"Component and Deployment Diagrams"
(Chapter 5*)

from Learning UML: Communicating Software Design Graphically
by Sinan Si Alhir
(O'Reilly, 2003)

"The Unified Modeling Language is a language for communicating about systems: an evolutionary, general-purpose, broadly applicable, tool-supported, and industry-standardized modeling language for specifying, visualizing, constructing, and documenting the artifacts of a system-intensive process." So writes Sinan Si Alhir in his new tutorial for this marvelous language, which was conceived by Rational Software Corporation's three amigos: Grady Booch, James Rumbaugh, and Ivar Jacobson.

Sinan Si Alhir's book focuses on mastering UML essentials in an orderly fashion. Featuring an example-driven approach and an evolving project management system case study, it progressively introduces and demonstrates the application of key concepts.

In last month's issue we featured Chapter 4, "Use-Case Diagrams." This month we offer Chapter 5, which shows how to model a system's implementation and environment, respectively. With UML-based component modeling, a special type of structural modeling, you can represent the system implementation and determine the elements of the system on which the implementation will focus. A deployment model, in contrast, represents the external resources that these components require and helps teams determine how deployment activities will make the system available to users. Both types of modeling usually begin when the system design is fairly complete. Covering basics for both types of models, this chapter includes a discussion of nodes and various relationships with respect to components and nodes.

*Chapter 5 is posted in its entirety by permission from O'Reilly (www.oreilly.com).

Chapter 5 pdf file (748 K)
CHAPTER 5

Component and Deployment Diagrams

This chapter focuses on component and deployment diagrams, which depict the implementation and environment of a system, respectively. First, I introduce component and deployment diagrams and how they are used. Next, I discuss components and nodes, which are elements depicted on those diagrams. Finally, I discuss various relationships relating to components and nodes. Many details of our project management system that were not fleshed out in Chapter 2 are more fully elaborated here, and throughout the chapter, I include suggestions relating to component and deployment diagrams.

Component modeling is a specialized type of structural modeling concerned with modeling the implementation of a system. Using the UML, you can communicate the implementation of a system using component diagrams. You usually apply component modeling during design activities to determine how implementation activities will build the system; that is, to determine the elements of the system on which implementation activities will focus. Component modeling typically starts after the design of the system is fairly complete, as determined by your system development process.

Deployment modeling is a specialized type of structural modeling concerned with modeling the implementation environment of a system. In contrast to modeling the components of a system, a deployment model shows you the external resources that those components require. You typically apply deployment modeling during design activities to determine how deployment activities will make the system available to its users; that is, to determine the elements of the system on which deployment activities will focus. Like component modeling, deployment modeling usually starts after the design of the system is fairly complete, as determined by your system development process.

Components

As mentioned in Chapter 2, a component is a part of the system that exists when the system is executing. For example, the project management system may be decomposed into the following components:
A user interface component
Responsible for providing a user interface through which users may interact with the system

A business-processing component
Responsible for implementing business functionality, including all the project management functionality provided by the project management system

A data component
For implementing data storage functionality

A security component
Provides various forms of security functionality to the business-processing and data components, including user authentication and verifying user privileges when accessing data

Components follow the type-instance dichotomy first discussed in Chapter 2 and applied to classes and objects in Chapter 3. You can use the UML to talk about classes of components as well as specific components of a class. When speaking of a class of components, it’s customary to use the terms component or component class. Thus, while you might think of a component as a specific thing, in the UML, a component really represents a class of things. When speaking of a specific component of a class, use the term component instance.

A component exists during execution time and requires a resource on which to execute, which I talk about in the next section, “Nodes.” In the UML, a component is shown as a rectangle with two small rectangles protruding from its side. The rectangle is labeled with the name of the component class.

Figure 5-1 shows various components associated with the project management system, including user interface, business-processing, data, and security components.

A component instance is a specific component. For example, specific components of the project management system include:

A web user interface component instance
Allows users to access the project management system via the Web

A client/server user interface component instance
Allows users to access the project management system in a client/server environment
A local data component instance
Stores project management data for a specific user or group of users

An enterprise data component instance
Stores project management data for a complete organization

A component instance is shown similar to a component class, but is labeled with the component instance name followed by a colon followed by the component class name, with all parts of the name fully underlined. Both names are optional, and the colon is present only if the component class name is specified.

Figure 5-2 shows various component instances of the component classes in Figure 5-1, including two user interface component instances, named Web and Client Server, two data component instances, named Local Data and Enterprise Data, a nameless business processing component instance, and a nameless security component instance.

![Figure 5-2. Component instances in the project management system](image)

**Nodes**

A node is a resource that is available during execution time. (Nodes were mentioned in Chapter 2.) Traditionally, nodes refer to computers on a network, but in the UML a node may be a computer, printer, server, Internet, or any other kind of resource available to components. For example, the project management system may be deployed on the following nodes:

A desktop client
On which the user interface component executes

A printer
Which the project management system uses to print reports

A business-processing server
On which the business-processing component executes

A database server
On which the data component executes and where project-related information is stored
Nodes follow the type-instance dichotomy first discussed in Chapter 2 and applied to classes and objects in Chapter 3. You can use the UML to talk about classes of nodes, as well as specific nodes of a class. When speaking of a class of nodes, it’s customary to use the terms node or node class. Thus, while you might think of a node as a specific thing, in the UML, a node really represents a class of nodes. When speaking of a specific component of a class, use the term node instance.

A node is available during execution time and is a resource on which components may execute. In the UML, a node is shown as a three-dimensional rectangle labeled with the node’s name.

Figure 5-3 shows various nodes associated with the project management system, including a desktop client, business-processing server, database server, and printer node.

![Nodes used by the project management system](image)

A node instance is a specific node. For example, specific nodes used by the project management system include:

* **A desktop client node instance**
  
  Used by Jonathan to access the project management system

* **A desktop client node instance**
  
  Used by Andy to access the project management system

* **A group business-processing server node instance**
  
  Used by a group of users to manage projects

* **An enterprise business-processing server node instance**
  
  Used by a complete organization to manage projects

A node instance is shown similarly to a node class but labeled with the node instance name followed by a colon followed by the node class name, all fully underlined. Both names are optional, and the colon is present only if the node class name is specified.

Figure 5-4 shows various node instances of the node classes in Figure 5-3, including two desktop client node instances, named Jonathan’s Computer and Andy’s Computer, two business-processing node instances, named Group Server and Enterprise Server, a printer node instance, named Group Printer, and a database server node instance.
Dependencies

Figure 5-1 shows components associated with the project management system, and Figure 5-3 shows nodes associated with the project management system, but how are components related to undifferentiated and differentiated classes, packages, subsystems, and to other components and nodes? Specialized types of dependencies—called reside, use, and deploy dependencies—address these questions. The next few sections in this chapter discuss these specialized types of dependencies. Dependencies in general are discussed in Chapter 3.

Reside Dependencies

A reside dependency from a component to any UML element indicates that the component is a client of the element, which is itself considered a supplier, and that the element resides in the component. The element may be an undifferentiated or differentiated class, package, or subsystem. An element may reside in any number of components, and a component may have any number of elements that reside in it.

A reside dependency is shown as a dashed arrow from a client component to a supplier element marked with the reside keyword. Figure 5-5 shows that the User Interface and Utility packages reside in the User Interface component. Because the User Interface package depends on the Utility package, the User Interface and Utility packages must reside in the same component; otherwise, the User Interface package would not be able to use the Utility package.

Figure 5-6 shows that the Business Processing subsystem and Utility package reside in the Business Processing component. Because the Business Processing subsystem provides the IBusiness Processing interface, the Business Processing component also provides the interface. Again, because the Business Processing subsystem depends on the Utility package, the Business Processing subsystem and Utility
package must reside in the same component; otherwise, the Business Processing subsystem would not be able to use the Utility package. Remember, it’s perfectly fine for an element to reside in more than one component. For example, the Utility package resides in both the User Interface and Business Processing components, and, as you will soon see, in the Data component.

Alternatively, an element that resides inside a component may be shown nested inside the component. Figure 5-7 shows that the Data subsystem and Utility package reside in the Data component. The Data subsystem is drawn inside the Data component, while the reside dependency to Utility is still drawn in the same manner as in Figures 5-5 and 5-6.

Notice that the Utility package resides in all the components in Figures 5-5, 5-6, and 5-7, because each component described in those figures has a package that uses the Utility package. Details of the Utility package are discussed in Chapter 3.

**Use Dependencies**

A *use* dependency from a client component to a supplier component indicates that the client component uses or depends on the supplier component. A use dependency
from a client component to a supplier component’s interface indicates that the client component uses or depends on the interface provided by the supplier component. A use dependency is shown as a dashed arrow from a client component to a supplier component or a supplier component’s interface. The dependency may be marked with the use keyword; however, the keyword is often omitted because this is the default, and the meaning is evident from how the dependency is used.

Figure 5-8 shows how the various components of the project management system are related:

The User Interface component
  Uses the Security component and the IBusiness Processing interface provided by the Business Processing component

The Business Processing component
  Uses the Security component and the IProducible and IConsumable interfaces provided by the Data component

The Data component
  Uses the Security component

**Deploy Dependencies**

A deploy dependency from a client component to a supplier node indicates that the client component is deployed on the supplier node.

A deploy dependency is shown as a dashed arrow from a client component to a supplier node marked with the deploy keyword. Figure 5-9 shows that the User Interface component is deployed on the Desktop Client node.

Figure 5-10 shows that the Business Processing component is deployed on the Business-Processing Server node.
Alternatively, a component that is deployed on a node may be shown nested inside the node. Figure 5-11 shows that the Data component is deployed on the Database Server node.
Communication Associations

Figure 5-3 shows nodes associated with the project management system, but how are those nodes related? A specialized type of association, called a communication association, addresses the question of how nodes are related. (Associations are discussed in Chapter 3.)

A communication association between nodes indicates a communication path between the nodes that allows components on the nodes to communicate with one another. A communication association is shown as a solid line between nodes. Figure 5-12 shows that the Business-Processing Server has a communication association with the Desktop Client, Printer, and Database Server nodes.

Figure 5-13 combines Figure 5-8 and Figure 5-12 to show how components are related to nodes. Notice that if two components are related and reside on different nodes, the nodes must have a communication association between them to allow the components to communicate; otherwise, the components are not able to communicate and be related to one another. For example, if the communication association between the Desktop Client and Business-Processing Server nodes was removed, the User Interface component could not be related to the IBusiness Processing interface and Security component. If the communication association between the Business-Processing Server and Database Server nodes was removed, the Data component could not be related to the Security component, and the Business Processing component could not be related to the IProducible and IConsumable interfaces.
Exercises

1. Describe Figure 5-14: identify components and nodes, and describe the relationships among components and nodes.

2. Describe Figure 5-15: identify the various elements and their relationships. Update the diagram stepwise to show the following details. After each step, check your answers against the solutions shown in Appendix B:
   a. The User Interface package uses the IView and IPrint interfaces provided by the Reporting subsystem.
   b. The User Interface and Utility packages resides in a User Interface component.
   c. The Reporting subsystem and Utility package reside in a Reporting component.
   d. The User Interface component is deployed on a Desktop Client node.
   e. The Reporting component is deployed on a Report Server node.
   f. The Desktop Client node is connected to the Report Server node, and the Report Server node is connected to a High-speed Printer node.
Figure 5-14. Components and nodes for the project management system

Figure 5-15. Packages and a subsystem