis estados pelos quais objetos da classe podem passar e as alterações dos estados como resultado de eventos (estímulos) que atingem esses objetos. Uma máquina de estado especifica a ordem válida dos estados pelos quais os objetos da classe podem passar ao longo de seu ciclo de vida. A Figura 7.2 mostra a notação básica da UML para diagramas de gráfico de estados.

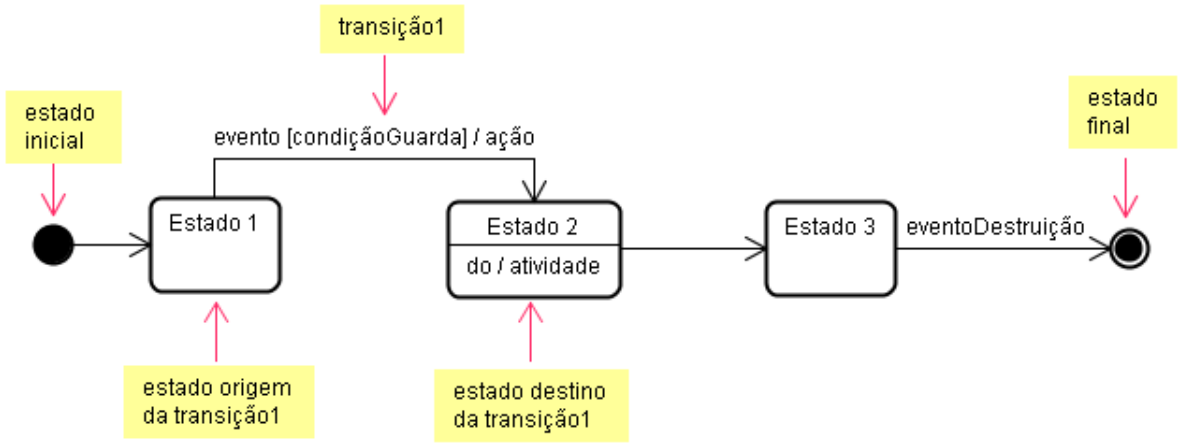

Figura 7.2 - Notação Básica da UML para Diagramas de Gráfico de Estados.

Um estado é uma situação na vida de um objeto durante a qual o objeto satisfaz alguma condição, realiza alguma atividade ou aguarda a ocorrência de um evento (BOOCH; RUMBAUGH; JACOBSON, 2006). Estados são representados por retângulos com os cantos arredondados, sendo que o nome de um estado deve ser único em uma máquina de estados. Uma regra prática para nomear estados consiste em atribuir um nome tal que sejam significativas sentenças do tipo "*o <<objeto>> está <<nome do estado>>*" ou "*o <<objeto>> está no estado <<nome do estado>>*". Por exemplo, em um sistema de locadora de automóveis, um estado possível de objetos da classe *Carro* seria "Disponível". A sentença "o carro está disponível" tem um significado claro (OLIVÉ, 2007).

Quando um objeto fica realizando uma atividade durante todo o tempo em que permanece em um estado, deve-se indicar essa atividade no compartimento de ações do

respectivo estado. É importante realçar que uma atividade tem duração significativa e, quando concluída, tipicamente a conclusão provoca uma transição para um novo estado. A notação da UML para representar atividades de um estado é: **do / <<nomeAtividade>>**.

Transições são representadas por meio de setas rotuladas. Uma transição envolve um estado origem, um estado destino e normalmente um evento, dito o gatilho da transição. Quando a máquina de estados se encontra no estado origem e recebe o evento gatilho, então o evento dispara a transição e a máquina de estados vai para o estado destino. Se uma máquina recebe um evento que não é um gatilho para nenhuma transição, então ela não é afetada pelo evento (OLIVÉ, 2007).

Uma transição pode ter uma condição de guarda associada. Às vezes, há duas ou mais transições com o mesmo estado origem e o mesmo evento gatilho, mas com condições de guarda diferentes. Neste caso, a transição é disparada somente quando o evento gatilho ocorre e a condição de guarda é verdadeira (OLIVÉ, 2007). Quando uma transição não possuir uma condição de guarda associada, então ela ocorrerá sempre que o evento ocorrer.

Por fim, quando uma transição é disparada, uma ação instantânea pode ser realizada. Assim, o rótulo de uma transição pode ter até três partes, todas elas opcionais:

evento [condiçãoGuarda] / ação

Basicamente a semântica de um diagrama de estados é a seguinte: quando o *evento* ocorre, se a *condição de guarda* é verdadeira, a *transição* dispara e a *ação* é realizada instantaneamente. O objeto passa, então, do *estado origem* para o *estado destino*. Se o estado destino possuir uma *atividade* a ser realizada, ela é iniciada.

O fato de uma transição não possuir um evento associado normalmente aponta para a existência de um evento implícito. Isso tipicamente ocorre em três situações: (i) o evento implícito é a conclusão da atividade do estado origem e a transição ocorrerá tão logo a atividade associada ao estado origem tiver sido concluída; (ii) o evento implícito é temporal, sendo disparado pela passagem do tempo; (iii) o evento implícito torna a condição de guarda verdadeira na base de informações do sistema, mas o evento em si não é modelado.

Embora ambos os termos ação e atividade denotem processos, eles não devem ser confundidos. Ações são consideradas processos instantâneos; atividades, por sua vez, estão sempre associadas a estados e têm duração no tempo. Vale a pena observar que, no mundo real, não há processos efetivamente instantâneos. Por mais rápida que seja, uma ação ocorrerá sempre em um intervalo de tempo. Esta simplificação de se considerar ações instantâneas no modelo conceitual pode ser associada à ideia de que a ação ocorre tão rapidamente que não é possível interrompê-la. Em contraste, uma atividade é passível de interrupção, sendo possível, por exemplo, que um evento ocorra, interrompa a atividade e provoque uma mudança no estado do objeto antes da conclusão da atividade.

Às vezes quer se modelar situações em que uma ação instantânea é realizada quando se entra ou sai de um estado, qualquer que seja a transição que o leve ou o retire desse estado. Seja o exemplo de um elevador. Neste contexto, ao parar em um andar, o elevador abre a porta. Suponha que a abertura da porta do elevador seja um processo que não possa ser interrompido e, portanto, que se opte por modelá-lo como uma ação. Essa ação deverá ocorrer sempre que o elevador entrar no estado "*Parado*" e deve ser indicada no compartimento de ações desse estado como sendo uma ação de entrada no estado. A notação da UML para representar ações de entrada em um estado é: **entry / <<nomeAção>>**. Para representar ações de saída de um estado a notação é: **exit / <<nomeAção>>**.

Restam ainda na Figura 7.2 dois tipos especiais de estados: os ditos estados inicial e final. Conforme citado anteriormente, um objeto está sempre em um e somente um estado. Isso implica que, ao ser instanciado, o objeto precisa estar em algum estado. O estado inicial é precisamente esse estado. Graficamente, um estado inicial é mostrado como um pequeno círculo preenchido na cor preta. Seu significado é o seguinte: quando o objeto é criado, ele é colocado no estado inicial e sua transição de saída é automaticamente disparada, movendo o objeto para um dos estados da máquina de estados (no caso da Figura 7.2, para o *Estado1*). Toda máquina de estados tem de ter um (e somente um) estado inicial. Note que o estado inicial não se comporta como um estado normal[20], uma vez que objetos não se mantêm nele por um período de tempo. Ao contrário, uma vez que eles entram no estado inicial, sua transição de saída é imediatamente disparada e o estado inicial é abandonado. A transição de saída do estado inicial tem como evento gatilho implícito o evento responsável pela criação do objeto (OLIVÉ, 2007) e, na UML, esse evento não é explicitamente representado. Estados iniciais têm apenas transições de saída. As transições de saída de um estado inicial podem ter condições de guarda e/ou ações associadas. Quando houver condições de guarda, deve-se garantir que sempre pelo menos uma das transições de saída poderá ser disparada.

Quando um objeto deixa de existir, obviamente ele deixa de estar em qualquer um dos estados. Isso pode ser dito no diagrama por meio de uma transição para o estado final. O estado final indica, na verdade, que o objeto deixou de existir. Na UML um estado final é representado como um círculo preto preenchido com outro círculo não preenchido ao seu redor, como mostra a Figura 7.2. As transições para o estado final definem os estados em que é possível excluir o objeto. Classes cujos objetos não podem ser excluídos, portanto, não possuem um estado final (OLIVÉ, 2007). Assim como o estado inicial, o estado final não se comporta como um estado normal, uma vez que o objeto também não permanece nesse estado (já que o objeto não existe mais). Ao contrário do estado inicial, contudo, uma máquina de estados pode ter vários estados finais. Além disso, deve-se representar o evento que elimina o objeto (na Figura 7.2, *eventoDestruição*).

Os eventos mostrados nas transições são os mesmos eventos de requisição de ação discutidos na Seção 7.1. Contudo, é importante indicar no diagrama de estados os eventos maiores (eventos de domínio e requisições de ações) e não os eventos estruturais que efetivamente alteram o estado do objeto. Assim, neste texto sugere-se indicar como eventos de transições de uma máquina de estados as requisições de realização de casos de uso do sistema (ou de fluxos de eventos específicos, quando um caso de uso tiver mais de um fluxo de eventos normal). Para facilitar a rastreabilidade, sugere-se usar como nome do evento exatamente o mesmo nome do caso de uso (ou do fluxo de eventos). Seja o exemplo de uma locadora de automóveis, que possua, dentre outros, os casos de uso mostrados na Figura 7.3, os quais possuem os fluxos de eventos mostrados nas notas anexadas aos casos de uso.

[20] Por não se comportar como um estado normal, o estado inicial é considerado um pseudoestado no metamodelo da UML.

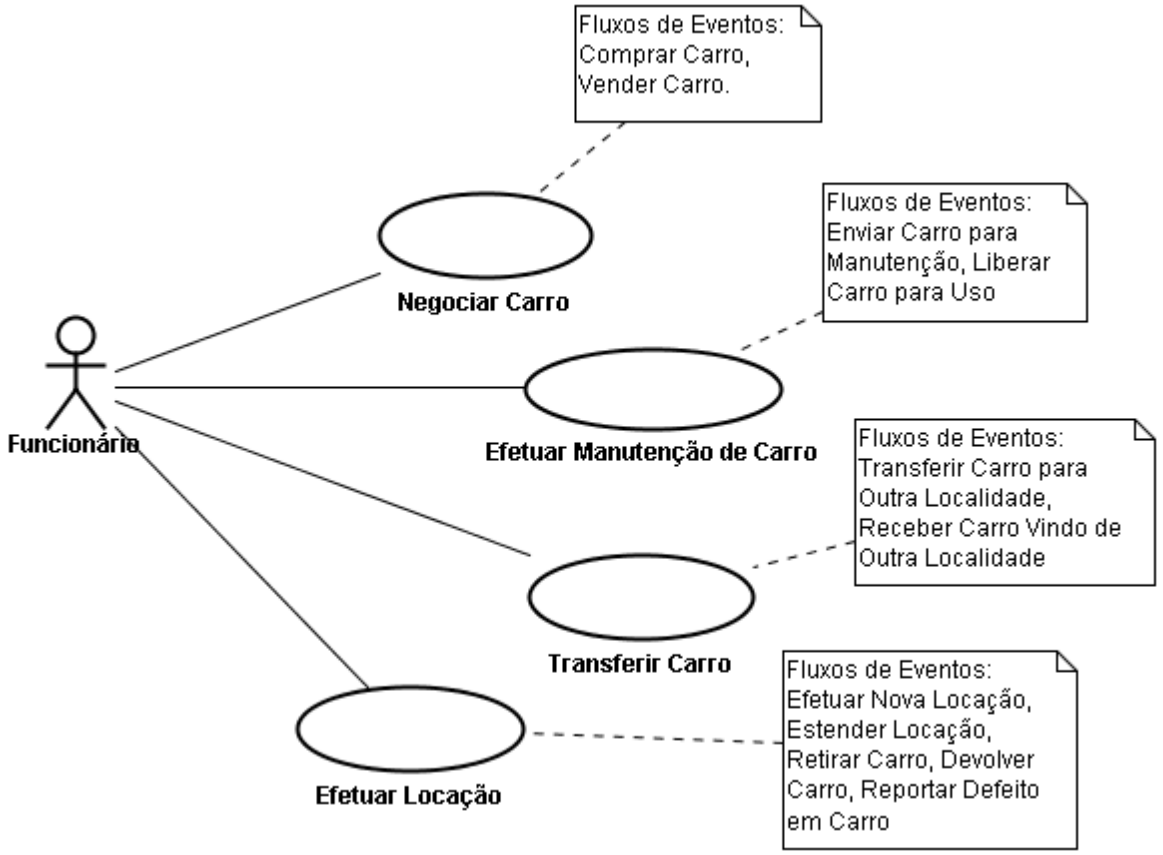

Figura 7.3 – Locadora de Automóveis - Casos de Uso e Fluxos de Eventos Associados.

A classe *Carro* tem o seu comportamento definido pela máquina de estados do diagrama de gráfico de estados da Figura 7.4. Ao ser adquirido (fluxo de eventos *Comprar Carro*, do caso de uso *Negociar Carro*), o carro é colocado *Em Preparação*. Quando liberado para uso (fluxo de eventos *Liberar Carro para Uso*, do caso de uso *Efetuar Manutenção de Carro*), o carro fica *Disponível*. Quando o cliente retira o carro (fluxo de eventos *Retirar Carro*, do caso de uso *Efetuar Locação*), este fica *Em Uso*. Quando é devolvido (fluxo de eventos *Devolver Carro*, do caso de uso *Efetuar Locação*), o carro fica novamente *Em Preparação*. Quando *Disponível*, um carro pode ser transferido de uma localidade para outra (fluxo de eventos *Transferir Carro para Outra Localidade* do caso de uso *Transferir Carro*). Durante o trânsito de uma localidade para outra, o carro está *Em Trânsito*, até ser recebido na localidade destino (fluxo de eventos *Receber Carro Vindo de Outra Localidade*, do caso de uso *Transferir Carro*), quando novamente é colocado *Em Preparação*. Finalmente, carros *Em Preparação* podem ser vendidos (fluxo de eventos *Vender Carro*, do caso de uso *Negociar Carro*), quando deixam de pertencer à locadora e são eliminados de sua base de informações.

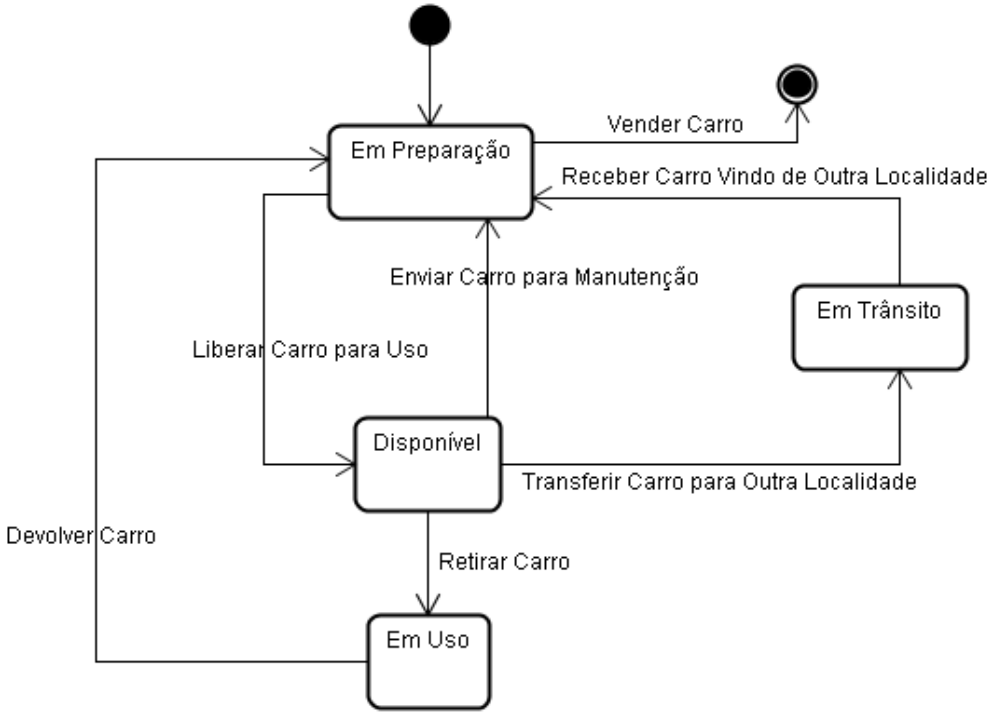

Figura 7.4 – Diagrama de Gráfico de Estados da Classe *Carro* – Disponibilidade (adaptado de (OLIVÉ, 2007)).

Nem todas as classes precisam ser modeladas como máquinas de estados. Apenas classes modais (i.e., classes que apresentam comportamento variável em função do estado de seus objetos) necessitam ser modeladas como máquinas de estados. Além disso, para os diagramas de estados serem efetivamente úteis, recomenda-se modelar uma máquina de estados somente se a classe em questão tiver três ou mais estados relevantes. Se uma classe possuir apenas dois estados relevantes, ainda cabe desenvolver uma máquina de estados. Contudo, de maneira geral, o diagrama tende a ser muito simples e a acrescentar pouca informação relevante que justifique o esforço de elaboração e manutenção do correspondente diagrama. Neste caso, os estados e transições podem ser levantados, sem, no entanto, ser elaborado um diagrama de estados.

Para algumas classes, pode ser útil desenvolver mais do que um diagrama de estados, cada um deles modelando o comportamento dos objetos da classe por uma perspectiva diferente. Em um determinado momento, um objeto está em um (e somente um) estado em cada uma de suas máquinas de estado. Cada diagrama define seu próprio conjunto de estados nos quais um objeto pode estar, a partir de diferentes pontos de vista (OLIVÉ, 2007). Seja novamente o exemplo da classe *Carro*. A Figura 7.4 mostra os possíveis estados de um carro segundo um ponto de vista de disponibilidade. Entretanto, independentemente da disponibilidade, do ponto de vista de possibilidade de negociação, um carro pode estar em dois estados (Não à Venda, À Venda), como mostra a Figura 7.5.

Vale ressaltar que os diferentes diagramas de estados de uma mesma classe não devem ter estados comuns. Cada diagrama deve ter seu próprio conjunto de estados e cada estado pertence a somente um diagrama de estados. Já os eventos podem aparecer em diferentes diagramas de estados, inclusive de classes diferentes. Quando um evento aparecer em mais de um diagrama de estados, sua ocorrência vai disparar as correspondentes transições em cada uma das máquinas de estados em que ele aparecer (OLIVÉ, 2007).

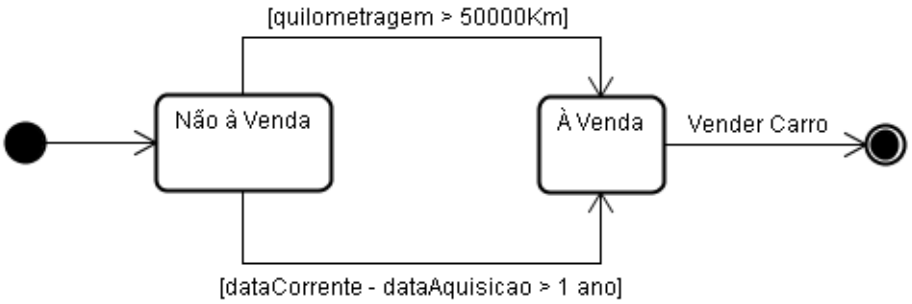

Figura 7.5 – Diagrama de Gráfico de Estados da Classe *Carro* – Possibilidade de Negociação (adaptado de (OLIVÉ, 2007)).

A Figura 7.5 mostra duas transições em que os eventos não são declarados explicitamente. No primeiro caso (quilometragem > 50.000Km), o evento implícito torna a condição de guarda verdadeira na base de informações do sistema. Esse evento corresponde ao registro no sistema de qual é a quilometragem corrente do carro. Caso esse registro ocorra sempre no ato da devolução do carro pelo cliente (fluxo de eventos *Devolver Carro*, do caso de uso *Efetuar Locação*) e/ou no ato do recebimento do carro vindo de outra localidade (fluxo de eventos *Receber Carro Vindo de Outra Localidade*, do caso de uso *Transferir Carro*), esses eventos poderiam ser explicitamente declarados. Contudo, se o registro pode ocorrer em vários eventos diferentes, é melhor deixar o evento implícito. O segundo caso (dataCorrente – dataAquisicao > 1 ano) trata-se de um evento temporal, disparado pela passagem do tempo.

Todos os estados mostrados até então são estados simples, i.e., estados que não possuem subestados. Entretanto, há também estados compostos, os quais podem ser decompostos em um conjunto de subestados disjuntos e mutuamente exclusivos e um conjunto de transições (OLIVÉ, 2007). Um subestado é um estado aninhado em outro estado. O uso de estados compostos e subestados é bastante útil para simplificar a modelagem de comportamentos complexos. Seja o exemplo da Figura 7.4, que trata da disponibilidade de um carro. Suponha que seja necessário distinguir três subestados do estado *Em Uso*, a saber: *Em Uso Normal*, quando o carro não está quebrado nem em atraso; *Quebrado*, quando o cliente reportar um defeito no carro; e *Em Atraso*, quando o carro não foi devolvido na data de devolução prevista e não está quebrado. A Figura 7.6 mostra a máquina de estados da classe *Carro* considerando, agora, que, quando um carro está em uso, ele pode estar nesses três subestados.

Nesse diagrama, não está sendo mostrado que os estados *Em Uso Normal*, *Em Atraso* e *Quebrado* são, de fato, subestados do estado *Em Uso* e, portanto, transições comuns (por exemplo, aquelas provocadas pelo evento *Devolver Carro*) são repetidas. Isso torna o modelo mais complexo e fica claro que esta solução representando diretamente os subestados (e omitindo o estado composto) não é escalável para sistemas que possuem muitos subestados, levando a diagramas confusos e desestruturados (OLIVÉ, 2007). A Figura 7.7 mostra uma solução mais indicada, em que tanto o estado composto quanto seus subestados são mostrados no mesmo diagrama. Uma outra opção é ocultar a decomposição do estado composto, mantendo o diagrama como o mostrado na Figura 7.4, e mostrar essa decomposição em um diagrama de estados separado.

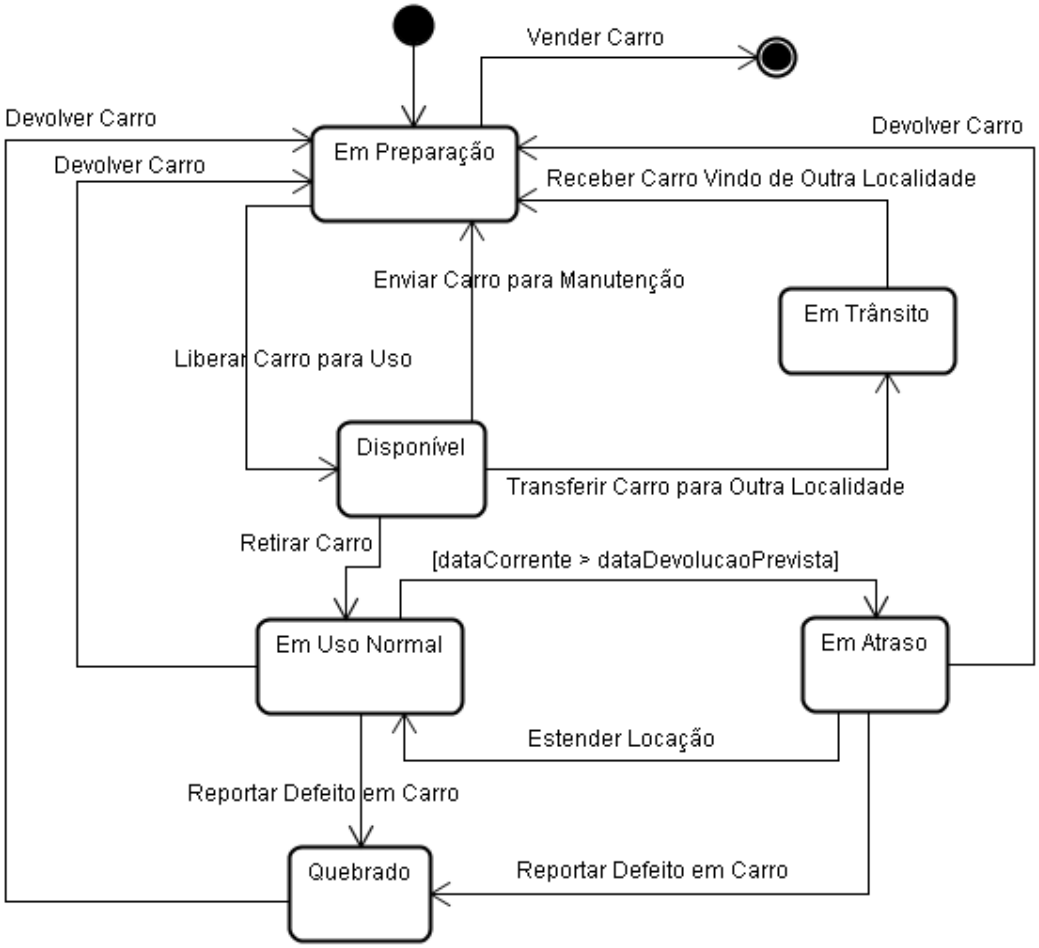

Figura 7.6 – Diagrama de Estados da Classe *Carro* (Disponibilidade) com Subestados de *Em Uso* (adaptado de (OLIVÉ, 2007)).

Se um objeto está em um estado composto, então ele deve estar também em um de seus subestados. Assim, um estado composto pode possuir um estado inicial para indicar o subestado padrão do estado composto, como representado na Figura 7.7. Entretanto, deve-se considerar que as transições podem começar e terminar em qualquer nível. Ou seja, uma transição pode ir (ou partir) diretamente de um subestado (OLIVÉ, 2007). Assim, uma outra opção para o diagrama da Figura 7.7 seria fazer a transição nomeada pelo evento *Retirar Carro* chegar diretamente ao subestado *Em Uso Normal*, ao invés de chegar ao estado composto *Em Uso*.

O estado de um objeto deve ser mapeado no modelo estrutural. De maneira geral, o estado pode ser modelado por meio de um atributo. Esse atributo deve ser monovalorado e obrigatório. O conjunto de valores possíveis do atributo é o conjunto dos estados possíveis, conforme descrito pela máquina de estados (OLIVÉ, 2007). Assim, é bastante natural que o tipo de dados desse atributo seja definido como um tipo de dados enumerado. Um nome adequado para esse atributo é "estado". Contudo, outros nomes mais significativos para o domínio podem ser atribuídos. Em especial, quando uma classe possuir mais do que uma máquina de estado e, por conseguinte, mais do que um atributo de estado for necessário, o nome do atributo de estado deve indicar a perspectiva capturada pela correspondente máquina de estados.

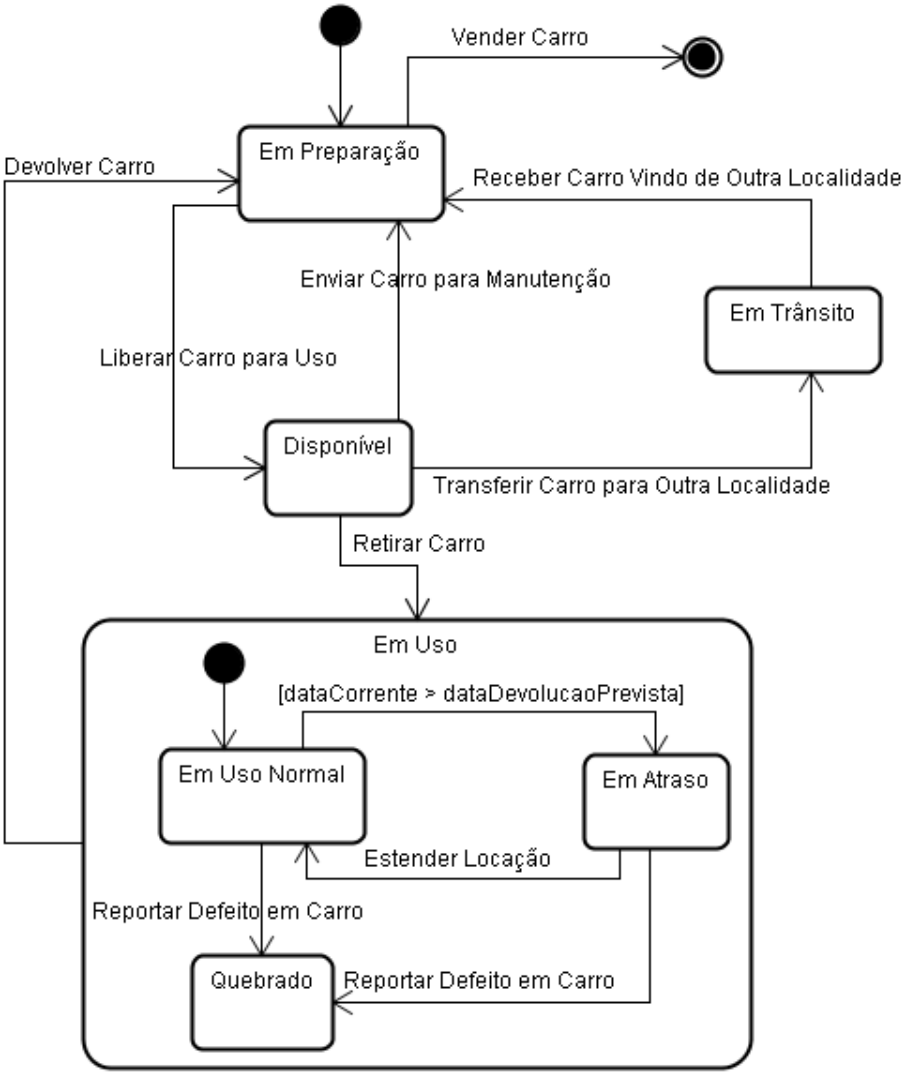

Figura 7.7 – Diagrama de Estados da Classe *Carro* (Disponibilidade) com Estado Composto *Em Uso* (adaptado de (OLIVÉ, 2007)).

É interessante observar que algumas transições podem mudar a estrutura da classe. Quando os diferentes estados de um objeto não afetam a sua estrutura, mas apenas, possivelmente, os valores de seus atributos e associações, diz-se que a transição é estável e os diferentes estados podem ser mapeados para um simples atributo (WAZLAWICK, 2004), conforme discutido anteriormente.

Entretanto, há situações em que, conforme um objeto vai passando de um estado para outro, ele vai ganhando novos atributos ou associações, ou seja, há uma mudança na estrutura da classe. Seja o exemplo de uma locação de carro. Como mostra a Figura 7.8, quando uma locação é criada, ela está ativa, em curso normal. Quando o carro não é devolvido até a data de devolução prevista, a locação passa a ativa com prazo expirado. Se a locação é estendida, ela volta a ficar em curso normal. Quando o carro é devolvido, a locação fica pendente. Finalmente, quando o pagamento é efetuado, a locação é concluída.

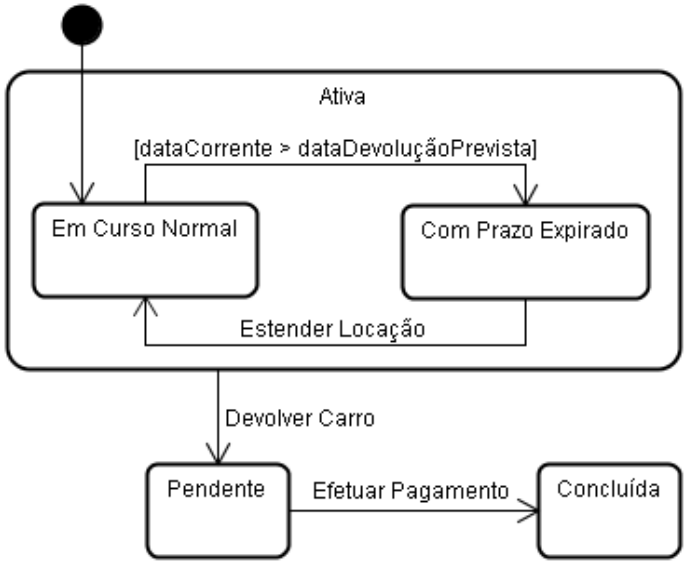

Figura 7.8 – Diagrama de Estados da Classe *Locação*.

Locações ativas (e em seus subestados, obviamente) têm como atributos: data de locação, data de devolução prevista, valor devido e caução. Quando uma locação vai para o estado pendente, é necessário registrar a data de devolução efetiva e os problemas observados no carro devolvido. Finalmente, quando o pagamento é efetuado, é preciso registrar a data do pagamento, o valor e a forma de pagamento. Diz-se que as transições dos estados de *Ativa* para *Pendente* e de *Pendente* para *Concluída* são monotônicas[21], porque a cada mudança de estado, novos relacionamentos (atributos ou associações) são acrescentados (mas nenhum é retirado).

Uma solução frequentemente usada para capturar essa situação no modelo conceitual estrutural consiste em criar uma única classe (*Locacao*) e fazer com que certos atributos sejam nulos até que o objeto mude de estado, como ilustra a Figura 7.9. Essa forma de modelagem, contudo, pode não ser uma boa opção, uma vez que gera classes complexas com regras de consistência que têm de ser verificadas muitas vezes para evitar a execução de um método que atribui um valor a um atributo específico de um estado (WAZLAWICK, 2004), tal como *dataPagamento*.

Figura 7.9 – Classe *Locação* com atributos inerentes a diferentes estados.

[21] Monotônico diz respeito a algo que ocorre de maneira contínua. Neste caso, a continuidade advém do fato de um objeto continuamente ganhar novos atributos e associações, sem perder os que já possuía.

É possível modelar essa situação desdobrando o conceito original em três: um representando a locação efetivamente, outro representando a devolução e outro representando o pagamento. Desta forma, capturam-se claramente os eventos de locação, devolução e pagamento, colocando as informações de cada evento na classe correspondente, como ilustra a Figura 7.10.

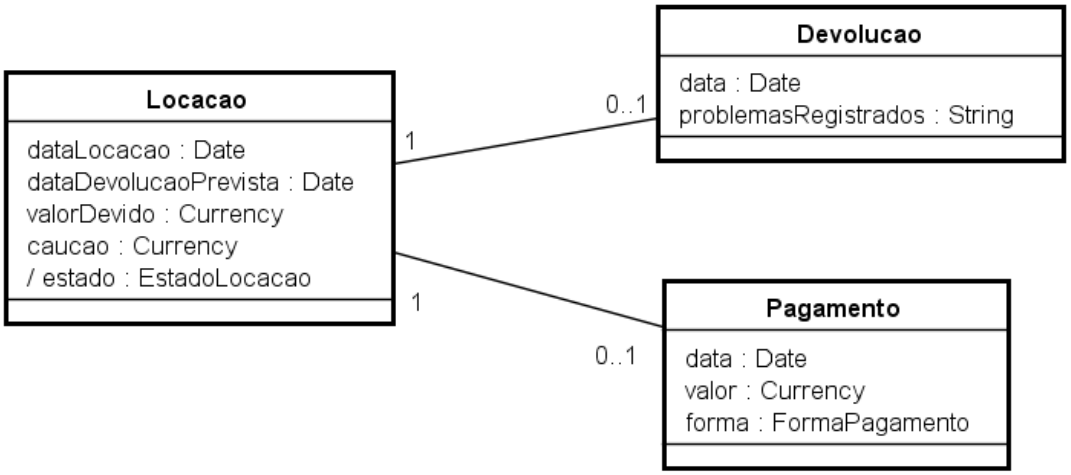

Figura 7.10 – Distribuindo as responsabilidades.

Finalmente, vale a pena comentar que estados de uma classe modal podem ser tratados por meio de operações ao invés de atributos. Seja o exemplo anterior de locações de carros (Figura 7.10). O estado de uma locação pode ser computado a partir dos atributos e associações da classe *Locacao*, sem haver a necessidade de um atributo *estado*. Se uma locação não tem uma devolução associada, então ela está ativa. Estando ativa, se a data corrente é menor ou igual à data de devolução prevista, então a locação está em curso normal; caso contrário, ela está com prazo expirado. Se uma locação possui uma devolução, mas não possui um pagamento associado, então ela está pendente. Finalmente, se a locação possui um pagamento associado, então ela está concluída. Em casos como este, pode-se optar por tratar estado como uma operação e não como um atributo. Opcionalmente, pode-se utilizar a operação para calcular o valor de um atributo derivado[22] *estado*. Atributos derivados são representados na UML precedidos por uma barra (no exemplo, */estado*).

7.3 - Diagramas de Atividades

Um diagrama de atividades é como um fluxograma, no sentido que focaliza o fluxo de controle de uma atividade para outra. Entretanto, ao contrário de um fluxograma tradicional, um diagrama de atividades mostra, além de fluxos sequenciais e ramificações de controle, fluxos concorrentes (BOOCH; RUMBAUGH; JACOBSON, 2006; BLAHA; RUMBAUGH, 2006). O propósito de um diagrama de atividades é mostrar as etapas de um processo complexo e a sequência entre elas (BLAHA; RUMBAUGH, 2006).

Diagramas de atividades são usados para representar processos, sendo utilizados tanto para modelar processos de negócio quanto para representar a realização de um caso de uso. Eles foram adicionados à UML relativamente tarde, adquirindo status de um tipo de diagrama

[22] Um atributo é derivado quando seu valor pode ser deduzido ou calculado a partir de outras informações (atributos e associações) já existentes no modelo estrutural.

somente na UML 2.0. Até a UML 1.5, diagramas de atividades eram considerados um tipo especial de diagrama de estados.

Os diagramas de atividades são muito usados para modelar processos de negócio e fluxos de trabalho em organizações (ver Seção 3.3), uma vez que esses processos / fluxos de trabalho envolvem muitas pessoas e unidades organizacionais que realizam atividades concorrentemente (BLAHA; RUMBAUGH, 2006). Principalmente no contexto de sistemas corporativos e de missão crítica, o sistema em desenvolvimento estará funcionando no contexto de processos de negócio de mais alto nível e pode ser útil usar diagramas de atividades para modelar esses processos com o intuito de investigar as formas como humanos e os vários sistemas automatizados colaboram (BOOCH; RUMBAUGH; JACOBSON, 2006). Esse importante uso dos diagramas de atividades, contudo, está fora do escopo deste texto e, portanto, não será aqui abordado.

No contexto da modelagem comportamental de sistemas (foco deste texto), os diagramas de atividades podem ser usados para modelar o fluxo de trabalho em um caso de uso. Para essa finalidade, os principais elementos de modelo dos diagramas de atividades da UML utilizados são: atividades, fluxos de controle, pontos de iniciação e conclusão, desvios (ou ramificações), bifurcação e união, fluxos de objetos e regiões de expansão. Na modelagem de processos, utilizam-se também raias para indicar quem (que pessoa ou unidade organizacional) é responsável por uma atividade.

Uma atividade é uma porção significativa de trabalho dentro de um fluxo de trabalho. Atividades podem ser atômicas ou complexas. Uma atividade atômica (i.e., que não pode ser decomposta) é dita uma ação na UML. Uma atividade complexa é composta de outras atividades (atômicas ou complexas) e na UML é representada por um nó de atividade. Assim, um nó de atividade representa um grupo de ações ou de outros nós de atividade aninhados, que possui uma subestrutura visível, representada em um outro diagrama de atividades (BOOCH; RUMBAUGH; JACOBSON, 2006). Atividades são representadas por elipses alongadas. Quando uma atividade é concluída, o fluxo de controle passa imediatamente para a atividade seguinte. O fluxo de controle é mostrado por meio de uma seta não rotulada de uma atividade para a sua sucessora.

O fluxo de controle inicia e termina em algum lugar. Os pontos de iniciação e de conclusão do fluxo de controle são mostrados em um diagrama de atividades usando a mesma notação de estados inicial e final de diagramas de gráficos de estados, respectivamente. Quando um diagrama de atividades é ativado, o fluxo de controle inicia no ponto de iniciação e avança por meio da(s) seta(s) de fluxo de controle em direção à(s) primeira(s) atividade(s) a ser(em) realizada(s). Quando o ponto de conclusão é atingido, todo o processo é encerrado e a execução do diagrama de atividades termina (BOOCH; RUMBAUGH; JACOBSON, 2006; BLAHA; RUMBAUGH, 2006). A Figura 7.11 mostra a notação da UML para atividades, fluxos de controle e pontos de iniciação e conclusão.

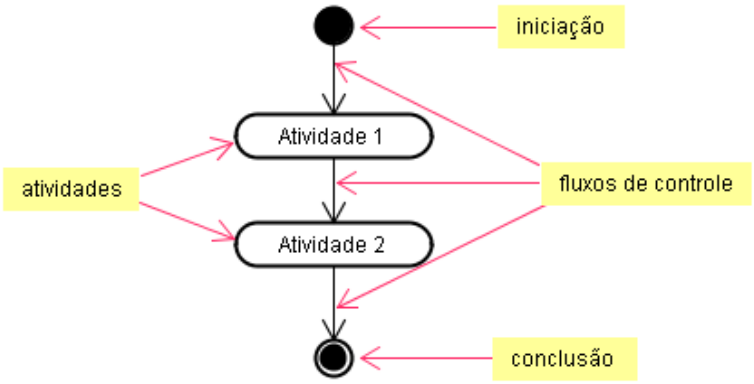

Figura 7.11 - Notação Básica da UML para Diagramas de Atividades.

Os elementos da Figura 7.11 só permitem modelar sequências de atividades. Contudo, a maioria dos fluxos de trabalho envolve fluxos alternativos, concorrentes e/ou iterativos. Para representar essas estruturas de controle, são necessários outros elementos de modelo. Em diagramas de atividades da UML, fluxos alternativos, concorrentes e/ou iterativos podem ser representados por meio de desvios (ou ramificações), bifurcações e uniões, e regiões de expansão, respectivamente.

Um desvio ou ramificação permite especificar caminhos alternativos a serem seguidos, tomando por base alguma expressão booleana. Uma ramificação possui um fluxo de entrada e dois ou mais de saída. Em cada fluxo de saída é colocada uma expressão booleana, a qual é avaliada quando o controle atinge a ramificação. As condições não podem se sobrepor, pois senão o fluxo de controle poderia seguir por mais de um caminho, o que não é admissível em uma ramificação. Além disso, elas têm de cobrir todas as possibilidades, pois, caso contrário o fluxo de controle pode ficar sem ter para onde seguir em alguma situação. Para evitar esse problema, pode-se utilizar a condição *else*, a qual é satisfeita caso nenhuma outra condição seja satisfeita (BOOCH; RUMBAUGH; JACOBSON, 2006; BLAHA; RUMBAUGH, 2006).

Uma ramificação é representada na UML por um losango. Quando dois caminhos de controle alternativos se fundem novamente, pode-se utilizar o mesmo símbolo para mesclá-los. Na fusão, contudo, há duas ou mais setas de fluxo de controle de entrada e somente uma de saída. A Figura 7.12 ilustra a notação de desvios e fusões.

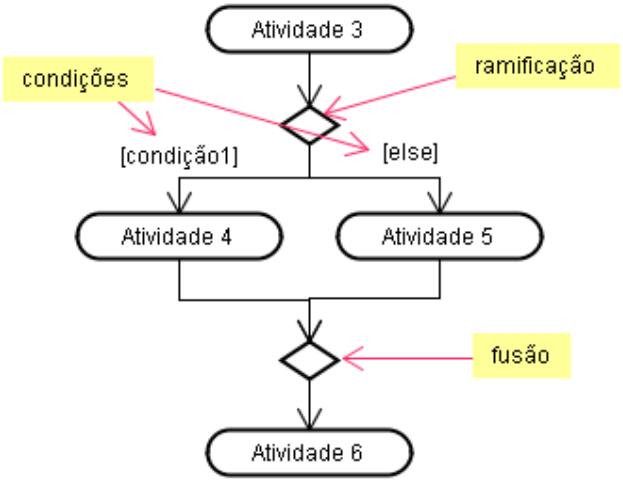

Figura 7.12 – Ramificações em Diagramas de Atividades da UML.

Para mostrar atividades realizadas concorrentemente, podem-se utilizar bifurcações e uniões, as quais são representadas por barras de sincronização. Uma barra de sincronização é representada por uma linha grossa horizontal ou vertical. Uma bifurcação tem de ter um único fluxo de controle de entrada e dois ou mais fluxos de saída, cada um deles representando um fluxo de controle independente. As atividades associadas a cada um desses caminhos prosseguem paralelamente, indicando que as atividades ocorrem concorrentemente. Já uma união representa a sincronização de um ou mais fluxos de controle concorrentes. Uma união tem de ter dois ou mais fluxos de entrada e apenas um fluxo de controle de saída. Na união, os fluxos concorrentes de entrada são sincronizados, significando que cada um deles aguarda até que todos os fluxos de entrada tenham atingido a união, a partir da qual o fluxo de controle de saída prossegue. De maneira geral, deve haver um equilíbrio entre bifurcações e uniões, indicando que o número de fluxos de controle que deixam uma bifurcação deve ser equivalente ao número de fluxos que chegam a uma união correspondente (BOOCH; RUMBAUGH; JACOBSON, 2006). A Figura 7.13 ilustra a notação de bifurcações e uniões.

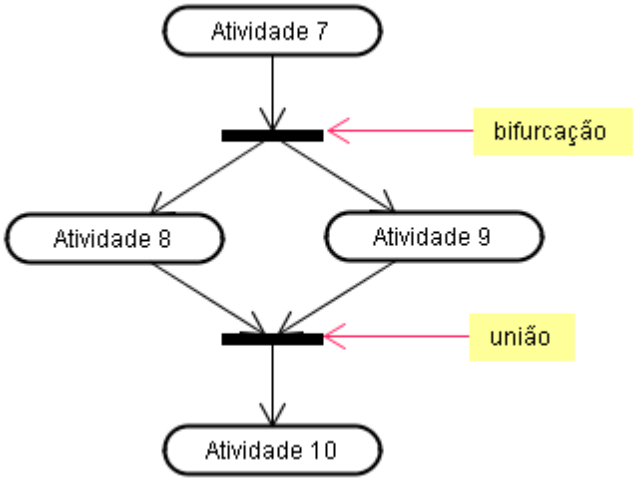

Figura 7.13 – Bifurcações e Uniões em Diagramas de Atividades da UML.

Para representar atividades que ocorrem várias vezes, operando sobre elementos de um conjunto, podem-se utilizar regiões de expansão. Uma região de expansão representa um fragmento do diagrama de atividades que é realizado operando sobre os elementos de uma lista ou conjunto. Ela é representada por uma linha tracejada em torno da região do diagrama que envolve as atividades iterativas. A região é executada uma vez para cada elemento do conjunto de entrada (BOOCH; RUMBAUGH; JACOBSON, 2006).

Muitas vezes é útil representar os objetos requeridos (entradas) e produzidos (saídas) por uma atividade em um diagrama de atividades. É possível especificar os objetos envolvidos nas atividades, conectando-os às atividades que os produzem ou consomem. Além de representar objetos e o seu fluxo nas atividades, pode-se representar, ainda, o estado em que se encontra o objeto. A Figura 7.14 mostra a notação de objetos, fluxos de objetos e estado do objeto.

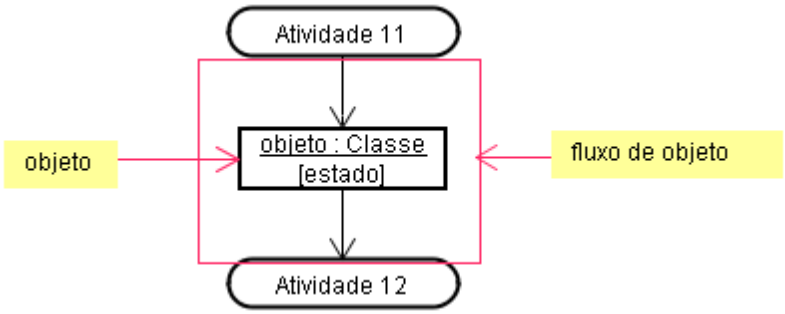

Figura 7.14 – Objetos e Fluxos de Objetos em Diagramas de Atividades da UML.

Um fluxo de objetos implica em um fluxo de controle, pois representa o fluxo de um objeto de uma atividade para outra. Portanto, é desnecessário desenhar um fluxo de controle entre as atividades conectadas por fluxos de objetos (BOOCH; RUMBAUGH; JACOBSON, 2006).

Neste texto, defendemos o uso de diagramas de atividades para complementar a visão comportamental de casos de uso complexos. Sugere-se elaborar um diagrama de atividades para cada fluxo de eventos normal complexo, mostrando no mesmo diagrama o fluxo de eventos normal e os correspondentes fluxos variantes e de exceção. Elaborar diagramas de atividade para casos de uso simples, tais como casos de uso cadastrais e de consulta, dificilmente adicionará algum valor ao projeto e, portanto, tais diagramas são dispensáveis.

Para ajudar os usuários a entender um diagrama de atividades, pode ser útil ilustrar a execução do mesmo usando fichas de atividade. Uma ficha é colocada no ponto de iniciação do diagrama e ela vai sendo deslocada pelas atividades do diagrama. Quando houver uma bifurcação, múltiplas fichas vão surgir, uma para cada fluxo de saída. De maneira análoga, uma união do controle reduz o conjunto de fichas de entrada para uma única ficha de saída (BLAHA; RUMBAUGH, 2006).

7.4 – Especificação das Operações

Uma vez estudado o comportamento do sistema, tem-se uma base para a definição das operações das classes que compõem o sistema. Operações correspondem a serviços que podem ser solicitados aos objetos de uma classe e são apresentadas na seção inferior do símbolo de classe, com a seguinte sintaxe[23] para a sua assinatura (BOOCH; RUMBAUGH; JACOBSON, 2006):

visibilidade nome(lista_de_parâmetros): tipo_de_retorno

A visibilidade de uma operação indica em que situações ela é visível por outras classes, podendo uma operação ser pública, protegida, privada e de pacote, da mesma forma que atributos (ver Seção 6.2.1). A informação de visibilidade é inerente à fase de projeto e não deve ser expressa em um modelo conceitual.

Para nomear uma operação, sugere-se o uso de um verbo no infinitivo, o qual pode ser combinado com complementos, omitindo-se preposições. A exceção fica por conta de operações com retorno booleano, para as quais se sugere usar um nome que dê uma noção de

[23] A UML admite outras informações na assinatura de uma operação. Entretanto, essas são as notações mais comumente utilizadas.

uma pergunta sendo feita. O nome da operação deve ser iniciado com letra minúscula, enquanto os nomes dos complementos devem iniciar com letras maiúsculas, sem dar um espaço em relação à palavra anterior. Acentos não devem ser utilizados. Ex.: calcularValor, emAtraso.

Na assinatura de uma operação, podem ser indicados zero ou mais parâmetros separados por vírgula, cada um deles com a sintaxe abaixo[24], onde *nome_parâmetro* é o nome para referenciar o parâmetro, *tipo* é seu tipo de dados ou classe, e *valor_padrão* é o valor que será assumido se um valor não for informado.

nome_parâmetro: tipo [= valor_padrão]

O *tipo_de_retorno* indica a classe ou o tipo de dado do valor retornado pela operação, o qual pode ser uma classe, um tipo de dado primitivo ou um tipo de dado específico de domínio. Caso uma operação não tenha retorno, nada é especificado.

Conforme discutido anteriormente, há operações, ditas básicas, que simplesmente manipulam atributos e associações, criam ou destroem objetos. Essas operações não são representadas nos diagramas de classes, nem especificadas e documentadas no Dicionário de Projeto, já que podem ser deduzidas do modelo conceitual estrutural. As demais operações devem ser descritas no Dicionário de Projeto, dando uma descrição sucinta de seu propósito.

Referências do Capítulo

BLAHA, M., RUMBAUGH, J., *Modelagem e Projetos Baseados em Objetos com UML 2*, Elsevier, 2006.

BOOCH, G., RUMBAUGH, J., JACOBSON, I., *UML Guia do Usuário*, 2a edição, Elsevier Editora, 2006.

OLIVÉ, A., *Conceptual Modeling of Information Systems*, Springer, 2007.

WAZLAWICK, R.S., *Análise e Projeto de Sistemas de Informação Orientados a Objetos*, Elsevier, 2004.

[24] Idem comentário anterior.