contained parts. However, remember that these contained parts may require services from the environment of the `EnvironmentalControlSystem` component, such as a gardening plan to meet this contract.

To be specific, `:EnvironmentalController` requires `GardeningPlan`, which specifies the environmental needs (lighting, heating, and cooling) of the Hydroponics Gardening System. The needs of this required interface are delegated to an unnamed port, to which is attached the `GardeningPlan` interface. In this manner, we know that we must provide the `EnvironmentalControlSystem` component with a gardening plan if we intend to use its services. We also recognize that we must provide it with `AmbientLight`, `AmbientTemp`, and `TempRamp` services.

The connectors of `EnvironmentalControlSystem` provide its communication links to its environment, as well as the means for its parts to communicate internally. In Figure 5–16, the type of connectors new to our view of `Environmental ControlSystem` are the *delegation connectors* to which we've alluded. Through these connectors, the responsibilities (provided interfaces) of `EnvironmentalControlSystem`, as well as its requirements (required interfaces), are communicated. For example, the `:LightingController` component opaquely provides the `LightingControl` services. A user of `EnvironmentalControlSystem` would not likely have this white-box perspective of the subsystem [91, 92].

5.4 Deployment Diagrams

A deployment diagram is used to show the allocation of artifacts to nodes in the physical design of a system. A single deployment diagram represents a view into the artifact structure of a system. During development, we use deployment diagrams to indicate the physical collection of nodes that serve as the platform for execution of our system.

The three essential elements of a deployment diagram are artifacts, nodes, and their connections.

Essentials: The Artifact Notation

An artifact is a physical item that implements a portion of the software design. It is typically software code (executable) but could also be a source file, a document, or another item related to the software code. Artifacts may have relationships with other artifacts, such as a dependency or a composition [20, 21].

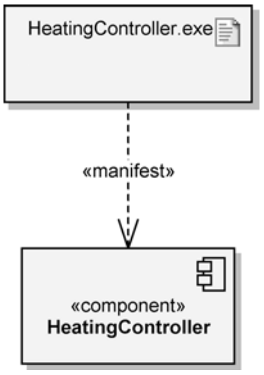

Figure 5–17 The Artifact Notation for `HeatingController.exe`

The notation for an artifact consists of a class rectangle containing the name of the artifact, the keyword label «`artifact`», and an optional icon that looks like a sheet of paper with the top right-hand corner folded over. Figure 5–17 shows the `HeatingController.exe` artifact, without the optional icon.

The name of this artifact includes the extension `.exe`, indicating that it is an executable (i.e., software code). The `HeatingController.exe` artifact has a dependency relationship to the `HeatingController` component, labeled with «`manifest`». This means that it physically implements the component, thereby connecting the implementation to the design. An artifact may manifest more than one component [22].

Essentials: The Node Notation

A node is a computational resource, typically containing memory and processing, on which artifacts are deployed for execution. Nodes may contain other nodes to represent complex execution capability; this is shown by nesting or using a composition relationship. There are two types of nodes: devices and execution environments [23, 24].

A device is a piece of hardware that provides computational capabilities, such as a computer, a modem, or a sensor. An execution environment is software that provides for the deployment of specific types of executing artifacts; examples include «`database`» and «`J2EE server`». Execution environments are typically hosted by a device [25].

Figure 5–18 shows the three-dimensional cube icon that we use to represent a node, in this case, the `PC` and `ApplicationServer` nodes. The icon may be adorned with a symbol to provide additional visual specification of the node type. There are no particular constraints on node names because they denote hardware, not software, entities.

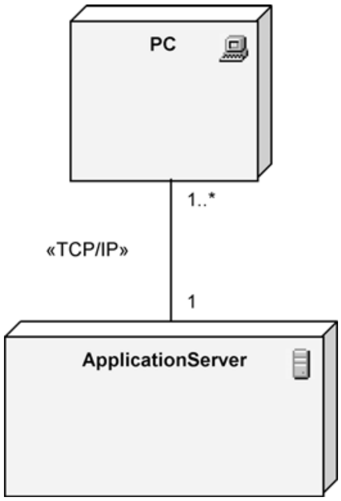

Figure 5–18 Notations for Two Nodes

Nodes communicate with one another, via messages and signals, through a communication path indicated by a solid line. Communication paths are usually considered to be bidirectional, although if a particular connection is unidirectional, an arrow may be added to show the direction. Each communication path may include an optional keyword label, such as `«http»` or `«TCP/IP»`, that provides information about the connection. We may also specify multiplicity for each of the nodes connected via a communication path.

In Figure 5–18, the communication between the `PC` and `ApplicationServer` nodes is bidirectional. A communication path usually represents some direct hardware coupling, such as a USB cable, an Ethernet connection, or even a path to shared memory. However, the path could also represent more indirect couplings, such as satellite-to-ground or mobile phone communications. In our case, it represents a bidirectional connection using TCP/IP protocols. We've specified the connection of one or more `PC` nodes to one `ApplicationServer` node.

Essentials: The Deployment Diagram

In Figure 5–19, we provide an example of a deployment diagram drawn primarily from the physical architecture of the Environmental Control System within the Hydroponics Gardening System. Here we see that our system architects have decided to decompose this portion of our system into a network of two nodes (`PC` and `ApplicationServer`) and two devices (`LightMeter` and `Thermometer`). If you compare this deployment diagram to the component diagram of `EnvironmentalControlSystem` (presented earlier in Figure 5–11), you will see that it does not account for all of its interfaces; we omitted some to somewhat simplify our example. In addition, we show each artifact implementing exactly one component.

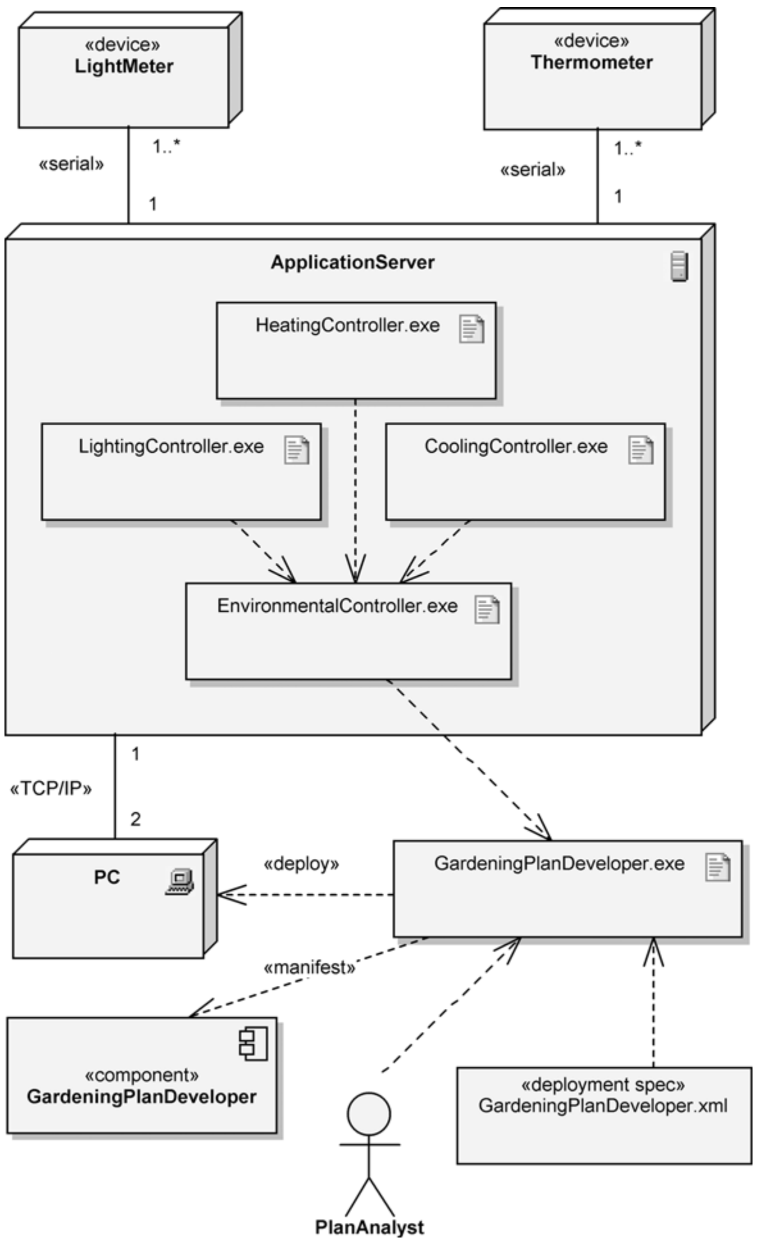

Figure 5–19 The Deployment Diagram for `EnvironmentalControlSystem`

The deployment of the EnvironmentalController.exe, LightingController.exe, HeatingController.exe, and CoolingController.exe artifacts on the ApplicationServer node is indicated by containment. Another way to denote deployment is shown by the dependency from the GardeningPlanDeveloper.exe artifact to the PC node labeled with «deploy». A third way to denote deployment is through textually listing the artifacts within the node icon; this is especially useful for larger or more complex deployment diagrams [26, 27].

We have three unnamed dependencies within Figure 5–19 between artifacts: from the LightingController.exe, HeatingController.exe, and CoolingController.exe artifacts to the EnvironmentalController.exe artifact. These denote the dependencies between the components that they implement, rather than deployment onto a node.

We also have another dependency, from the EnvironmentalController.exe artifact to the GardeningPlanDeveloper.exe artifact. This relates back to the interface on the EnvironmentalController component, which requires a gardening plan. Here we see that the gardening plan will be developed by PlanAnalyst using the GardeningPlanDeveloper.exe artifact, which manifests the GardeningPlanDeveloper component. Note that PlanAnalyst may perform this task from either of two PC nodes.

The two devices, LightMeter and Thermometer, provide the ambient light and ambient temperature sensor readings required by the EnvironmentalController.exe artifact in support of its provision of functionality to the system. One item we have yet to discuss is the GardeningPlanDeveloper.xml deployment specification, which has a dependency relationship to the GardeningPlanDeveloper.exe artifact. This deployment specification describes deployment properties of the artifact, such as its execution and transaction specifics [28].

5.5 Use Case Diagrams

Evidence over the years has shown that some of the most common reasons software projects fail center around poor or nonexistent communication between the key stakeholders. This is particularly critical when there is lack of alignment between the development organization and the line of business. The business people may know that they have a certain problem that needs to be solved, but the development organization may receive only a general description of what the