Physical Architecture

As stated earlier, architecture can be defined at both a logical and physical level. You will recall that the logical architecture is the more generic view of the architecture and how it works with less technical specificity.

The physical architecture, on the other hand, describes in more detail how the software and systems are designed, including specifics about how the architecture must fit into different technologies that exist within the organization and how the software integrates with itself and with other systems. We use several modeling elements and techniques to describe the physical architecture.

Operations

Operations specify the business logic concerning how a class functions and how classes can interact with each other. For example, getCustomer could be an operation on the Customer class. This provides the application logic for how to query for customer information (see Figure 4-11).

Although all parts of the class can be translated to the physical code and become part of a running application, what causes the operations to differ is that they are much more specifically defined when you get into the physical implementation of the system and model. The operations are used in a logical architecture model to show expected software behavior but are more physically defined to show algorithms, functions, and more. They describe how the system will function and will be implemented with respect to the technology to be used.

We will revisit classes, attributes, operations, and so forth in Chapter 5 when we discuss application modeling.

Component Diagrams

Components are made up of one or more classes and describe parts of an application that can be assembled and reused. A component-based architecture (CBD) is the design of a software system made up of multiple components. For efficiency, it is important to develop software based on multiple smaller parts (components), which you can use to assemble the overall system. This enables you to reuse software components instead of writing them all from scratch.

Component-based architecture also enables different teams to work on the software and plug their pieces together using what are called “interfaces.” An interface is a named set of operations that enable the components to work together through this interface code. Interfaces enable you to get and provide information to and from a component using code specific to the technology on which it is deployed. Figure 4-12 shows how a component and interface are displayed on a component diagram. The interface is displayed as a circle, but it can also be displayed as a typical class with the stereotype of <<interface>>. A component diagram can display multiple components and can show how those components integrate together, as seen in Figure 4-13.

Figure 4-12. Component with interface.

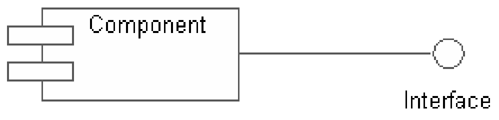
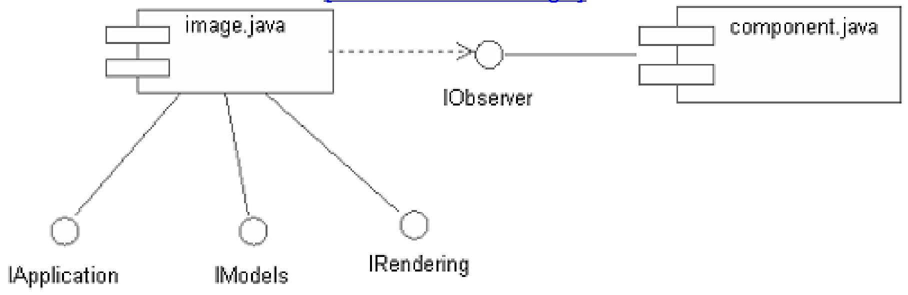


Figure 4-13. Component diagram.

[View full size image]

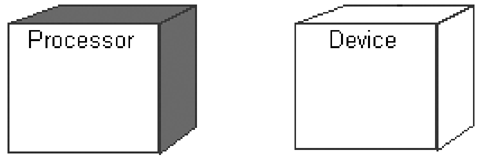


You can use a component diagram and components to model the architecture of different parts of your system. You can also use them to model the application internals, as we just discussed, as well as to view how different applications work together. For example, consider two executables (.exe), one that starts the other, both modeled as components. When writing Java applications, each Java file is represented in the UML as a component as well. This demonstrates that different levels of the architecture can be (and usually are) modeled as components. As classes represent the logical architecture of code, components represent the physical architecture and identify what has actually been implemented.

Deployment Diagrams

Deployment diagrams represent the runtime architecture of your system. A deployment diagram can be made up of nodes that represent a piece of hardware that generally has memory and a processor built in. There are two different types of nodes: processor and device. Figure 4-14 shows what the deployment diagram elements look like.

Figure 4-14. Deployment diagram elements.



Sometimes, a deployment diagram can show multiple applications running on a single device but on different processors. A server, for example, is a device that can have multiple processors running on it. A deployment diagram of a server would show that multiple applications could be running on a single server but on different processors.

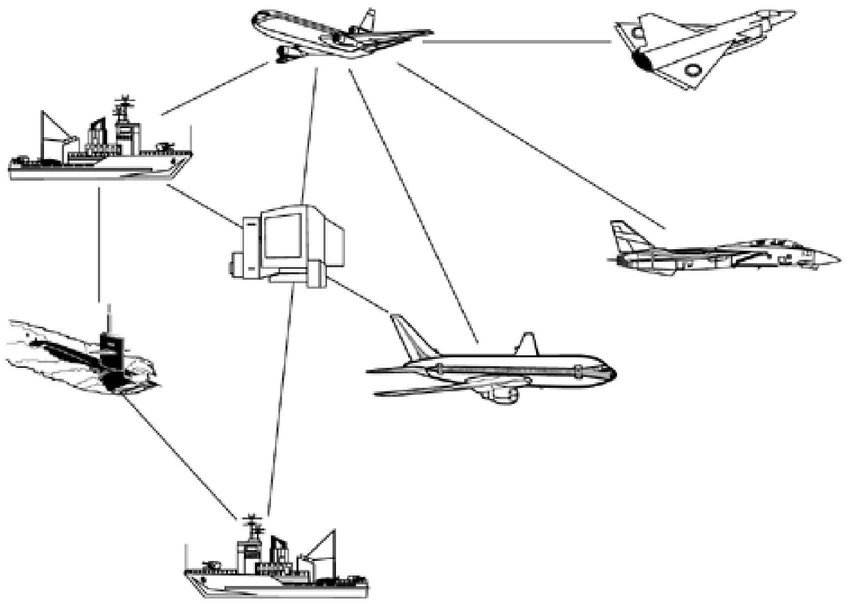
When trying to understand the information architecture of your organization, it is important to know not only what software exists and how it is architected, but also how the architecture of the systems reside on the hardware. This means knowing what hardware exists, what software is on what hardware, what servers are used for backup, and how the software is stored on multiple pieces of hardware.

An enterprise architecture looks at the overall architecture and how all the pieces fit together. Many organizations, when defining their enterprise architectures, include not just hardware one would typically think of, such as servers, but also other pieces of hardware, such as aircraft, that are part of the enterprise. For instance, the DoDAF, which is used as the framework for the United States Department of Defense enterprise architecture, consists of visualizing the architecture of the different components that make up the information network, and they don't just include what you traditionally think of as software systems, but also airplanes, telecommunication systems, satellites, ships, and more because all these systems are run and managed by software, but often, one wouldn't consider such software in terms of applications. However, when the DoD defines their enterprise architecture, it is very important for them to understand where these applications fit in. Just as the military needs to understand how all facets of its air defense system work together, businesses need to understand how all of their systems work together.

Stereotypes

You can use stereotypes in all UML diagrams to better define the elements that you are modeling so that everyone who views the models understands what is being modeled and the story that is being told. Standard stereotypes are defined in the UML. We discussed some of them earlier in this chapter, and we will cover more later in this book. A stereotype can be just a title, but it can also carry a picture with it to better define itself and to provide an easy-to-understand graphical view the UML model. A few stereotypes we have seen used in deployment diagrams display the different types of deployment elements, such as different types of airplanes and ships (see Figure 4-15).

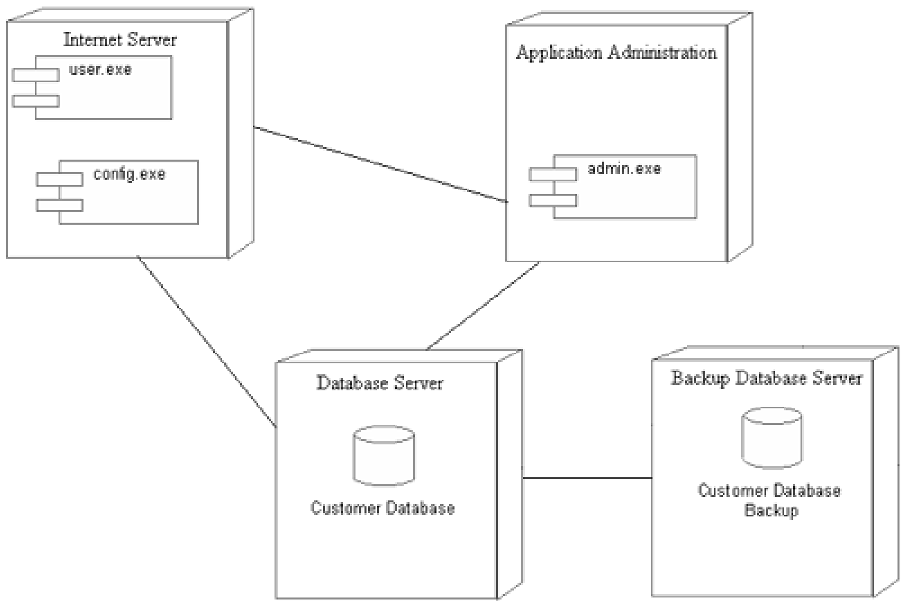
Figure 4-15. Deployment diagram displaying pictures for stereotypes.



As discussed earlier, you also can use a deployment diagram in a more traditional sense: to understand the hardware and applications in your organization and how they exist within the enterprise. Figure 4-16 demonstrates how this can be achieved.

Figure 4-16. Application deployment diagram.

[View full size image]


Internet Server
user.exe
config.exe
Application Administration
admin.exe
Database Server
Customer Database
Backup Database Server
Customer Database Backup