# 20

# THE DEPLOYMENT VIEWPOINT

| | |
|---|---|
| **Definition** | Describes the environment into which the system will be deployed, including the dependencies the system has on its runtime environment |
| **Concerns** | Types of hardware required, specification and quantity of hardware required, third-party software requirements, technology compatibility, network requirements, network capacity required, and physical constraints |
| **Models** | Runtime platform models, network models, and technology dependency models |
| **Problems and Pitfalls** | Unclear or inaccurate dependencies, unproven technology, lack of specialist technical knowledge, and late consideration of the deployment environment |
| **Stakeholders** | System administrators, developers, testers, communicators, and assessors |
| **Applicability** | Systems with complex or unfamiliar deployment environments |

The Deployment view focuses on aspects of the system that are important after the system has been tested and is ready to go into live operation. This view defines the physical environment in which the system is intended to run, including the hardware environment your system needs (e.g., processing nodes, network interconnections, and disk storage facilities), the technical environment requirements for each node (or node type) in the system, and the mapping of your software elements to the runtime environment that will execute them.

The Deployment viewpoint applies to any information system with a required deployment environment that is not immediately obvious to all of the interested stakeholders. This includes the following scenarios:

- Systems with complex runtime dependencies (e.g., particular third-party software packages are needed to support the system)
- Systems with complex runtime environments (e.g., elements are distributed over a number of machines)

- Situations where the system may be deployed into a number of different environments and the essential characteristics of the required environments need to be clearly illustrated (which is typically the case with packaged software products)
- Systems that need specialist or unfamiliar hardware or software in order to run

In our experience, most large information systems fall into one of these groups, so you will almost always need to create a Deployment view.

# CONCERNS

## Types of Hardware Required

The Deployment view must clearly identify the types of hardware the system needs and the role each type plays. This includes general-purpose hardware to execute the main functional elements of the system, storage hardware to support databases, hardware that allows users to access the system, network elements required to meet certain quality properties (such as firewalls for security), specialist hardware (such as cryptographic accelerators), and so on.

This involves identifying the general types of hardware required and mapping each of your functional elements to one of the hardware types. In effect, this is a logical model of the hardware your system requires. Then, when you have defined what each piece of hardware does, you can think about the details of exactly what hardware elements you need.

## Specification and Quantity of Hardware Required

This concern, which follows from the previous one, addresses the specific details of the hardware that will need to be procured and commissioned in order to deploy the system—in effect, a physical model of the hardware your system needs.

This is a separate concern from the previous one because it is much more specific and of interest to different stakeholders. For example, developers are interested in whether the deployment platform will use Intel or Sun SPARC servers, whether the servers will run Linux or Sun Solaris, and what general processing resources will be available to them. However, system administrators are interested in the details of the specific machines that need to be procured and commissioned in order to create your runtime environment.

Be specific when considering the specification and quantity of hardware you need. If specific models of equipment are required, you need to clearly identify them and record their specifications for easy reference. If specific models aren't required, you should still be precise where needed.

## Third-Party Software Requirements

All information systems make use of third-party software as part of their deployment environment—even if only an operating system. Many information systems make use of dozens of third-party software products, including operating systems, programming libraries, messaging systems, application servers, databases, data movement products, Web servers, and so on. Although these third-party products are extremely valuable, using them incurs additional complexity that needs to be managed.

Your Deployment view should make clear all of the dependencies between your system and any third-party software products. This ensures that the developers know what software will be available for them to use and that the system administrators know exactly what needs to be installed and managed on each piece of hardware.

## Technology Compatibility

Each software and hardware element in your system may impose requirements on other technology elements. For example, a database interface library may require a particular operating system network library in order to function correctly, or a disk drive may require a particular type of controller in the machines that will access it.

Furthermore, if you use a number of pieces of third-party technology together, there is always the danger of uncovering incompatible requirements. For example, your database interface library may require a certain version of the operating system, while a graphics library you want to use isn't supported on that version. Such incompatibilities have a habit of emerging late in the testing cycle and causing a lot of disruption—so if you consider them early, you will avoid problems later.

## Network Requirements

Your Functional and Concurrency views define the functional structure of your architecture and make it clear how its elements interact. Part of the process of

creating the Deployment view is to decide which hardware elements host each of these functional elements. Because elements that need to communicate often end up on different machines, some of the interelement interactions can be identified as network interactions.

One of the concerns the Deployment view addresses is the set of requirements the system places on its underlying network as a result of these network interactions. This view needs to clearly identify the required links between machines, the required capacity and reliability of the links, and any special network functions the system requires (load balancing, firewalls, encryption, and so on).

## Network Capacity Required

In our experience, software architects need to get less involved in specifying network capacity than in identifying the processing and storage hardware because the network is normally provided by a group of specialists who design, implement, and operate the network for an entire organization. However, this group needs to know how much network capacity your system requires and the type of traffic you need to carry over the network. In order to provide this information, you must calculate and record the amount and type of network traffic that needs to be carried over each intermachine link in the proposed network topology.

## Physical Constraints

Software architects and software engineers are lucky when compared to our colleagues working in other engineering disciplines. Normally, we don't have to worry that much about physical constraints because software has no weight, has no physical size, requires no storage space, and so on. However, when taking a system-level view, physical constraints suddenly become important again.

Considerations such as desk space for client workstations, floor space for servers, power, temperature control, cabling distances, and so on may seem relatively mundane. However, if someone doesn't consider them, your system simply won't be deployed. There is no point in specifying four monitors for each workstation in order to create an incredibly powerful user interface if your users have desk space for only two. Similarly, if there isn't enough floor space in your data center for your servers, they aren't going to get installed.

## Stakeholder Concerns

Typical stakeholder concerns for the Deployment viewpoint include those shown in Table 20–1.

**TABLE 20–1** STAKEHOLDER CONCERNS FOR THE DEPLOYMENT VIEWPOINT

| Stakeholder Class | Concerns |
|---|---|
| System administrators | Types, specification, and quantity of hardware required; third-party software requirements; technology compatibility; network requirements; network capacity required; and physical constraints |
| Developers | Types and (general) specification of hardware required, third-party software requirements, technology compatibility, and network requirements (particularly topology) |
| Communicators | Types and specification of hardware required, third-party software requirements, and network requirements (particularly topology) |
| Testers | Types, specification, and quantity of hardware required; third-party software requirements, and network requirements |
| Assessors | Types of hardware required, technology compatibility, and network requirements |

# MODELS

## Runtime Platform Models

The runtime platform model is the core of this viewpoint. This description defines the set of hardware nodes that are required, which nodes need to be connected to which other nodes via network (or other) interfaces, and which software elements are hosted on which hardware nodes.

A runtime platform model has the following main elements.

- *Processing nodes*: Each computer in your system is represented by one processing node in the runtime platform model. This allows you and other stakeholders to see what processing resources are required for the system. For situations where many similar machines are required (e.g., Web server farms), you can use a summary notation (such as UML's shadow notation) to simplify the diagram, but make sure that the number of nodes required is still clear.
- *Client nodes*: You also need to represent client hardware, but probably in less detail than the main processing hardware. You may have less control over client hardware than server hardware, and if this is the case, you need only represent the types and quantities of client machines required rather than the precise details of each. If you have special needs for presentation or user interaction hardware (e.g., touch screens, printers), this is specified as part of the client hardware.

- *Online storage hardware*: This defines how much storage is needed, how it is partitioned, what it is used for, and whether or not processing takes place close to its associated stored data. The storage hardware could be disk devices within a processing node or dedicated storage nodes such as disk arrays. Make the distinction between the two types clear so that the physical impact of separate storage nodes on the deployment environment is understood. You need to include the capacity (and possibly speed) of each type of storage hardware in the model.
- *Offline storage hardware*: Despite the ever-growing capacity of online storage hardware, many systems that deal with a lot of information still require offline storage (archives) as well. Somehow the problems always grow faster than the hardware capacity. Offline storage will also probably be required to allow backup of information held online. You need to ensure that there is sufficient capacity, that the hardware is fast enough to complete archive and retrieval in an acceptable time, and that there is sufficient network bandwidth between it and the online storage. The type, capacity, speed, and location of your offline storage hardware all need to be included in your models.
- *Network links*: Your model needs to capture the essential connections required by your system (rather than your ideas on how the network will be built from specific network elements). It is sufficient at this point to show the links between your hardware nodes; you'll capture more details about the network, such as internode bandwidth requirements, in the network model (described next in this chapter).
- *Other hardware components*: You may need to consider specialist hardware for network security, user authentication, special interfacing to other systems, or specialist processing (e.g., for automated teller machines).
- *Runtime element-to-node mapping*: The final element of this model is a mapping of the system's functional elements to the processing nodes where they execute. How to go about defining this mapping depends on how complex your concurrency structure is. If you have a Concurrency view, you can map the operating system processes identified in that view to the processing nodes. If you don't have a Concurrency view, you can map functional elements from the Functional view directly to processing nodes (and in this case, presumably the details of the operating system processes in use aren't architecturally significant).

This runtime platform model is typically captured as a network node diagram that shows nodes, storage, the interconnections required between the nodes, and the allocation of the software elements between the nodes.

**NOTATION** Common notations used for capturing the runtime platform model include the use of UML, traditional boxes-and-lines diagrams, and textual notations. Each of these options is outlined in this subsection.

- *UML deployment diagram*: You can use a UML deployment diagram to document a runtime platform model. This diagram shows computing nodes with software elements inside them and associations between the nodes representing the required communication links. Associations between the elements can also be used to indicate interelement dependencies. Figure 20–1 shows an example of a simple runtime platform model that maps functional elements straight to processing nodes and is represented as a UML deployment diagram.

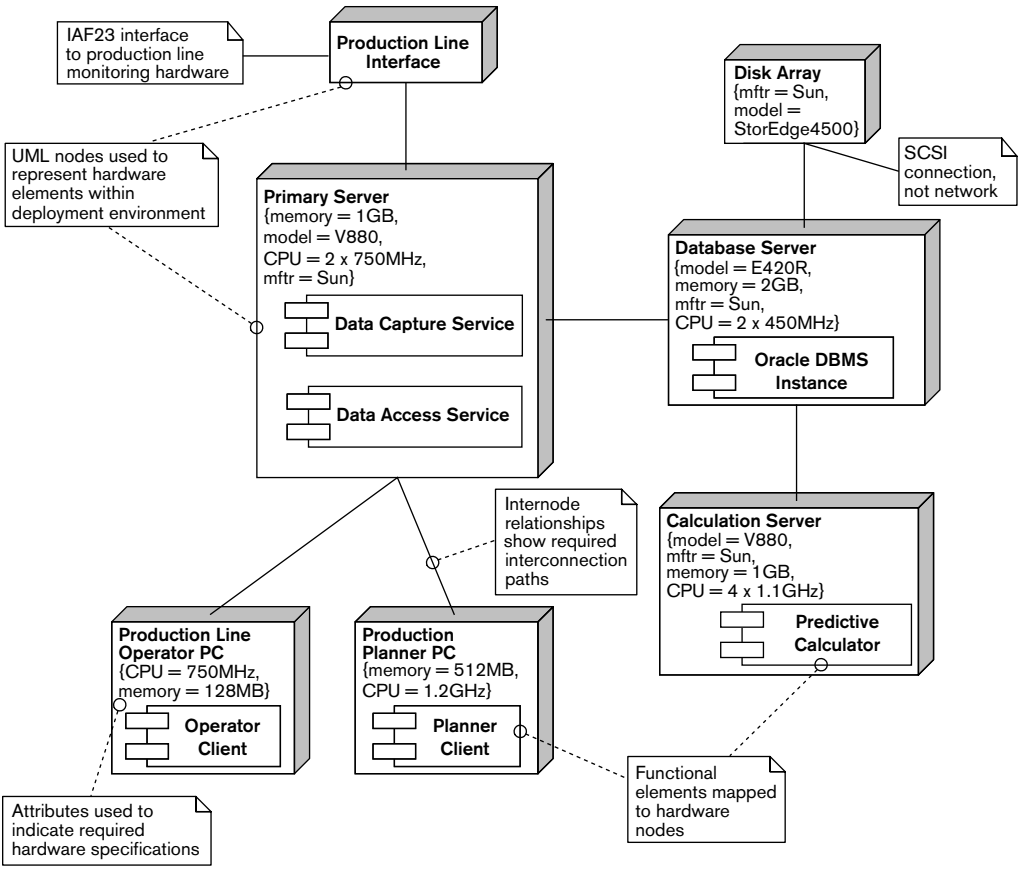


**FIGURE 20–1** EXAMPLE OF A RUNTIME PLATFORM MODEL

UML does not associate detailed semantics with the nodes or associations and does not provide different types of each. Therefore, effective use of this diagram type normally relies heavily on stereotypes, tagged values, and comments in order to distinguish between different types of nodes and links. A runtime platform model such as this needs to be augmented by plain text descriptions of each of the major elements, clearly defining their important attributes.

- *Boxes-and-lines diagram*: Given the lack of precision in the basic UML deployment diagram, many architects choose a simple boxes-and-lines notation for Deployment views. Boxes are used to represent nodes and elements, with arrows for interconnection, annotating this as required in order to make the meaning of each diagram element clear. With such an approach, you need to carefully define the diagrammatic elements used to avoid causing any confusion for the reader. This notation is easier to draw with drawing tools that don't support UML, and it is more comprehensible to nontechnical stakeholders.
- *Text and tables*: Reference information such as required hardware specifications is best represented by text that is organized into tables for easy, unambiguous reference.

### ACTIVITIES

**Design the Deployment Environment**. You typically start by identifying the key servers in the system and the key client hardware (assuming that you need to supply or specify the clients), and then you identify the network links between these nodes. With this done, you have the backbone of your deployment environment. The rest of the process is normally elaboration, adding special-purpose hardware (e.g., disk storage, cryptographic accelerators, or nodes for redundant capacity) as required and specifying the hardware specification and software requirements for each node along with the required specification of any interconnections.

**Map the Elements to the Hardware**. Once you have a proposed deployment environment, you need to find a home in it for each of your functional (software) elements. In reality, this is an iterative process where mapping the software elements to hardware resources may suggest changes in the deployment environment design (or newly identified deployment environment options may suggest new alternatives for software element locations). The main challenges here relate to managing dependencies, ensuring enough machine capacity is available, and trading off the advantages of separated versus colocated elements (e.g., security versus performance). Refer to Chapters 24 and 25 for more depth on these topics.

**Estimate the Hardware Requirements**. This activity normally starts with some initial estimation before initial deployment environment design, followed by an iterative process of refinement as architecture and design progresses.

The resources you need to estimate include processing power, memory, disk space, and I/O bandwidth for each processing node.

**Conduct a Technical Evaluation**. In order to design and estimate the deployment environment, you may need to perform a number of technical evaluation exercises such as prototype element development, benchmarks, and compatibility tests. For example, you may wish to create a representative prototype system to ensure that your application server, object persistence library, and database all work smoothly together and to check the transaction throughput you can achieve.

To ensure a representative test, identify the key attributes of your application (size, type of processing, and so on), and make sure you include all of this in your technical evaluation.

Obtaining time and resources for technical evaluation is often a problem. We have found that arguing for evaluation resources in terms of risk management is often the most effective way to deal with this.

**Assess the Constraints**. It is rare for architects to be left to define a Deployment view without any external constraints. The constraints you encounter may be formal standards, informal guidelines, or simply implicit constraints that you know exist. However the constraints are expressed, you need to review your proposed deployment environment design to ensure they are met.

## Network Models

In the interests of simplicity, the runtime platform description does not typically illustrate the network in any detail. If the underlying network is complex, it is usually described in a separate (but related) network model.

In our experience, the network is usually designed and implemented by networking specialists rather than the software architect. However, it is important that you provide the networking specialists with a clear specification of the network you are expecting. This description must indicate which nodes need to be connected, any specific network hardware (such as firewalls or routers) you are expecting to be present, and the bandwidth requirements and quality properties required from each part of the network. As such, this model is normally a logical or service-based view of what you require of the network, rather than a physical view that specifies its individual elements. In the case of software product development, such a model is a valuable specification for customers planning the deployment of your software.

The primary elements of a network model are as follows.

- *Processing nodes*: The processing nodes represent your system elements that use the network to transport data. This set of nodes should match the set from the runtime platform model, but here they are abstracted to simple elements that have just network interfaces.

- *Network nodes*: In most cases, the network nodes indicate a type of network service that you expect to be available (such as firewall security, load balancing, or encryption).
- *Network connections*: The network connections are the links between the network and processing nodes. They are elaborated to include the characteristics of the service you expect the link to provide (most typically bandwidth, but perhaps quality of service or other network services).

This description is typically represented as an annotated network diagram, which can really be thought of as a network-oriented specialization of the runtime environment diagram. In cases where your network requirements are very simple, you can describe the network sufficiently by elaborating on the runtime platform model, rather than creating a separate network model. However, given the critical dependency that most of today's systems have on the underlying network, a separate network model is a useful tool to focus attention on this aspect of the system.

Figure 20–2 shows a simple example of a network model for the runtime platform we depicted earlier in Figure 20–1. This diagram would be augmented with textual descriptions for each of the major elements.

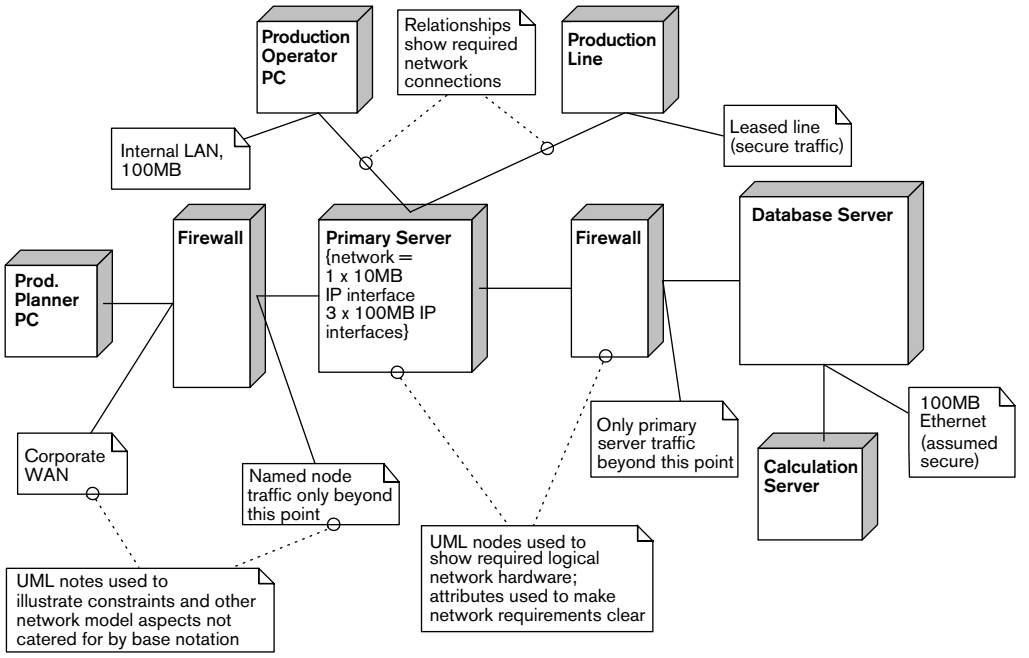


**FIGURE 20–2** EXAMPLE OF A NETWORK MODEL

**NOTATION** Common notations used for capturing the network model include the use of UML and traditional boxes-and-lines diagrams. These options are briefly discussed in this subsection.

- *UML deployment diagram*: UML's deployment diagram is a useful base notation for a network model. However, as with the runtime platform description, you will probably need to heavily annotate it to make your intentions clear.
- *Boxes-and-lines diagram*: For reasons similar to those discussed earlier, the network model is often drawn using an informal notation.

### ACTIVITIES

**Design the Network**. The network design is typically handled separately from that for the computer hardware because different specialists are involved. From your point of view, this is a process of sketching what you need from the network (in terms of connections, capacity, quality of service, and security). This results in what is effectively a logical rather than physical network design. Your logical network design then becomes a specification for a specialist network designer to take further.

**Estimate the Capacity**. Part of designing your logical network is to estimate the capacity you need between each node. Precision isn't that important at this stage, but a realistic estimation of the magnitude of the traffic that must be carried is important. You can estimate the capacity figures by combining peak transaction throughput and a rough approximation of the size of messages required to carry the transaction's information; the result is normally combined with a judicious scaling factor to allow for inevitable overheads and prediction inaccuracies.

## Technology Dependency Models

In some cases, you may choose to manage the dependencies within your development or test environment by bundling your elements and the software they rely on into one deployment unit. However, in many cases this simply won't be possible for reasons such as efficiency, cost, licensing, or flexibility. If this is the case, you need to manage the dependencies in your deployment environment.

Technology dependencies are usually captured on a node-by-node basis in simple tabular form. The software dependencies are typically derived from the Development view, where you define the environment used by the software developers. You can also derive hardware dependencies from test or development environments, but in many cases you have to rely on manufacturer specifications and some judicious testing to confirm them.

**TABLE 20–2** SOFTWARE DEPENDENCIES FOR THE PRIMARY SERVER NODE

| Component | Requires |
|---|---|
| Data Access Service | Solaris 8.0.2<br>Sun C++ 4.1.2 libraries |
| Data Capture Service | Solaris 8.0.2 |
| | Sun C++ 4.1.2 libraries |
| | Oracle OCI libraries 8.1.7.3 |
| Sun C++ 4.1.2 | Solaris patch 1534567 |
| | Solaris patch 1538367 |
| Oracle OCI 8.1.7.3 | Solaris optional module SUNWcipx |
| | Solaris patch 1583956 |

**EXAMPLE** Table 20–2 shows an example of software dependencies for the Primary Server node in our example from Figure 20–1.

From this table, it is possible to see that this node in the system needs a particular version of Solaris with three patches and one optional module installed, as well as a particular version of an Oracle product and a particular version of a set of language libraries.

In simple cases, it may be possible to use the Development view contents rather than listing dependencies in this view. However, in more complex cases, it is unlikely that the Development view contains the detail required to fully define the software dependencies for each node in the system.

**NOTATION** A technology dependency model is often best captured by using a simple text-based approach, but it can sometimes benefit from the use of some simple graphical notations. Both of these options are briefly discussed in this subsection.

- *Graphical notations*: One way to capture software dependencies is to extend your runtime platform model to add an indication of the software stack required on each machine to support the system elements executing there. In simple cases, this can be a useful elaboration of the runtime platform model. The problem with this is that complete and accurate software dependency stacks on each node can clutter the runtime platform

model to the point where it is no longer usable—in this case, you should record this information separately.

- *Text and tables*: Dependencies are almost always captured as simple text tables. It is important to capture the exact requirements for third-party software (e.g., detailed version numbers, option names, and patch levels).

### ACTIVITIES

**Analyze the Runtime Dependency**. This is usually a manual exercise to work through your system elements, identifying the dependencies they have and then repeating this process for each of the third-party elements. You normally derive the runtime dependencies from documentation supplied with each piece of third-party technology you are using and your own build and test environment requirements. With this done, you can clearly define the third-party elements you need for each processing node in the system.

**Conduct a Technical Evaluation**. In order to correctly document dependencies, you may need to do some prototyping or technical investigation.

## Intermodel Relationships

For complex systems, a Deployment view contains two or three closely related models rather than a single model. We have found that the three models described earlier tend to be used by different stakeholders at different times. Senior staff in the groups responsible for deployment refer to the runtime platform model early in the project, a specialist networking group consults the network model, and system administrators use the technology dependency model during more detailed installation planning close to deployment. For this reason, we've found it valuable to present each separately.

A good way to consider these models is as a set of informal layers, with the core of the view being the runtime platform model. You can think of the network model as a lower-level layer supporting the runtime platform by elaborating on the details of the network required. Think of the technology dependency model as a more detailed layer on top of the runtime platform that defines the software and hardware installation requirements on each machine in the deployment environment.

In an ideal world, a software architecture tool would allow you to create a single model for yourself and then extract different aspects of it automatically as required. However, we aren't aware of any such tool in the commercial area.

Figure 20–3 illustrates this relationship between the models within the Deployment view. The runtime platform model is the core of the view, with the network model providing more details of the network underpinning the system and the technology dependency model providing more detail about the hardware and software installed on each node to provide the runtime environment.

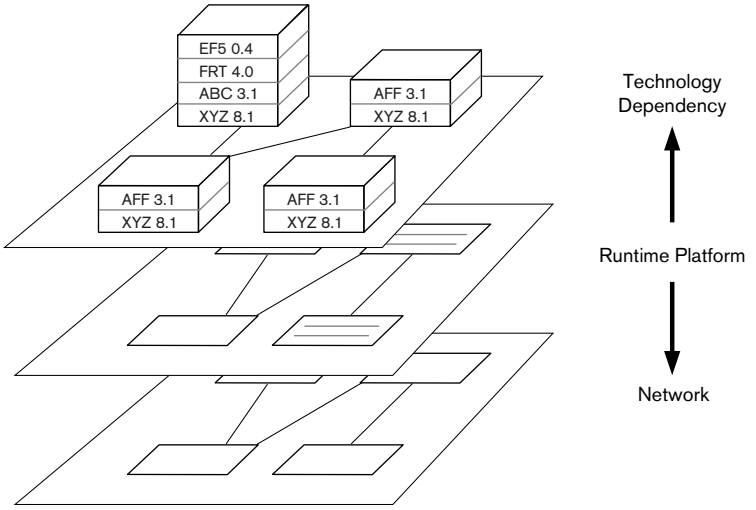


**FIGURE 20–3** MODELS IN THE DEPLOYMENT VIEW

# PROBLEMS AND PITFALLS

## Unclear or Inaccurate Dependencies

Large-scale computing technology tends to be fairly complex, and it often has many explicit and implicit dependencies on its runtime environment that, if not met, cause runtime problems. This difficulty is compounded by the fact that most of these dependencies are invisible and can't be checked easily—you may not discover that you have the wrong version of a utility library until your database server fails to start.

"You need Oracle and Solaris" or "It uses SPARC hardware" are pretty common dependency statements. For all but the smallest systems, these are too vague to allow safe deployment of the system. You should specify which versions are required, whether any optional parts of the products are needed, whether any patches are required, and so on. With the complexity and flexibility of enterprise software products today, you need to be very clear what is required and what isn't.

### RISK REDUCTION

- Capture clear, accurate, detailed dependencies between your software elements and the runtime environment in the Deployment view.
- Capture dependencies between third-party software and the runtime environment it needs.

- Perform compatibility testing to ensure that the dependencies between the elements are correct.
- Use existing, proven combinations of technologies where the dependencies are well understood.

## Unproven Technology

Everyone wants to use the newest and coolest technology—and understandably so, as it often has the potential to bring great benefits. However, because its characteristics are unknown, using technology that you don't have experience with brings significant risks: functional shortcomings, for example, or inadequate performance, availability, or security.

### RISK REDUCTION

- As much as possible, use existing software and hardware that you can test before committing to its use.
- When you must use new technology (or technology new to you), get advice from people who have used the technology before.
- Create realistic, practical prototypes and benchmarks to make sure that technologies work as advertised.
- Perform compatibility testing to ensure that new technologies work well with existing technologies.

## Lack of Specialist Technical Knowledge

Designing a large information system is a complex undertaking that requires a huge amount of specialist knowledge about many different subjects. No one person can possibly be an expert on all of the technologies you may need to use. This is why we use teams of people to develop systems and why some people specialize in particular technologies, allowing them to advise others.

Given the number of technologies used in many systems, it can be difficult to assemble a project team with expertise in all of the technologies required. This can lead to a situation where you end up relying on vendor claims for products rather than proven knowledge and experience.

### RISK REDUCTION

- Bring specialist knowledge into your team so that you have mastery of all of the key technologies you need to use to deliver your system. If you don't need the knowledge full-time, hire trusted and experienced part-time experts as needed.

- Obtain external expert review of your architecture to validate your assumptions and decisions.
- Obtain binding contractual commitments from your technology suppliers where possible.

### Late Consideration of the Deployment Environment

The deployment environment is where your system hits reality. We've seen problems in some projects when the system is designed from a purely software-oriented perspective and the deployment environment is considered only when the software is complete. Remember that an inappropriate deployment environment can make an otherwise good system totally unusable.

The deployment environment also often affects how the software is designed and implemented, and this can be expensive to change. For example, if plans change and you need to use a group of small machines rather than a single large machine to host your server elements, this could have a significant impact on the architecture of your server software, a change that would be expensive to make late in delivery.

#### RISK REDUCTION

- Design your deployment environment as part of architecture definition rather than as part of a separate exercise performed after the system has been developed.
- Obtain external expert review of your architecture to get early feedback before you spend too much time or money.

## CHECKLIST

- Have you mapped all of the system's functional elements to a type of hardware device? Have you mapped them to specific hardware devices if appropriate?
- Is the role of each hardware element in the system fully understood? Is the specified hardware suitable for the role?
- Have you established detailed specifications for the system's hardware devices? Do you know exactly how many of each device are required?
- Have you identified all required third-party software and documented all the dependencies between system elements and third-party software?
- Is the network topology required by the system understood and documented?

- Have you estimated and validated the required network capacity? Can the proposed network topology be built to support this capacity?
- Have network specialists validated that the required network can be built?
- Have you performed compatibility testing when evaluating your architectural options to ensure that the elements of the proposed deployment environment can be combined as desired?
- Have you used enough prototypes, benchmarks, and other practical tests when evaluating your architectural options to validate the critical aspects of the proposed deployment environment?
- Can you create a realistic test environment that is representative of the proposed deployment environment?
- Are you confident that the deployment environment will work as designed? Have you obtained external review to validate this opinion?
- Are the assessors satisfied that the deployment environment meets their requirements in terms of standards, risks, and costs?
- Have you checked that the physical constraints (such as floor space, power, cooling, and so on) implied by your required deployment environment can be met?

## FURTHER READING

A great deal of literature describes specific deployment technologies; unfortunately, little of it discusses how to design a realistic and reliable system deployment environment. Some software architecture books [CLEM03; GARL03; HOFM00] contain explanations of how to document deployment views. Dyson and Longshaw [DYSO04] includes a number of patterns appropriate to the Deployment view.