UTILIZANDO UML E PADRÕES

3ª Edição

Uma introdução à análise e ao projeto orientados a objetos e ao desenvolvimento iterativo

CRAIG LARMAN

Prefácio de **Philippe Kruchten**

"Com freqüência me perguntam qual é o melhor livro para conhecer o projeto orientado a objeto. Desde que o conheci, *Utilizando UML e Padrões* é a minha sugestão."
Martin Fowler, autor de *UML Essencial*

Bookman®

Capítulo

37

Diagramas UML de Implantação e de Componentes

Chame-me paranóico, mas encontrar '/' dentro desse comentário me causa suspeitas.*
– Um alerta do compilador C da MPW

Objetivos

- Resumir a notação de diagramas UML de implantação e componente.

O que vem a seguir? Enfocado mais projeto de objetos com padrões GoF, este capítulo resume a notação UML útil na documentação da arquitetura do sistema. O capítulo seguinte mostra a aplicação de diversos padrões GoF no projeto de um framework de persistência.

37.1 Diagramas de implantação

Um diagrama de implantação mostra a atribuição de artefatos concretos de software (como arquivos executáveis) a nós computacionais (algo com serviços de processamento). Mostra a implantação de elementos de software à **arquitetura física** e a comunicação (geralmente em uma rede) entre elementos físicos. Veja a Figura 37.1. Diagramas de implantação são úteis para comunicar a arquitetura física ou de implantação, por exemplo, no Documento de Arquitetura de Software do PU discutido a partir da pág. 649.

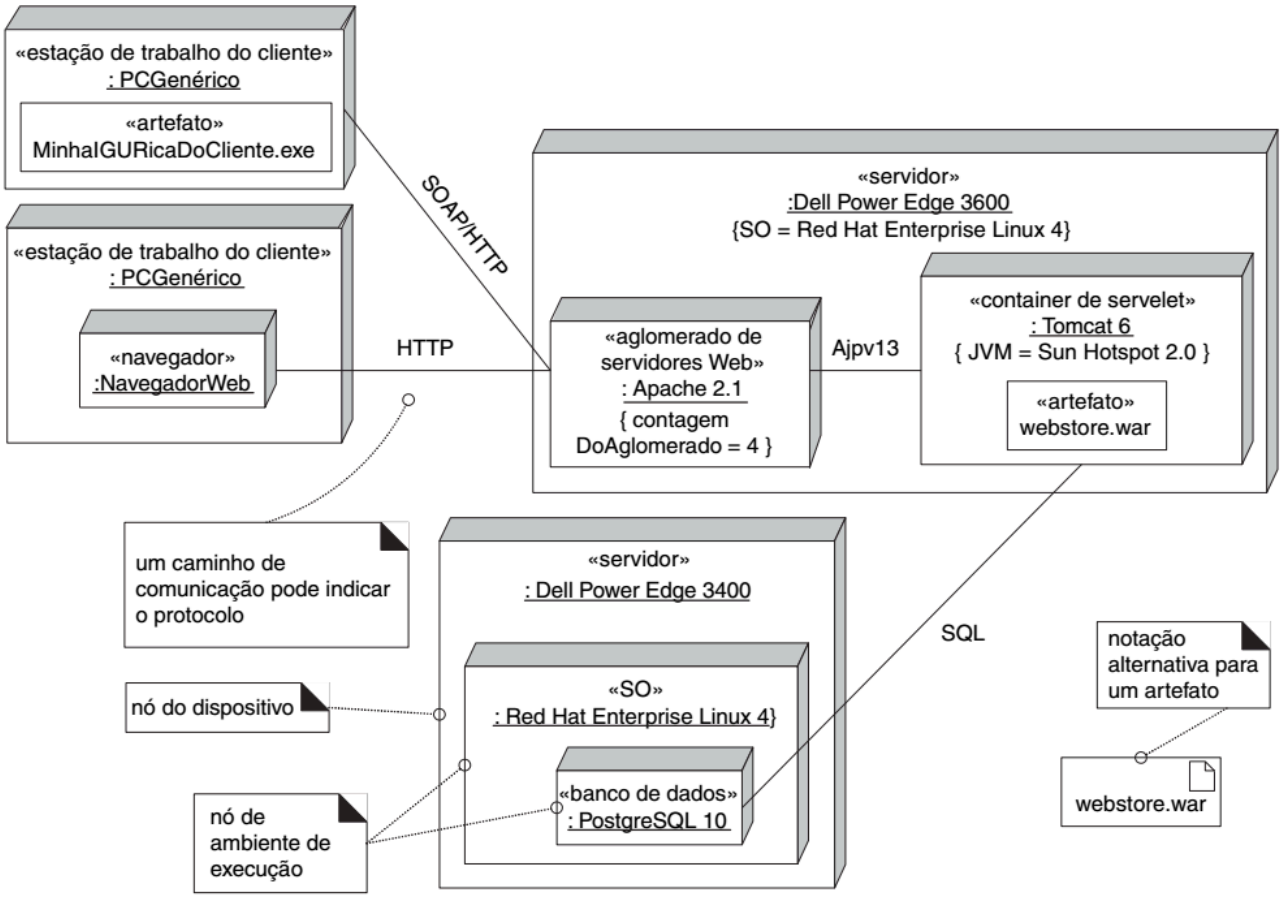

Figura 37.1 Um diagrama de implantação.

O elemento básico de um diagrama de implantação é um **nó,** de dois tipos:

- **Nó de dispositivo** (ou **dispositivo**) – Um recurso computacional físico (por exemplo, eletrônico digital) com serviços de processamento e memória para executar software, por exemplo, um computador típico ou um telefone móvel.
- **Nó de ambiente de execução** (execution environment node – EEN) – Esse é um recurso computacional de software que executa em um nó externo (como um computador) e que em si fornece um serviço ao hospedeiro e executa outros elementos de software executável. Por exemplo:
  - *Um sistema operacional* (SO) é um software que hospeda e executa programas.
  - *Uma máquina virtual* (MV, como Java ou .NET VM) hospeda e executa programas.
  - *Um motor de banco de dados* (por exemplo, PostgreSQL) recebe solicitações de um programa SQL e as executa, e hospeda/executa procedimentos internos armazenados (escritos em Java ou em uma linguagem proprietária).
  - *Um navegador da Web* hospeda e executa JavaScript, applets Java, Flash, e outras tecnologias executáveis.

- *Um motor de fluxo de trabalho*
- *Um contêiner servlet ou um contêiner EJB.*

Como a especificação UML sugere, muitos tipos de nó podem mostrar estereótipos, como «servidor», «SO», «banco de dados» ou «navegador», mas esses não são estereótipos predefinidos oficiais da UML.

Note que um nó de dispositivo ou EEN pode conter um outro EEN. Por exemplo, uma máquina virtual dentro de um SO dentro de um computador.

Um EEN específico pode estar implícito, ou não mostrado, ou indicado informalmente com uma cadeia de propriedade UML, como {SO= Linux}. Por exemplo, pode não haver valor em mostrar o SO EEN como um nó explícito. A Figura 37.1 mostra estilos alternativos, usando um SO como exemplo.

A conexão normal entre nós é um **caminho de comunicação**, que pode ser rotulado com o protocolo. Esse geralmente indica as conexões de rede.

Um nó pode conter e mostrar um **artefato** – um elemento físico concreto, geralmente um arquivo. Isso inclui executáveis, como JARs, montadores, arquivos .exe e scripts. Também inclui arquivos de dados, como XML, HTML, etc.

Um diagrama de implantação geralmente mostra um conjunto-exemplo de *instâncias* (em vez de classes). Por exemplo, uma instância de um computador servidor executando uma instância do SO Linux. Geralmente, na UML, **instâncias** concretas são mostradas com um sublinhado sob seu nome e a ausência de um sublinhado significa uma classe em vez de uma instância. Note que uma exceção importante dessa regra é mostrar instâncias em diagramas de interação, em que os nomes de coisas significando instâncias nas caixas de linha de vida não são sublinhados.

Em qualquer evento, em diagramas de implantação, você geralmente verá os objetos com seus nomes sublinhados, para indicar instâncias. No entanto, a especificação UML estabelece que para diagramas de implantação, o sublinhado pode ser omitido e assumido. Assim, você pode ver exemplos nos dois estilos.

37.2 Diagramas de componentes

Componentes são um conceito ligeiramente confuso na UML, porque tanto classes quanto componentes podem ser usados para modelar a mesma coisa. Por exemplo, citando Rumbaugh (um dos criadores da UML):

> *A distinção entre uma classe estruturada e um componente é algo vago e mais uma questão de intenção do que de semântica rígida. [RJB04]*

E para citar a especificação UML [OMG03b]:

> *Um **componente** representa uma parte modular de um sistema que encapsula seu conteúdo e cuja manifestação é substituível dentro de seu ambiente. Um componente define seu comportamento em termos de interfaces fornecidas e requeridas. Como tal, um componente serve como um tipo, cuja conformidade é definida por essas interfaces fornecidas e requeridas.*

Outra vez, essa idéia pode ser modelada com uma classe UML regular e suas interfaces fornecidas e requeridas. Lembre que uma classe UML pode ser usada para modelar qualquer nível de elemento de software, de um sistema completo a subsistemas e até um pequeno objeto utilitário.

Mas quando alguém usa um componente UML, a intenção da modelagem e projeto é enfatizar 1) que as *interfaces* são importantes e 2) é *modular, auto-contido* e *substituível*. O segundo ponto implica que um componente tende a ter pouca ou nenhuma dependência de outros elementos externos (exceto talvez bibliotecas centrais normalizadas); isso é um módulo relativamente independente (stand-alone).

Componentes UML são uma perspectiva no nível de projeto; eles não existem na perspectiva concreta de software, mas mapeiam para artefatos concretos, como um conjunto de arquivos.

Uma boa analogia para modelagem de componentes de software é um sistema de entretenimento doméstico; esperamos poder facilmente substituir o reprodutor de DVD ou auto-falantes. Eles são modulares, auto-contidos, substituíveis e funcionam por meio de interfaces normalizadas. Por exemplo, em um nível de granularidade grossa, um motor de banco de dados SQL pode ser modelado com um componente; qualquer banco de dados que entende a mesma versão de SQL e apóia a mesma semântica de transações pode ser substituído. Em um nível mais fino, qualquer solução que implementa a API Java normalizada para Serviço de Mensagens pode ser usado ou substituído em um sistema.

Como a ênfase da modelagem baseada em componentes é em partes substituíveis (talvez para melhorar as qualidades não-funcionais, como desempenho), é uma diretriz geral fazer modelagem de componentes para elementos relativamente de larga escala, porque é difícil raciocinar sobre ou projetar para muitas partes substituíveis pequenas, de granularidade fina. A Figura 37.2 ilustra a notação essencial.

O tópico de modelagem e desenvolvimento baseados em componentes é um assunto extenso, dedicado, especializado e fora do escopo desta introdução de A/POO.

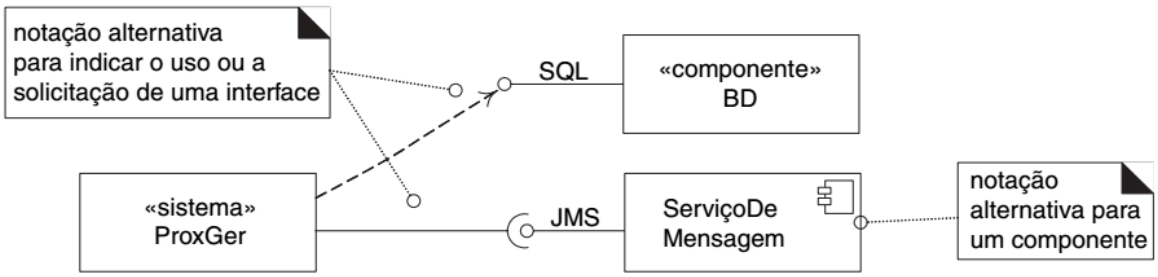

Figura 37.2 Componentes UML.