Sun Java Enterprise System Deployment Planning Guide

Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95054
U.S.A.

Part No: 819–2326–12
March 2007

070402@17039

Contents

Tables

Figures

Preface

The *Sun Java Enterprise System Deployment Planning Guide* provides an introduction to planning and designing enterprise deployment solutions based on Sun Java™ Enterprise System. This guide presents basic concepts and principles of deployment planning and design, discusses the solution life cycle, which encapsulates the phases and tasks of a deployment design project, and provides high-level examples and strategies that you can when planning enterprise-wide deployment solutions with Java Enterprise System (Java ES).

Who Should Use This Book

This guide is primarily intended for deployment architects and business planners responsible for the analysis and design of enterprise deployments. This guide is also useful for system integrators and others responsible for the design and implementation of various aspects of an enterprise application.

Before You Read This Book

This guide assumes you are familiar with the design and installation of enterprise-level applications, and that you have read the *Sun Java Enterprise System 5 Technical Overview*.

How This Book Is Organized

This guide is based on a solution life cycle which describes the various phases of deployment planning. Chapter 1 provides a description of the solution life cycle.

Java ES Documentation Set

The Java ES documentation set describes deployment planning and system installation. The URL for system documentation is `http://docs.sun.com/coll/1286.2`. For an introduction to Java ES, refer to the books in the order in which they are listed in the following table.

TABLE P–1 Java Enterprise System Documentation

Document Title	Contents
Sun Java Enterprise System 5 Release Notes for UNIX *Sun Java Enterprise System 5 Release Notes for Microsoft Windows*	Contains the latest information about Java ES, including known problems. In addition, components have their own release notes listed in the Release Notes Collection (`http://docs.sun.com/coll/1315.2`).
Sun Java Enterprise System 5 Technical Overview	Introduces the technical and conceptual foundations of Java ES. Describes components, the architecture, processes, and features.
Sun Java Enterprise System Deployment Planning Guide	Provides an introduction to planning and designing enterprise deployment solutions based on Java ES. Presents basic concepts and principles of deployment planning and design, discusses the solution life cycle, and provides high-level examples and strategies to use when planning solutions based on Java ES.
Sun Java Enterprise System 5 Installation Planning Guide	Helps you develop the implementation specifications for the hardware, operating system, and network aspects of your Java ES deployment. Describes issues such as component dependencies to address in your installation and configuration plan.
Sun Java Enterprise System 5 Installation Guide for UNIX *Sun Java Enterprise System 5 Installation Guide for Microsoft Windows*	Guides you through the process of installing Java ES. Also shows how to configure components after installation, and verify that they function properly.
Sun Java Enterprise System 5 Installation Reference for UNIX	Gives additional information about configuration parameters, provides worksheets to use in your configuration planning, and lists reference material such as default directories and port numbers on the Solaris Operating System and Linux operating environment.
Sun Java Enterprise System 5 Upgrade Guide for UNIX *Sun Java Enterprise System 5 Upgrade Guide for Microsoft Windows*	Provides instructions for upgrading to Java ES 5 from previously installed versions.
Sun Java Enterprise System 5 Monitoring Guide	Gives instructions for setting up the Monitoring Framework for each product component and using the Monitoring Console to view real-time data and create monitoring rules.

TABLE P–1 Java Enterprise System Documentation *(Continued)*

Document Title	Contents
Sun Java Enterprise System Glossary	Defines terms that are used in Java ES documentation.

Typographic Conventions

The following table describes the typographic changes that are used in this book.

TABLE P–2 Typographic Conventions

Typeface	Meaning	Example
`AaBbCc123`	The names of commands, files, and directories, and onscreen computer output	Edit your `.login` file. Use `ls -a` to list all files. `machine_name% you have mail.`
`AaBbCc123`	What you type, contrasted with onscreen computer output	`machine_name%` **`su`** `Password:`
AaBbCc123	A placeholder to be replaced with a real name or value	The command to remove a file is `rm` *filename*.
AaBbCc123	Book titles, new terms, and terms to be emphasized (note that some emphasized items appear bold online)	Read Chapter 6 in the *User's Guide*. A *cache* is a copy that is stored locally. Do *not* save the file.

Shell Prompts in Command Examples

The following table shows default system prompts and superuser prompts.

TABLE P–3 Shell Prompts

Shell	Prompt
C shell on UNIX and Linux systems	`machine_name%`
C shell superuser on UNIX and Linux systems	`machine_name#`
Bourne shell and Korn shell on UNIX and Linux systems	`$`
Bourne shell and Korn shell superuser on UNIX and Linux systems	`#`

TABLE P–3 Shell Prompts *(Continued)*

Shell	Prompt
Microsoft Windows command line	`C:\`

Symbol Conventions

The following table explains symbols that might be used in this book.

TABLE P–4 Symbol Conventions

Symbol	Description	Example	Meaning
`[ ]`	Contains optional arguments and command options.	`ls [-l]`	The `-l` option is not required.
`{ | }`	Contains a set of choices for a required command option.	`-d {y|n}`	The `-d` option requires that you use either the `y` argument or the `n` argument.
`${ }`	Indicates a variable reference.	`${com.sun.javaRoot}`	References the value of the `com.sun.javaRoot` variable.
-	Joins simultaneous multiple keystrokes.	Control-A	Press the Control key while you press the A key.
+	Joins consecutive multiple keystrokes.	Ctrl+A+N	Press the Control key, release it, and then press the subsequent keys.
→	Indicates menu item selection in a graphical user interface.	File → New → Templates	From the File menu, choose New. From the New submenu, choose Templates.

Accessing Sun Resources Online

The docs.sun.com^{SM} web site enables you to access Sun technical documentation online. You can browse the docs.sun.com archive or search for a specific book title or subject. Books are available as online files in PDF and HTML formats. Both formats are readable by assistive technologies for users with disabilities.

To access the following Sun resources, go to `http://www.sun.com`:

- Downloads of Sun products
- Services and solutions
- Support (including patches and updates)
- Training
- Research

- Communities (for example, Sun Developer Network)

Third-Party Web Site References

Third-party URLs are referenced in this document and provide additional, related information.

Note – Sun is not responsible for the availability of third-party web sites mentioned in this document. Sun does not endorse and is not responsible or liable for any content, advertising, products, or other materials that are available on or through such sites or resources. Sun will not be responsible or liable for any actual or alleged damage or loss caused or alleged to be caused by or in connection with use of or reliance on any such content, goods, or services that are available on or through such sites or resources.

Sun Welcomes Your Comments

Sun is interested in improving its documentation and welcomes your comments and suggestions. To share your comments, go to `http://docs.sun.com` and click Send Comments. In the online form, provide the full document title and part number. The part number is a 7-digit or 9-digit number that can be found on the book's title page or in the document's URL. For example, the part number of this book is 819-2326.

◆ ◆ ◆ CHAPTER 1

Introduction to Deployment Planning

This chapter provides a brief overview of Sun Java™ Enterprise System (Java ES), discusses deployment planning concepts, and introduces the solution life cycle, which outlines the various steps for planning and designing enterprise software systems. This chapter contains the following sections:

About Java Enterprise System

Java Enterprise System is a software infrastructure that provides a complete set of middleware services to support enterprise applications distributed across a network or Internet environment. The Java Enterprise System components that provide the services are installed using a common installer, synchronized on a common set of shared libraries, and share an integrated user identity and security management system.

System Services

The main infrastructure services provided by Java Enterprise System components can be categorized as follows:

- **Portal services.** These services enable mobile employees, telecommuters, knowledge workers, business partners, suppliers, and customers to securely access their personalized corporate portal from anywhere outside the corporate network through the Internet. These services provide anytime, anywhere access capabilities to user communities, delivering integration, aggregation, personalization, security, mobile access, and search.
- **Communications and collaboration services.** These services enable the secure interchange of information among diverse user communities. Specific capabilities include messaging, real-time collaboration, and calendar scheduling in the context of the user\qs business environment.

- **Network identity and security services.** These services improve security and protection of key corporate information assets by ensuring that appropriate access control policies are enforced across all communities, applications, and services on a global basis. These services work with a repository for storing and managing identity profiles, access privileges, and application and network resource information.
- **Web and application services.** These services enable distributed components to communicate with one another and support the development, deployment, and management of applications for a broad range of servers, clients, and devices. These services are based on Java 2 Platform, Enterprise Edition (J2EE™) technology.
- **Availability services.** These services provide near-continuous availability and scalability for applications and web services.

The following table lists the preceding service categories and specifies the Java Enterprise System components that provide services for each category.

TABLE 1–1 Java Enterprise System Service Categories

Service Category	Java Enterprise System Components
Portal services	Portal Server, Portal Server Secure Remote Access, Access Manager, Directory Server, Application Server or Web Server
Communication and collaboration services	Messaging Server, Calendar Server, Instant Messaging, Access Manager, Directory Server, Application Server or Web Server
Network identity services	Access Manager, Directory Server, Web Server
Web and application services	Application Server, Message Queue, Web Server
Availability services	Sun Cluster, Sun Cluster Agents

For more information about Java Enterprise System services, components, and Java Enterprise System architectural concepts, refer to the *Sun Java Enterprise System 5 Technical Overview*.

Built-In Services and Custom-Developed Services

Deployment solutions based on Java Enterprise System typically fall into two general categories:

- **80:20 deployments.** These solutions consist primarily of services provided by Java Enterprise System. Java Enterprise System provides about 80% or more of the services.
- **20:80 deployments.** These solutions consist of a significant number of custom-developed services and third-party applications.

The 80:20 and 20:80 categories are broad generalizations. The exact percentage of the type of services offered is not important. However, the percentage indicates the amount of customization a solution contains.

Java Enterprise System is well suited for 80:20 deployments because of the rich set of services provided by Java ES. For example, it is relatively easy to deploy an enterprise-wide communications system or an enterprise-wide portal system using services provided by Java Enterprise System.

For deployments that require custom development, Java Enterprise System supports the creation and integration of custom-developed services and applications.

Most of the service categories listed in "System Services" on page 17 can be used to deliver 80:20 deployments. For example, communications and collaboration services provide email, calendar, and instant messaging services to end users, allowing them to aggregate and personalize the content. Similarly, the network identity and enterprise portal categories of services allow you to install and configure enterprise-wide applications without developing or integrating custom services.

Enterprise solutions that require custom development of J2EE platform services can leverage Application Server, Message Queue, or Web Server which are provided with Java Enterprise System web and application services.

Enterprise deployments can vary greatly in the number of custom-developed services they require. Because of the interoperability between Java Enterprise System services, you can create your own suite of services tailored to your particular enterprise needs.

Migrating to Java Enterprise System

The planning, designing, and implementing of an enterprise solution that uses Java Enterprise System depends largely on your current deployment strategy. For enterprises that are planning a first-time deployment solution, the planning, design, and implementation is driven largely by the specific needs of your enterprise. However, first-time deployments solutions are not typical. More likely are solutions that use Java Enterprise System to enhance existing enterprise solutions or to upgrade from earlier versions of Java Enterprise System components.

When replacing or upgrading existing solutions, you must take additional planning, design, and implementation steps to ensure that existing data is preserved and that software is properly upgraded to current versions. As you proceed through the analysis and design outlined in this guide, keep in mind the preparation and planning required to replace and upgrade existing software systems.

For more information about upgrading to the current version of Java Enterprise System and strategies for migration from other applications, refer to the *Java Enterprise System Upgrade and Migration Guide*.

About Deployment Planning

Deployment planning is a critical step in the successful implementation of a Java Enterprise System solution. Each enterprise has its own set of goals, requirements, and priorities to consider. Successful planning starts with analyzing the goals of an enterprise and determining the business requirements to meet those goals. The business requirements must then be converted into technical requirements that can be used as a basis for designing and implementing a system that can meet the goals of the enterprise.

Successful deployment planning is the result of careful preparation, analysis, and design. Errors and missteps that occur anywhere during the planning process can result in a system that can misfire in many ways. Significant problems can arise from a poorly planned system. For example, the system could under-perform, be difficult to maintain, be too expensive to operate, could waste resources, or could be unable to scale to meet increasing needs.

Solution Life Cycle

The solution life cycle shown in the following figure depicts the steps in the planning, design, and implementation of an enterprise software solution based on Java Enterprise System. The life cycle is a useful tool for keeping a deployment project on track.

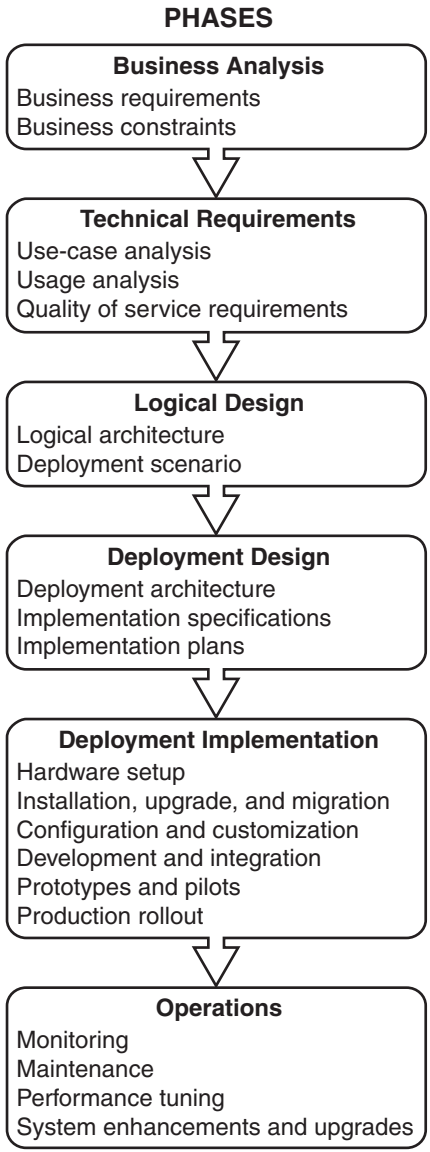


FIGURE 1–1 Solution Life Cycle

The life cycle consists of ordered phases. Each phase consists of related tasks that result in outputs that are carried forward as inputs to subsequent phases. The tasks within each phase are iterative, requiring thorough analysis and design before generating the outputs for that phase. The early phases can be iterative also. For example, during the deployment design phase, you might discover that the analysis in earlier phases is insufficient and requires more work.

The following sections in this chapter briefly describe each life cycle phase.

Business Analysis Phase

During business analysis, you define the business goals of a deployment project and state the business requirements that must be met to achieve those goals. When stating the business requirements, consider any business constraints that might affect the ability to achieve the business goal. Throughout the life cycle, you measure the success of your deployment planning, and ultimately your deployment solution, according to the analysis performed in the business analysis phase.

During the business analysis phase you create business requirements documents that you later use as inputs to the technical requirements phase.

For more information on the business analysis phase, refer to Chapter 2

Technical Requirements Phase

The technical requirements phase starts with the business requirements and business constraints defined during the business analysis phase and translates them into technical specifications that can be used to subsequently design the deployment architecture. The technical requirements specify quality of service (QoS) features, such as performance, availability, security, and others.

During the technical requirements phase, you create documents that contain the following information:

- Analysis of user tasks and usage patterns
- Use cases that model user interaction with the planned system
- Quality of service requirements derived from the business requirements, possibly taking into consideration the analysis of user tasks and usage patterns

The resulting usage analysis, use cases, and QoS requirements documents are inputs to the logical design phase of the solution life cycle. The usage analysis also plays a significant role in the deployment design phase.

During the technical requirements phase, you might also specify service level requirements that are the basis for subsequently creating service level agreements (SLA). Service level agreements specify the terms under which customer support must be provided to maintain the system and are generally signed as part of project approval in the deployment design phase.

For more information on technical requirements, refer to Chapter 3

Logical Design Phase

During logical design, using use cases from the technical requirements phase as inputs, you identify the Java Enterprise System components necessary to implement a solution. You identify components that provide support to those Java ES components, and also identify any additional custom-developed components necessary to meet the business requirements. You then map the components within a logical architecture that shows the interrelationships among the components. The logical architecture does not specify any hardware required to implement the solution.

The output of the logical design phase is the logical architecture. The logical architecture by itself is not sufficient to begin deployment design. You also need the QoS requirements from the technical requirements phase. The logical architecture and the QoS requirements from the technical requirements phase form a deployment scenario. This deployment scenario is the input to the deployment design phase.

For more information on logical design, refer to Chapter 4

Deployment Design Phase

During deployment design, you map the components specified in the logical architecture to a physical environment, producing a high-level deployment architecture. You also create an implementation specification, which provides low-level details specifying how to build the deployment architecture. Additionally, you create a series of plans and specifications that detail different aspects of implementing the software solution.

Project approval occurs during the deployment design phase. During project approval, the cost of the deployment is assessed. If approved, contracts for implementation of the deployment are signed, and resources to build the project are acquired. Often, project approval occurs after the implementation specification has been detailed. However, approval can also occur upon completion of the deployment architecture.

The outputs of the deployment design phase include the following:

- **Deployment architecture.** A high-level design document that represents the mapping of components to network hardware and software.
- **Implementation specifications.** Detailed specifications used as blueprints for building the deployment.
- **Implementation plans.** A group of plans and specifications that cover various aspects of implementing an enterprise software solution. Implementation plans include a migration plan, installation plan, user management plan, test plan, and others.

For more information about deployment design, refer to Chapter 5

Implementation Phase

During the implementation phase, you work from specifications and plans created during deployment design to build the deployment architecture and implement the solution. Depending on the nature of your deployment project, this phase includes some or all of the following tasks:

- Installing and configuring the hardware infrastructure
- Installing and configuring the software
- Modeling users and resources within an LDAP directory design
- Migrating data from existing directories and databases according to a user management plan
- Creating and deploying pilot and prototype deployments in a test environment
- Designing and running functional tests to measure compliance with system requirements
- Designing and running stress tests to measure performance under peak loads
- Developing and integrating any custom enterprise applications
- Creating a production deployment, which might be phased into production in stages

  Once a deployment is in production, you proceed to the operations phase of the solution life cycle.

For more information on the implementation phase, refer to Chapter 6

Operations Phase

The operations phase covers tasks necessary to keep the implementation of the deployment running smoothly. This phase includes the following:

- Monitoring the deployment to ensure that the system is running according to plans
- Performance tuning to ensure that the deployed software runs at optimal levels
- Providing scheduled maintenance for smooth operations and unscheduled maintenance as the need arises
- Upgrading software and hardware as the need arises

Details about the operations phase are out of scope for this guide.

◆ ◆ ◆ CHAPTER 2

Business Analysis

During the business analysis phase of the solution life cycle you define business goals by analyzing a business problem and identifying the business requirements and business constraints to meet that goal.

This chapter contains the following sections:

About Business Analysis

Business analysis starts with stating the business goals. You then analyze the business problems you must solve and identify the business requirements that must be met to achieve the business goals. Consider also any business constraints that limit your ability to achieve the goals. The analysis of business requirements and constraints results in a set of business requirements documents.

You use the resulting set of business requirements documents as a basis for deriving technical requirements in the technical requirements phase. Throughout the solution life cycle, you measure the success of your deployment planning and ultimately the success of your solution according to the analysis performed in the business analysis phase.

Defining Business Requirements

No simple formula exists that can identify all business requirements. You determine the requirements based on collaboration with the stakeholders requiring a software solution, your own knowledge about the business domain, and applied creative thinking.

This section provides some factors to consider when defining business requirements.

Setting Business Goals

Business analysis should articulate the goals of a deployment project. Clear goals help focus design decisions and prevent the project from going astray. Contrasting the business goals with current operations also helps determine design decisions.

Scope

Business requirements should state the scope of the deployment project. Make sure you identify areas that can be solved and avoid open-ended requirements that make the goal either unclear or unreachable. A poorly defined scope can lead to a deployment design that insufficiently addresses business needs or that is extravagant with resources.

Priorities

Prioritize your goals to ensure that the most important aspects of the deployment can be achieved first. Limited resources might require postponement or modification of some goals. For example, large and complex deployments generally require phased implementation of the solution. By stating the priorities, you provide guidance on decisions that might need to be made for your deployment design to be accepted by the stakeholders.

Critical Qualities

Identify areas that are critical to success to allow stakeholders and designers to concentrate on the most important criteria.

Growth Factors

As you set business goals, consider not only the current needs of the organization, but anticipate how these needs might change and grow over extended periods. You do not want a solution that is outdated prematurely.

Safety Margin

The design of your solution is based on assumptions made during this business analysis phase. These assumptions might not be accurate for various reasons, such as insufficient data, errors in

judgement, or unanticipated external events. Make sure you plan for a safety margin not only in your business goals but throughout your planning so the solution that you design can handle unexpected events.

Understanding User Needs

Do the research necessary to understand the types of users that the solution targets, their needs, and the expected benefits to them. For example, the following list provides one way to categorize users:

- Current employees only
- Current and previous employees
- Administrators
- Active customers
- All customers
- Membership site
- General public
- Restricted access

Clearly stating the expected benefits to users helps drive design decisions. For example, here are some benefits that a solution can provide to users:

- Remote access to company resources
- Enterprise collaboration
- Simplification of daily tasks
- Sharing of resources by remote teams
- Increased productivity
- Self-administration by end-users

Developing Operational Requirements

Express operational requirements as a set of functional requirements with straightforward goals. Typically, you create operational specifications for areas such as:

- End-user functionality
- Reduced response time
- Availability and uptime
- Reduced error rate
- Information archival and retention

Express operational requirements in measurable terms that all stakeholders can understand. Avoid ambiguous language, such as "adequate end-user response time." Examples of operational requirements could be the following:

- Ability to restore services within 10 minutes of an outage
- Ability to replay the last 48 hours of inbound message delivery

- Online transactions completed within 60 seconds during peak periods
- End-user authentication completed within four seconds during peak periods

Supporting Existing Usage Patterns

Express existing usage patterns as clearly measurable goals. The following questions can help determine such goals.

- How are current services utilized?
- What are the usage patterns (for example, sporadic, frequent, or heavy usage)?
- To which sites do your users typically connect?
- What size messages do users commonly send?
- How many transactions do users typically complete per day or per hour?

Study the users who access your services. Factors such as when users access existing services and for how long are keys to identifying your goals. If your organization's experience cannot provide these patterns, study the experience of similar organizations.

Understanding Corporate Culture

Requirements analysis should take into account various aspects of corporate culture and politics. Lack of attention to corporate culture can result in a solution that is not well received or is difficult to implement.

Stakeholders

Identify individuals and organizations that have a vested interest in the success of the proposed solution. All stakeholders should actively participate in defining the business goals and requirements. If a stakeholder does not participate or is uninformed of planned changes, the plans could have significant shortcomings. Such a stakeholder could even block the implementation of the deployment.

Standards and Policies

Make sure you understand the standards and policies of the organization requesting the solution. These standards and policies might affect technical aspects of the design, product selection, and methodology of deployment.

One example is the confidentiality of personnel data that might be owned and controlled by the human resources organization or unit managers. Another example would be company procedures for change management. Change management policies could dramatically affect acceptance of a solution and influence the implementation methodology and time table.

Regulatory Requirements

Regulatory requirements vary greatly, depending on the nature of the business. Research and understand any regulatory requirements that might affect the deployment. Many companies and government agencies require compliance with accessibility standards. When deploying global solutions, consider foreign laws and regulations. For example, many European countries have strict controls on storing personal information.

Security

Some goals that you identify might have implicit security issues that should be emphasized. Call out specific security goals essential to the solution. For example:

- Authorized access to proprietary information
- Role-based access to confidential information
- Secure communication between remote locations
- Invocation of remote applications on local systems
- Secure transactions with third-party businesses and organizations
- Enforcement of security policies

Site Distribution

The geographic distribution of sites and the bandwidth between the sites can affect design decisions. Additionally, some sites might require local management.

Such geographic considerations can raise the project's training costs, complexity, and so forth. Clearly state requirements resulting from geographic distribution of sites. Highlight which sites are critical to the design's success.

Using an Incremental Approach

Often, you view a software solution as a whole, comprehensive system. However, you often achieve deployment of the complete system incrementally by taking measured steps.

When adopting an incremental approach, you typically design a road map that provides milestones leading to the ultimate, comprehensive solution. Additionally, you might have to consider short-term plans for aspects of the comprehensive solution that are deferred for later implementation.

The incremental approach provides these advantages:

- You can adapt to requirement changes due to business growth.
- You can leverage the existing infrastructure as you transition to your ultimate deployment implementation.
- You can accommodate capital expenditure requirements.

- You can leverage a small supply of human resources.
- You can allow for partnership possibilities.

Understanding Service Level Agreements

A service level agreement (SLA) specifies minimum performance requirements and, upon failure to meet those requirements, the level and extent of customer support that must be provided. A service level agreement is based on business requirements defined during business analysis, which are later specified as service level requirements during the technical requirements phase. The SLA is signed during project approval, which occurs in the deployment design phase.

You should develop an SLA around areas such as uptime, response time, message delivery time, and disaster recovery. An SLA should account for items such as an overview of the system, the roles and responsibilities of support organizations, how to measure service levels, change requests, and so forth. Identifying your organization's expectations around system availability is key in determining the scope of an SLA.

Defining Business Constraints

Business constraints play a significant role in determining the nature of a deployment project. One key to successful deployment design is finding the optimal way to meet business requirements within known business constraints. The business constraints can be fiscal limitations, physical limitations (for example, network capacity), time limitations (for example, completion before significant events such as the next annual meeting), or any other limitation you anticipate as a factor that affects the achievement of the business goal.

This section describes several factors to consider when defining business constraints.

Migration Issues

Typically, a deployment project replaces or supplements existing software infrastructure and data. Any new solution must be able to migrate data and procedures from the existing infrastructure to the new solution, often retaining interoperability with existing applications. An analysis of the current infrastructure is necessary to determine the extent migration issues play into the proposed solution.

Schedule Mandates

The schedule for implementation of a solution can affect design decisions. Aggressive schedules might result in scaling back of goals, changing priorities, or adopting an incremental solution

approach. Within a schedule, significant milestones might exist that deserve consideration as well. Milestones can be set by internal events such as scheduled service rollouts or external events such as the opening date of a school semester.

Budget Limitations

Most deployment projects must adhere to a budget. Considering the cost of building the proposed solution and the resources required to maintain the solution over a specific lifetime including the following:

- **Existing hardware and network infrastructure.** Reliance on existing infrastructure can affect the design of a system.
- **Development resources needed to implement the solution.** Limited development resources, including hardware, software, and human resources, might suggest incremental deployment. You might have to reuse the same resources or development teams for each incremental phase you implement.
- **Maintenance, administration, and support.** Analyze the resources available to administer, maintain, and support users on the system. Limited resources might impact design decisions.

Cost of Ownership

In addition to maintenance, administration, and support, analyze other factors that affect the cost of ownership. Hardware and software upgrades might be necessary, the impact of the solution on the power grid, telecommunications cost, and other factors influence out-of-pocket expenses. Service level agreements specifying availability levels for the solution also affect the cost of ownership by requiring increased redundancy.

The implementation of a solution should provide a return on the investment into the solution. Analysis of return on investment typically involves measuring the financial benefits gained from the expenditure of capital.

Estimating the financial benefits of a solution involves a careful analysis of the goals to be achieved in comparison with alternate ways of achieving those goals and with the cost of doing nothing at all.

◆ ◆ ◆ CHAPTER 3

Technical Requirements

During the technical requirements phase of the solution life cycle you perform a usage analysis, identify use cases, and determine quality of service requirements for the proposed deployment solution.

This chapter contains the following sections:

About Technical Requirements

Technical requirements analysis begins with the business requirements documents created during the business analysis phase of the solution life cycle. Using the business analysis as a basis, you do the following:

- Perform a usage analysis to aid in determining expected load conditions.
- Create use cases that model typical user interaction with the system.
- Create a set of quality of service requirements (QoS) that define how a deployed solution must perform in areas such as response time, availability, security, and others.

  The quality of service requirements are derived from the usage analysis and the use cases, keeping in mind business requirements and constraints previously identified.

  The quality of service requirements are later paired with logical architectures in the logical design phase to form a deployment scenario. The deployment scenario is the main input to the deployment design phase of the solution life cycle.

As with business analysis, no simple formula for technical requirements analysis exists that generates the usage analysis, use cases, and system requirements. Technical requirements analysis requires an understanding of the business domain, business objectives, and the underlying system technology.

Usage Analysis

Usage analysis involves identifying the various users of the solution you are designing and determining the usage patterns for those users. The information you gather provides a basis for estimating the load conditions on the system. Usage analysis information is also useful when assigning weights to use cases, as described in "Use Cases" on page 35.

During usage analysis, you should interview users whenever possible, research existing data on usage patterns, and interview builders and administrators of previous systems. The following table lists factors to consider when performing a usage analysis.

TABLE 3–1 Usage Analysis Factors

Topic	Description
Number and type of users	Identify how many users your solution must support, and categorize those users, if necessary. For example: ■ A Business to Customer (B2C) solution might have a large number of visitors, but only a small number of users who register and engage in business transactions. ■ A Business to Employee (B2E) solution typically accommodates each employee, although some employees might need access from outside the corporate network. In a B2E solution, managers might need authorization to areas that regular employees cannot access.
Active and inactive users	Identify the usage patterns and ratios of active and inactive users. Active users are those users logged into the system and interact with the system's services. Inactive users can be users who are not logged in, users who log in but do not interact with the system's components, or users who are in the database but never log in.
Administrative users	Identify users that access the deployed system to monitor, update, and support the deployment. Determine any specific administrative usage patterns that might affect technical requirements (for example, administration of the deployment from outside the firewall).

TABLE 3–1 Usage Analysis Factors *(Continued)*

Topic	Description
Usage patterns	Identify how various types of users access the system and provide targets for expected usage. For example: ■ Are there peak times when usage spikes? ■ What are normal business hours? ■ Are users distributed globally? ■ What is the expected duration of user connectivity?
User growth	Determine if the size of the user base is fixed or if the deployment expects growth in the number of users. If the user base is expected to grow, try to create reasonable projections of the growth.
User transactions	Identify the type of user transactions that must be supported. These user transactions can be translated into use cases. For example: ■ What tasks do users perform? ■ When users log in, do they remain logged in? Do they typically perform a few tasks and log out? ■ Will significant collaboration between users require common calendars, web-conferences, and deployment of internal web pages?
User studies and statistical data	Use pre-existing user studies and other sources to determine patterns of user behavior. Often, enterprises or industry organizations have user research studies from which you can extract useful information about users. Log files for existing applications might contain statistical data useful in making estimates for a system.

Use Cases

Use cases model typical user interaction with the solution that you are designing, and describe the complete flow of an operation from the perspective of an end user. Prioritizing the design around a complete set of use cases ensures a continual focus on the delivery of expected functionality. Use cases are the principal input to logical design.

Assign relative weights to use cases, with the highest weighted use cases representing the most common user tasks. The weighting of use cases allows you to focus your design decisions on the system services that are used the most.

Use cases can be described at two levels.

- **Use-case reports.** Descriptions of individual use cases, including primary and alternative flows of events.
- **Use-case diagrams.** Diagrams depicting the relationships among actors and the use cases, presenting a more formal organization of the flow of events. Use-case diagrams are useful to model long or complex use cases. Typically, you use Unified Modeling Language (UML) standards to draw use case diagrams.

Quality of Service Requirements

Quality of service (QoS) requirements are technical specifications that specify the system quality of features such as performance, availability, scalability, and serviceability. QoS requirements are driven by business needs specified in the business requirements. For example, if services must be available 24 hours a day throughout the year, the availability requirement must address this business requirement.

The following table lists the system qualities that typically form a basis for QoS requirements.

TABLE 3–2 System Qualities Affecting QoS Requirements

System Quality	Description
Performance	The measurement of response time and throughput with respect to user load conditions.
Availability	A measure of how often a system's resources and services are accessible to end users, often expressed as the *uptime* of a system.
Scalability	The ability to add capacity (and users) to a deployed system over time. Scalability typically involves adding resources to the system but should not require changes to the deployment architecture.
Security	A complex combination of factors that describe the integrity of a system and its users. Security includes authentication and authorization of users, security of data, and secure access to a deployed system.
Latent capacity	The ability of a system to handle unusual peak loads without additional resources. Latent capacity is a factor in availability, performance, and scalability qualities.
Serviceability	The ease by which a deployed system can be maintained, including monitoring the system, repairing problems that arise, and upgrading hardware and software components.

System qualities are closely interrelated. Requirements for one system quality might affect the requirements and design for other system qualities. For example, higher levels of security might affect performance, which in turn might affect availability. Adding additional servers to address availability issues affect serviceability (maintenance costs).

Understanding how system qualities are interrelated and the trade-offs that must be made is the key to designing a system that successfully satisfies both business requirements and business constraints.

The following sections describe further the system qualities that impact deployment design, providing guidance on factors to consider when formulating QoS requirements. A section on service level requirements, which form the basis of service level agreements, is also included.

Performance

Business requirements typically express performance in nontechnical terms that specify response time. For example, a business requirement for web-based access might state the following:

Users expect a reasonable response time upon login, typically no greater than four seconds.

Starting with this business requirement, examine all use cases to determine how to express this requirement at a system level. In some cases, you might want to include user load conditions determined during usage analysis. Express the performance requirement for each use case in terms of response time under specified load conditions or response time plus throughput. You might also specify the allowable number of errors.

Here are two examples of how to specify system requirements for performance:

- Response for web page refresh must be no greater than four seconds throughout the day, sampled at 15-minute intervals, with fewer than 3.4 errors per million transactions.
- During defined peak periods, the system must allow 25 secure logins per second with response time no greater than 12 seconds for any user and with fewer than 3.4 errors per million transactions.

Performance requirements are closely related to availability requirements (how failover impacts performance) and latent capacity (how much capacity is available to handle unusual peak loads).

Availability

Availability is a way to specify the uptime of a system and is typically measured as the percentage of time that the system is accessible to users. The time that the system is not accessible (downtime) can be due to the failure of hardware, software, the network, or any other factor (such as loss of power) that causes the system to be down. Scheduled downtime for service (maintenance and upgrades) is not considered downtime. A basic equation to calculate system availability in terms of percentage of uptime is:

```
Availability = uptime / (uptime + downtime) * 100%
```

Typically you measure availability by the number of "nines" you can achieve. For example, 99% availability is two nines. Specifying additional nines significantly affects the deployment design. The following table quantifies the unscheduled downtime for additional nines of availability to a system that is running 24x7 year-round (a total of 8,760 hours).

TABLE 3–3 Unscheduled Downtime for a System Running Year-Round (8,760 hours)

Number of Nines	Percentage Available	Unscheduled Downtime
2	99%	88 hours
3	99.9%	9 hours
4	99.99%	45 minutes
5	99.999%	5 minutes

Fault-Tolerant Systems

Availability requirements of four or five nines typically require a system that is fault-tolerant. A fault-tolerant system must be able to continue service even during a hardware or software failure. Typically, fault tolerance is achieved by redundancy in both hardware (such as CPUs, memory, and network devices) and in software providing key services.

A single point of failure is a hardware or software component that is part of a critical path but is not backed up by redundant components. The failure of this component results in the loss of service for the system. When designing a fault-tolerant system, you must identify and eliminate potential single points of failure.

Fault-tolerant systems can be expensive to implement and maintain. Make sure you understand the nature of the business requirements for availability and consider the strategies and costs of availability solutions that meet those requirements.

Prioritizing Service Availability

From a user perspective, availability often applies more on a service-by-service basis rather than on the availability of the entire system. For example, the unavailability of instant messaging services usually has little or no impact on the availability of other services. However, the unavailability of services upon which many other services depend (such as Directory Server) has a much wider impact. Higher availability specifications should clearly reference specific use cases and usage analysis that require the increased availability.

It is helpful to list availability needs according to an ordered set of priorities. The following table prioritizes the availability of different types of services.

TABLE 3–4 Availability of Services by Priority

Priority	Service Type	Description
1	Mission critical	Services that must be available at all times. For example, database services (such as LDAP directories) to applications.
2	Must be available	Services that must be available, but can be available at reduced performance. For example, messaging service availability might not be critical in some business environments.
3	Can be postponed	Services that must be available within a given time period. For example, calendar services availability might not be essential in some business environments.
4	Optional	Services that can be postponed indefinitely. For example, in some environments instant messaging services can be considered useful but not necessary.

Loss of Services

Availability design includes consideration for what happens when availability is compromised or when a component is lost. This includes considering whether users connected must restart sessions and how a failure in one area affects other areas of a system. QoS requirements should consider these scenarios and specify how the deployment reacts to these situations.

Scalability

Scalability is the ability to add capacity to a system so the system can support additional load from existing users or from an increased user-base. Scalability usually requires the addition of resources, but should not require changes in the design of the deployment architecture or loss of service due to the time required to add additional resources.

As with availability, scalability applies more to individual services provided by a system rather than to the entire system. However, for services upon which other services depend, such as Directory Server, scalability can have system-wide impact.

You do not necessarily specify scalability requirements with QoS requirements unless projected growth of the deployment is clearly stated in the business requirements. However, during the deployment design phase of the solution life cycle, the deployment architecture should always add some tolerance for scaling the system even if no QoS requirements for scalability have been specified.

Estimating Growth

Estimating the growth of a system to determine scalability requirements involves working with projections, estimates, and guesses that might not be fulfilled. Three keys to developing requirements for a scalable system are the following.

- **High performance design strategy.** During the specification of performance requirements, include latent capacity to handle loads that might increase over time. Also, maximize availability within budget constraints. This strategy allows you to absorb growth and better schedule milestones for scaling the system.
- **Incremental deployment.** Incremental deployment helps with scheduling the addition of resources. Specify clear milestones for scaling the system. Milestones are typically load-based requirements coordinated with specific dates for assessing scalability.
- **Extensive performance monitoring.** Monitoring performance helps determine when to add resources to the system. Requirements for monitoring performance can provide guidance to operators and administrators responsible for maintenance and upgrades.

The following table lists factors to consider for determining scalability requirements.

TABLE 3–5 Scalability Factors

Topic	Description
Analyze usage patterns	Understand the usage patterns of the current (or projected) user base by studying existing data. In the absence of current data, analyze industry data or market estimates.
Design for reasonable maximum scale	Design with a goal towards the maximum required scale for both known demand and possible demand. Often, this is a 24-month estimate based on performance evaluation of the existing user load and reasonable expectations of future load. The time period for the estimate depends largely on the reliability of projections.
Set appropriate milestones	Implement the deployment design in increments to meet short-term requirements with a buffer to allow for unexpected growth. Set milestones for adding system resources. For example: ■ Capital acquisition (such as quarterly or yearly) ■ Lead time to purchase hardware and software (such as one to six weeks) ■ Buffer (10% to 100%, depending on growth expectations)
Incorporate emerging technology	Understand emerging technology, such as faster processors and Web servers, and how this technology can affect the performance of the underlying architecture.

Security Requirements

Security is a complex topic that involves all levels of a deployed system. Developing security requirements revolves around identifying the security threats and developing a strategy to combat them. This security analysis includes the following steps:

1. Identifying critical assets

2. Identifying threats to those assets
3. Identifying vulnerabilities that expose the threats that create risk to the organization
4. Developing a security plan that mitigates the risk to the organization

The analysis of security requirements should involve a cross-section of stakeholders from your organization, including managers, business analysts, and information technology personnel. Often, an organization appoints a security architect to take the lead in the design and implementation of security measures.

The following section describes some of the areas that are covered in security planning.

Elements of a Security Plan

Planning for security of a system is part of deployment design that is essential to successful implementation. Consider the following when planning for security:

- **Physical security.** Physical security is the physical access to routers, servers, server rooms, data centers, and other parts of your infrastructure. Other security measures become compromised if an unauthorized person can walk into a server room and unplug routers.
- **Network security.** Network security is access to your network through firewalls, secure access zones, access control lists, and port access. For network security you develop strategies for unauthorized access, tampering, and denial of service (DoS) attacks.
- **Application and application data security.** Application and application data security covers access to user accounts, corporate data, and enterprise applications through authentication and authorization procedures and policies. This area includes defining the following policies:
  - Password policies
  - Access rights, such as delegated administration to users as opposed to administrator access
  - Account inactivation
  - Access control
  - Encryption policies, including secure transport of data and using certificates to sign data
- **Personal security practices.** An organization-wide security policy defines the working environment and practices with which all users must comply to ensure other security measures perform as designed. Typically, you develop a handbook or manual on security and also offer training to users on security practices. For an effective overall security policy, sound security practices must become part of the organization culture.

Latent Capacity

Latent capacity is the ability of a deployment to handle unusual peak load usage without the addition of resources. Typically, you do not specify QoS requirements directly around latent capacity, but this system quality is a factor in the availability, performance, and scalability of the system.

Serviceability Requirements

Serviceability is the ease with which a deployed system can be maintained, including tasks such as monitoring the system, repairing problems that arise, adding and removing users from the system, and upgrading hardware and software components.

When planning requirements for serviceability, consider the topics listed in the following table.

TABLE 3–6 Topics for Serviceability Requirements

Topic	Description
Downtime planning	Identify maintenance tasks that require specific services to be unavailable or partially unavailable. Some maintenance and upgrades can occur seamlessly to users, while others require interruption of service. When possible, schedule with users those maintenance activities that require downtime, allowing the users to plan for the downtime.
Usage patterns	Identify the usage patterns to determine the best time to schedule maintenance. For example, on systems where peak usage is during normal business hours, schedule maintenance in the evening or weekends. For geographically distributed systems, identifying these times can be more challenging.
Availability	Serviceability is often a reflection of your availability design. Strategies for minimizing downtime for maintenance and upgrades revolve around your availability strategy. Systems that require a high degree of availability have limited opportunities for maintenance, upgrades, and repair. Strategies for handling availability requirements affect how you handle maintenance and upgrades. For example, on systems that are distributed geographically, servicing can depend on the ability to route workloads to remote servers during maintenance periods. Also, systems requiring a high degree of availability might require more sophisticated solutions that automate restarting of systems with little human intervention.

TABLE 3–6 Topics for Serviceability Requirements *(Continued)*

Topic	Description
Diagnostics and monitoring	You can improve the stability of a system by regularly running diagnostic and monitoring tools to identify problem areas. Regular monitoring of a system can avoid problems before they occur, help balance workloads according to availability strategies, and improve planning for maintenance and downtime.

Service Level Requirements

A service level agreement (SLA) specifies minimum performance requirements and, upon failure to meet those requirements, the level and extent of customer support that must be provided. Service level requirements are system requirements that specify the conditions upon which the SLA is based.

As with QoS requirements, service level requirements derive from business requirements and represent a guarantee about the overall system quality that the deployed system must meet. Because the service level agreement is considered to be a contract, specification of service level requirements should be unambiguous. The service level requirements define exactly under what conditions the requirements are tested and precisely what constitutes failure to meet the requirements.

◆ ◆ ◆ CHAPTER 4

Logical Design

During the logical design phase of the solution life cycle, you design a logical architecture showing the interrelationships of the logical components of the solution. The logical architecture and the usage analysis from the technical requirements phase form a deployment scenario, which is the input to the deployment design phase.

This chapter contains the following sections:

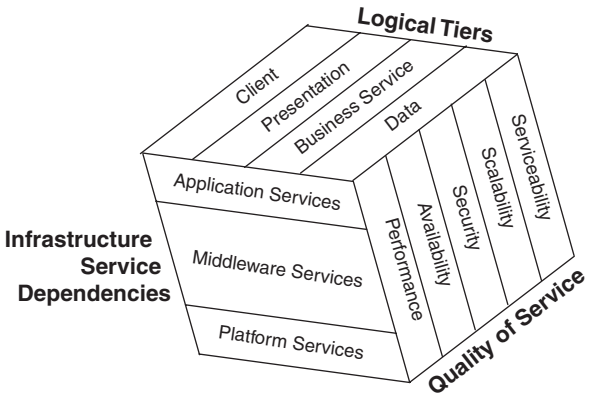
About Logical Architectures

A logical architecture identifies the software components needed to implement a solution, showing the interrelationships among the components. The logical architecture and the quality of service requirements determined during the technical requirements phase form a deployment scenario. The deployment scenario is the basis for designing the deployment architecture, which occurs in the next phase, deployment design.

When developing a logical architecture you need to identify not only the components that provide services to users, but also other components that provide necessary middleware and platform services. Infrastructure service dependencies and logical tiers provide two complementary ways of performing this analysis.

Infrastructure service dependencies and logical tiers are two of the three dimensions of the solution architecture upon which Sun Java™ Enterprise System is based. The three dimensions are listed below and are also represented in "About Logical Architectures" on page 45.

- **Infrastructure service dependencies.** Interacting software components that provide enterprise services. The software components require an underlying set of infrastructure services that allows the distributed components to communicate with each other and to interoperate.
- **Logical tiers.** A logical organization of software components into tiers that represent the logical and physical independence of software components, based on the nature of the services they provide.
- **Quality of service.** System service qualities, such as performance, availability, scalability, and others that represent particular aspects of a software solution's design and operation.



Note – For more information on Java Enterprise System architecture concepts, refer to the "Java Enterprise System Architecture" chapter in the *Sun Java Enterprise System 5 Technical Overview*.

A logical architecture depicts infrastructure service levels by showing the necessary components and their dependencies. A logical architecture also distributes the components among logical tiers that represent presentation, business, and data services that can be ultimately accessed by a client tier. Quality of service requirements are not modeled in the logical architecture but are paired with the logical architecture in a deployment scenario.

Designing a Logical Architecture

When you design a logical architecture, use the use cases identified during the technical requirements phase to determine the Java Enterprise System components that provide the services necessary for the solution. You must also identify any components providing services to the components you initially identify.

You place the Java Enterprise System components within the context of a multitiered architecture according to the type of services that they provide. Understanding the components as part of a multitiered architecture helps you later determine how to distribute the services provided by the components and also helps determine a strategy for implementing quality of service (such as scalability, availability, and others.)

Additionally, you can provide another view of the logical components that places them within secure access zones. The section "Access Zones" on page 61 provides an example of secure access zones.

Java Enterprise System Components

Java Enterprise System consists of interacting software components providing enterprise services that you can use to build your enterprise solution. The following figure shows the key software components provided with Java Enterprise System. The *Sun Java Enterprise System 5 Technical Overview* provides additional information on Java Enterprise System components and the services they provide.

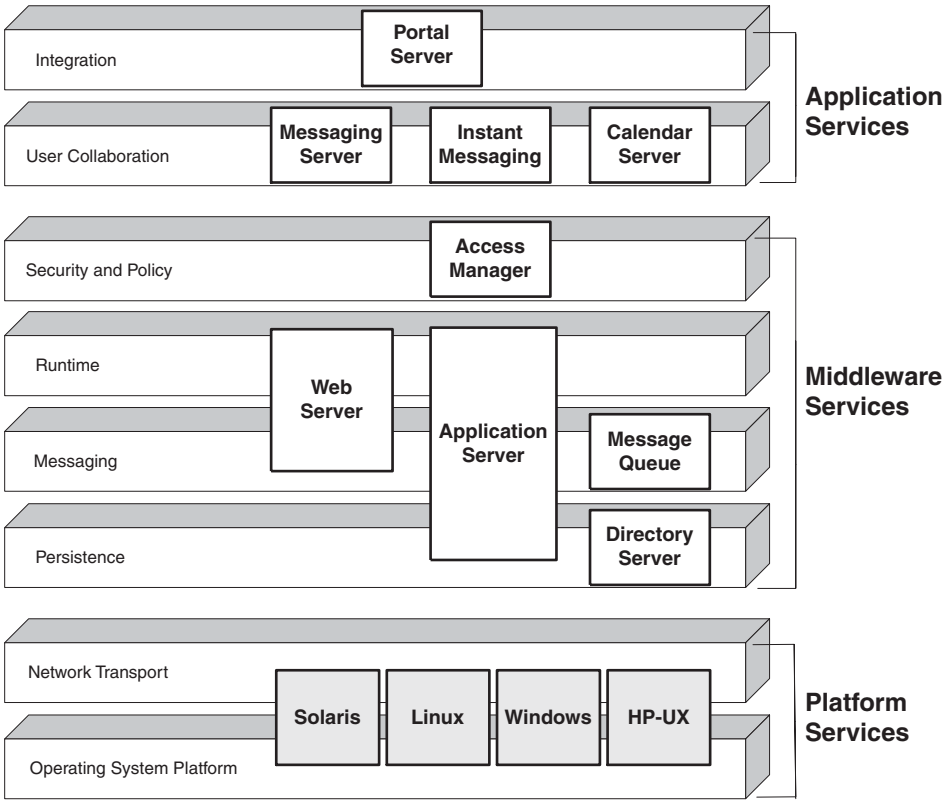


FIGURE 4–1 Java Enterprise System Components

Component Dependencies

When identifying Java Enterprise System components for a logical architecture, you need to also identify supporting components. For example, if you identify Messaging Server as a necessary component to a logical architecture, then your logical architecture must also include Directory Server and possibly Access Manager. Messaging Server depends on Directory Server for directory services and Access Manager for solutions requiring single sign-on.

The following table lists dependencies of Java Enterprise System components. Refer to "Component Dependencies" on page 48 for a visual representation of dependencies among key components. When designing a logical architecture, use this table and accompanying figure to determine dependent components in your design.

TABLE 4–1 Java Enterprise System Component Dependencies

Java Enterprise System Component	Depends On
Application Server	Message QueueDirectory Server (optional)
Calendar Server	Messaging Server (for email notification service)Access Manager (for single sign-on)Web Server (for web interface)Directory Server
Communications Express	Access Manager (for single sign-on)Calendar ServerMessaging ServerInstant MessagingWeb Server (for web interface)Directory Server
Directory Proxy Server	Directory Server
Directory Server	None
Access Manager	Application Server or Web ServerDirectory Server
Instant Messaging	Access Manager (for single sign-on)Directory Server
Message Queue	Directory Server (optional)
Messaging Server	Access Manager (for single sign-on)Web Server (for web interface)Directory Server
Portal Server	If configured to use Portal Server Channels: Calendar ServerMessaging ServerInstant Messaging Access Manager (for single sign-on)Application Server or Web ServerDirectory Server
Portal Server Secure Remote Access	Portal Server
Web Server	Access Manager (optional, for access control

Note – The dependencies among Java Enterprise System components listed in "Component Dependencies" on page 48 does not list all component dependencies. "Component Dependencies" on page 48 does not list dependencies that you must consider when planning for installation. For a complete list of Java Enterprise System dependencies, refer to the *Sun Java Enterprise System 5 Installation Guide for UNIX*.

Web Container Support

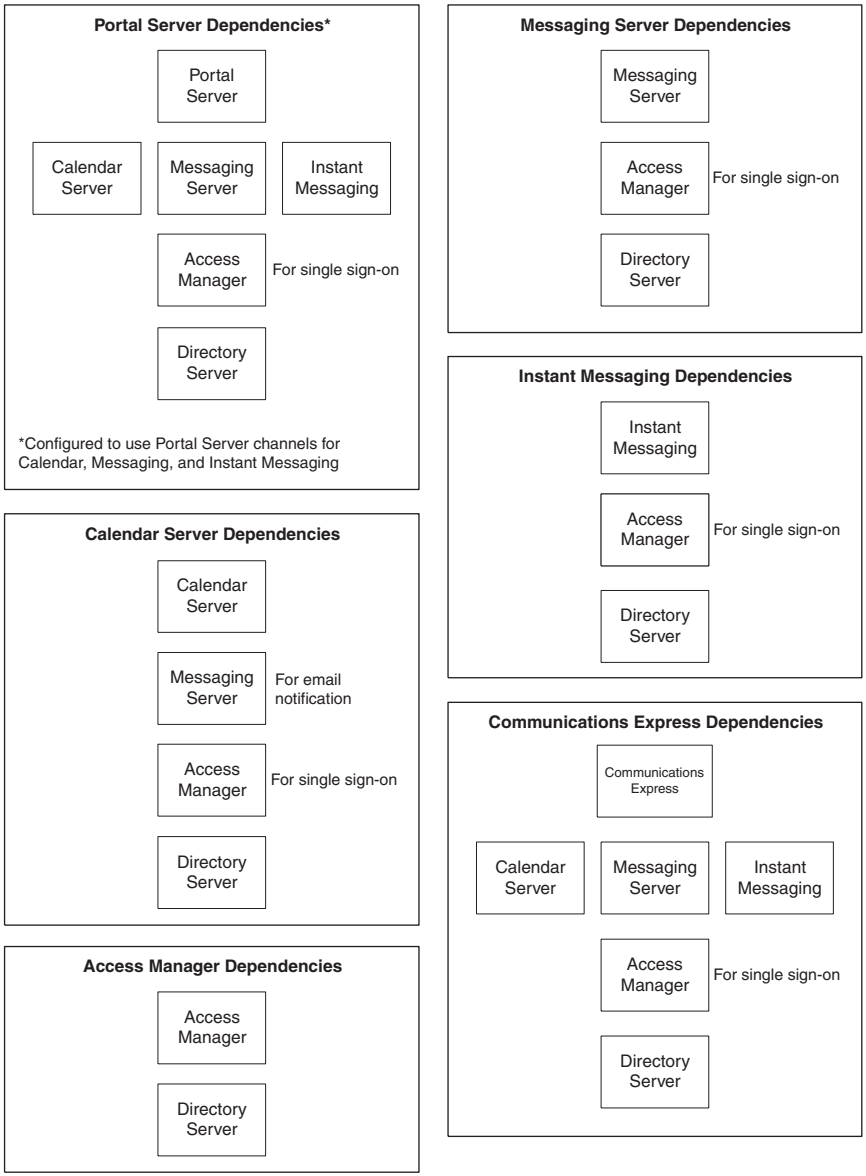


FIGURE 4–2 Java Enterprise System Component Dependencies

The previous section, "Component Dependencies" on page 48 does not account for the web container in which Portal Server and Access Manager run. This web container can be provided

by Application Server, Web Server, or a third-party product. When designing a logical architecture that includes Portal Server or Access Manager be sure to account for the web container required for these components.

Logically Distinct Services Provided by Messaging Server

The Java Enterprise System Messaging Server can be configured to provide separate instances that provide the following logically distinct services:

- Message Transfer Agent
- Message Multiplexor
- Message Express Multiplexor
- Message Store

These various configurations of Messaging Server provide functionality that can be deployed on separate physical servers and can be represented in different tiers of a logical architecture. Because these configurations for Messaging Server represent logically distinct services in separate tiers, consider them as logically distinct components when designing a logical architecture. The section "Example Logical Architectures" on page 53 provides an example of logically distinct components.

The following table describes the logically distinct configurations of Messaging Server.

TABLE 4–2 Messaging Server Configurations

Subcomponent	Description
Message Transfer Agent (MTA)	Supports the sending of email by handling SMTP connections, routing emails, and delivering messages to the proper message stores. The MTA components can be configured to support email sent from outside the enterprise (inbound) or sent from within the enterprise (outbound).
Message Store (STR)	Provides for the retrieval and storage of email messages.
Message Multiplexor (MMP)	Supports the retrieval of email by accessing the message store for email clients, using either IMAP or POP protocols.
Messenger Express Multiplexor (MEM)	Supports the retrieval of email by accessing the message store on behalf of web- based (HTTP) clients.

Access Components

Java Enterprise System also contains components that provide access to system services, often from outside an enterprise firewall. Some configurations of Messaging Server can also provide

network access, such as Messaging Server configured for message multiplexor. The following table describes Java Enterprise System components that provide remote access to system services.

TABLE 4–3 Java Enterprise System Components Providing Remote Access

Component	Description
Directory Proxy Server	Provides enhanced directory access, schema compatibility, routing, and load balancing for multiple Directory Server instances.
Portal Server, Portal Server Secure Remote Access	Provides secure Internet access from outside a corporate firewall to Portal Server content and services, including internal portals and Internet applications.
Portal Server, Portal Server Mobile Access	Provides wireless access from mobile devices and voice access to Portal Server.
Messaging Server Message Multiplexor (MMP)	Supports the retrieval of email by accessing the message store on behalf of web-based (HTTP) clients.

Components providing remote access are generally deployed in secures access zones, as illustrated by the example in the section "Access Zones" on page 61.

Multitiered Architecture Design

Java Enterprise System is well-suited for multitiered architecture design, where services are placed in tiers according to the functionality they provide. Each service is logically independent and can be accessed by services in either the same tier or a different tier. The following figure depicts a multitiered architecture model for enterprise applications, illustrating the client, presentation, business service, and data tiers.

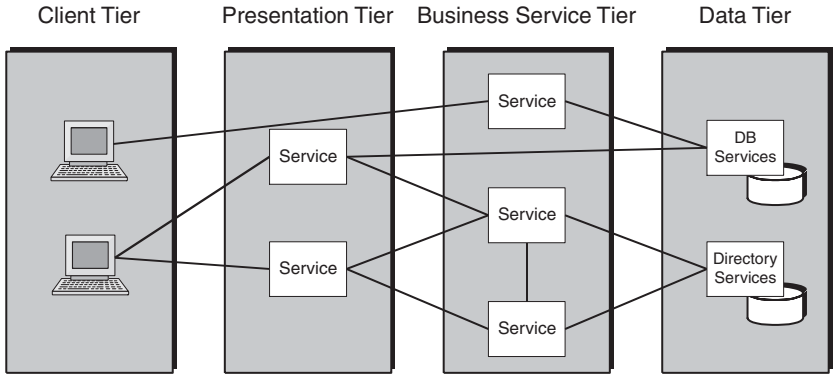


FIGURE 4–3 Multitiered Architecture Model

The following table describes the logical tiers depicted in "Multitiered Architecture Design" on page 52.

TABLE 4–4 Logical Tiers in a Multitiered Architecture

Tier	Description
Client tier	Contains client applications that present information to end users. For Java Enterprise System, these applications are typically mail clients, web browsers, or mobile access clients.
Presentation tier	Provides services that display data to end users, allowing users to process and manipulate the presentation. For example, a web mail client or Portal Server component allows users to modify the presentation of information they receive.
Business service tier	Provides back-end services that typically retrieve data from the data tier to provide to other services within the presentation or business service tiers or directly to clients in the client tier. For example, Access Manager provides identity services to other Java Enterprise System components.
Data tier	Provides database services accessed by services within the presentation tier or business service tier. For example, Directory Server provides LDAP directory access to other services.

Multitiered architecture design provides several advantages. During the deployment design phase, the placement of services according to functionality in a multitiered architecture helps you determine how to distribute services in your network. You also can see how components within the architecture access services of other components. This visualization helps you plan for availability, scalability, security, and other quality of service solutions.

Example Logical Architectures

This section provides some examples of logical architectures for Java Enterprise System solutions. These examples show how you place logical components within the appropriate tiers of a multitiered architecture and then analyze the relationships between the components by studying the use cases. Use the logical architectures examples in this section as a basis for understanding logical architecture design in Java Enterprise System solutions.

The first example is a basic Messaging Server solution that illustrates how the logically distinct components of Messaging Server interact with other components. The second example shows a logical architecture for an identity-based deployment solution that might be appropriate for a medium-sized enterprise of about 1,000 to 5,000 employees.

Messaging Server Example

The following figure shows a basic logical architecture for a deployment of Messaging Server. This logical architecture shows only the logically distinct components required for Messaging Server. Later figures illustrate the relationships among these components.

Note – Typically, a deployment of Messaging Server is part of an enterprise solution that includes other Java Enterprise System components, as illustrated in "Identity-Based Communications Example" on page 58.

The following table describes the components depicted in "Messaging Server Example" on page 54.

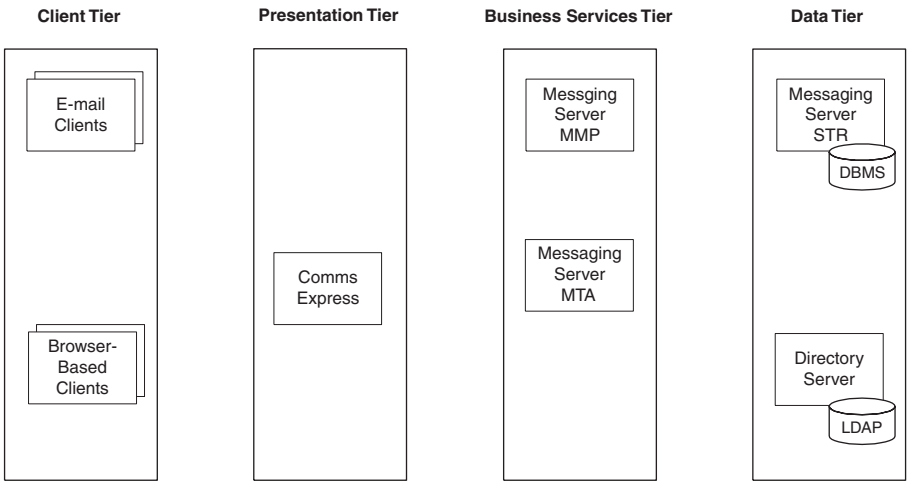


FIGURE 4–4 Logical Architecture for Messaging Server Deployment

TABLE 4–5 Components in Messaging Server Logical Architecture

Component	Description
Email clients	Client applications for reading and sending email.
Messaging Server MTA	Messaging Server configured as a message transfer agent (MTA) to receive, route, transport, and deliver email messages.
Messaging Server MMP	Messaging Server configured as a message multiplexor (MMP) to route connections to appropriate message stores for retrieval and storage. MMP accesses Directory Server to look up directory information to determine the proper message store.

TABLE 4–5 Components in Messaging Server Logical Architecture *(Continued)*

Component	Description
Messaging Server STR	Messaging Server configured as a message store for retrieval and storage of email messages.
Directory Server	Provides access to LDAP directory data.

The logical architecture does not specify replication of services for the Messaging Server components. For example, enterprise deployments typically create separate inbound and outbound MTA instances but "Messaging Server Example" on page 54 shows only one MTA component. The replication of logical components into multiple instances is a design decision that you make during the deployment design phase.

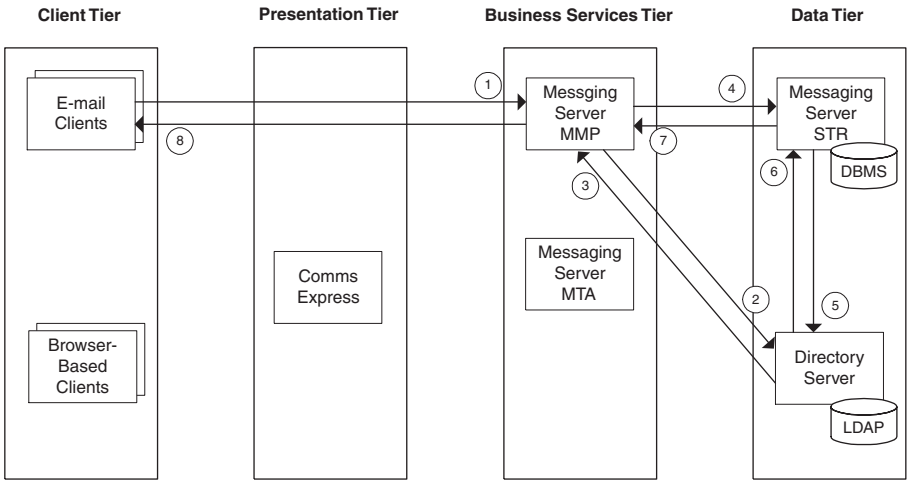
Messaging Server Use Cases

Use cases help identify the relationships among the logical components in an architecture. By mapping the interactions between the components according to the use cases, you get a visual picture of component interaction that is helpful in deployment design.

Typically, you analyze each use case to determine the interaction of components prior to deployment design. The following three use cases are typical for Messaging Server and show interactions among the logical components.

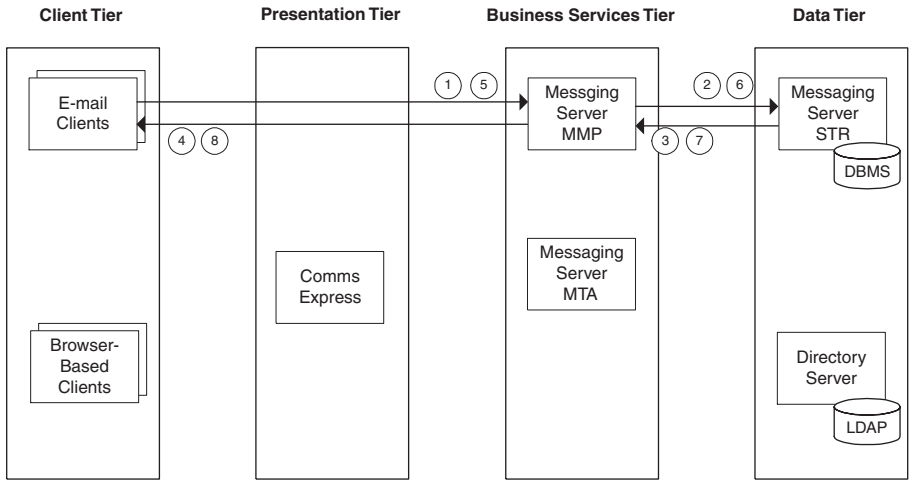
▼ Use Case 1: User Logs in Successfully to Messaging Server

1. **Email client sends login information to Messaging Server Multiplexor (MMP)**
2. **MMP requests verification of user ID and password from Directory Server.**
3. **Directory Server returns verification to MMP.**
4. **MMP requests message list from Messaging Server Message Store (STR).**
5. **STR requests user's LDAP record from Directory Server.**
6. **Directory Server returns user's LDAP record to STR.**
7. **STR returns message list to MMP.**
8. **MMP forwards message list to email client.**



Use Case 2: Logged-In User Reads and Deletes Mail

1. Email client requests message to read from Messaging Server Multiplexor (MMP).
2. MMP requests message from Messaging Server Message Store (STR).
3. STR returns message to MMP.
4. MMP forward message to email client.
5. Email client sends deletes message action to MMP.
6. MMP forwards delete message action to STR.
7. STR deletes message from database and sends confirmation to MMP.
8. MMP forwards delete confirmation to email client.



▼ Use Case 3: Logged-In User Sends Email Message

1 **Email client sends message composed in client to Messaging Server Message Transfer Agent (MTA).**

2 **MTA requests verification of user ID and password from Directory Server.**

3 **Directory Server returns verification to MTA.**

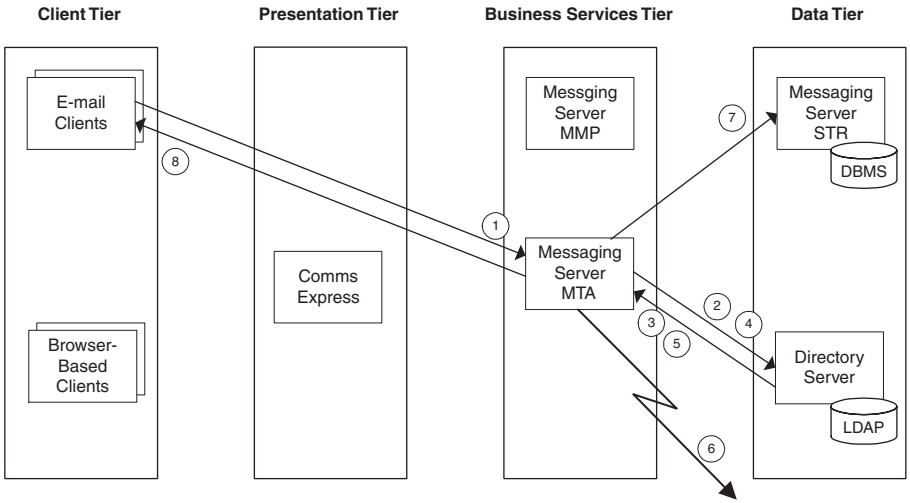
4 **MTA checks Directory Server for the destination domain for each recipient.**

5 **Directory Server returns to MTA the destination domain for each recipient.**

6 **MTA forwards message to each recipient.**

7 **MTA forwards message to Messaging Server Message Store (STR) to store message in outbox.**

8 **MTA sends confirmation to email client.**



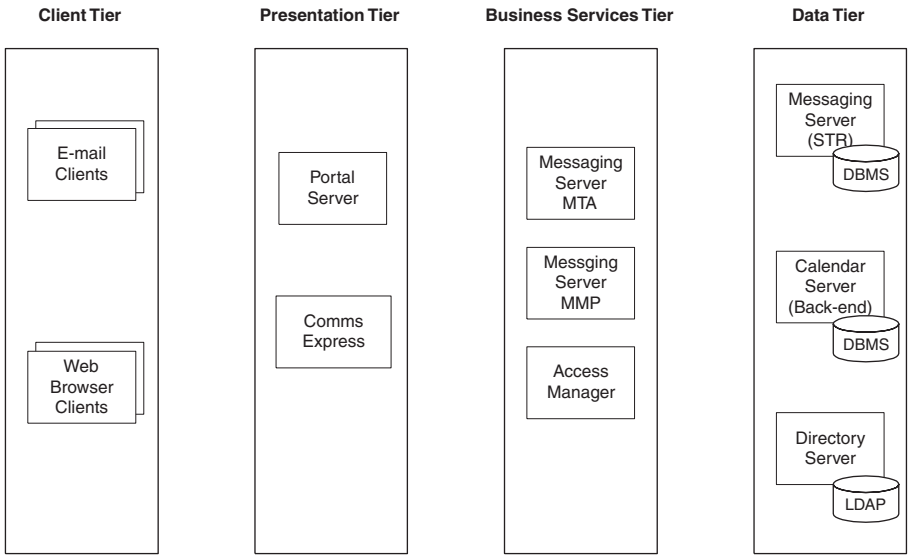
Identity-Based Communications Example

This example illustrates an identity-based communications solution for a medium-sized enterprise of about 1,000 to 5,000 employees. Typically, an exhaustive business analysis followed by detailed technical requirements analysis is needed to design the logical architecture. However, because this is a theoretical example, assume that the following business requirements have been determined:

- Employees of the enterprise require personalized access to internal web sites, communications services, calendar services, and other resources.
- Enterprise-wide authentication and authorization provide access to the internal web sites and other services.
- Single identity is tracked across all enterprise services, enabling a single sign-on (SSO) that provides access to the internal websites and other services.

Use cases for this example would detail login procedures, reading email, sending email, personalizing the portal, synchronizing calendars, and other similar user activities.

The following figure shows a logical architecture for this type of identity-based communications solution.



Use Cases for Identity-Based Communications Example

For a deployment solution of this nature, there typically are numerous detailed use cases outlining the user interaction with the services provided by the solution. This example focuses on the interaction among components when a user logs into a portal from a web browser client. The example splits this login scenario into two use cases:

- User logs in, becomes authenticated, and Portal Server retrieves the user's portal configuration.
- Portal Server retrieves email and calendar information to display in the web client.

The two use cases can be considered one extended use case. However, for this example, the use cases are separated for simplicity.

▼ Use Case 1: User Logs in Successfully and Portal Retrieves User's Configuration

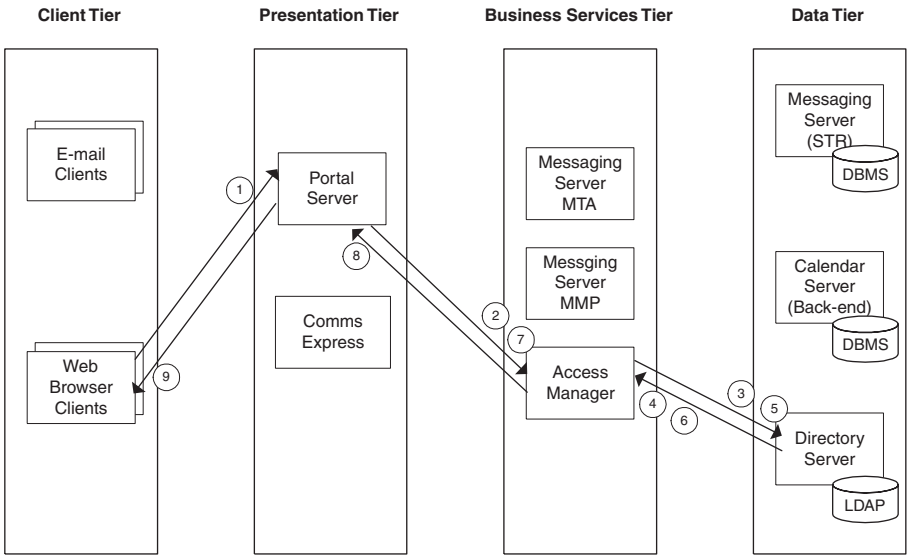
1. **Web browser client sends user ID and password to Portal Server.**
2. **Portal Server requests authentication from Access Manager.**
3. **Access Manager requests verification of user ID and password from Directory Server.**
4. **Directory Server verifies user ID and password.**
5. **Access Manager requests user profile from Directory Server.**

6 Directory Server returns user profile.

7 Portal Server requests user display profile from Access Manager.

8 Access Manager returns portal configuration.

9 Portal configuration is displayed in web browser client.



▼ Use Case 2: Portal Server Displays Email and Calendar Information

1 After successful log in, authentication, and retrieval of portal configuration, Portal Server requests email messages from Messaging Server MMP.

2 MMP requests message list from Messaging Server STR.

3 STR returns message list to MMP.

4 MMP forwards message headers to Portal Server.

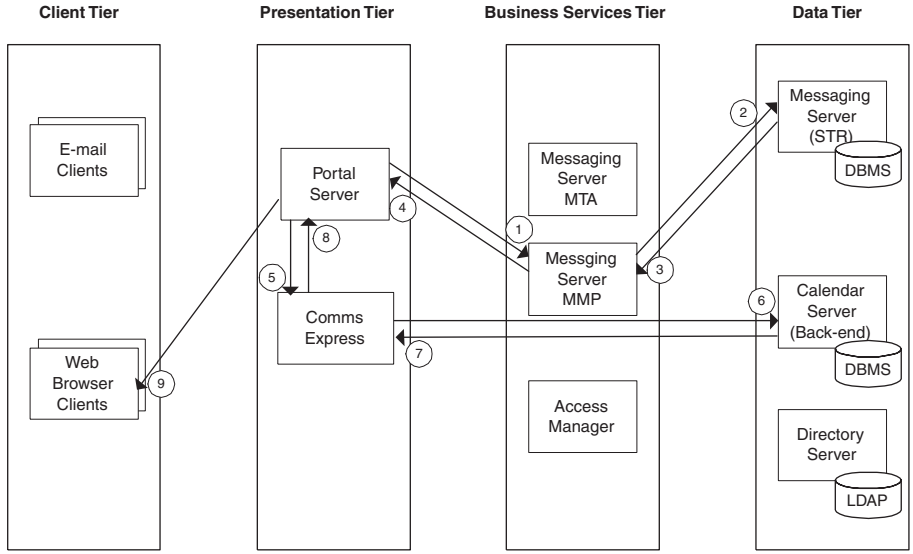
5 Portal Server requests calender information from Communications Express.

6 Communications Express requests calendar information from Calendar Server backend.

7 Calendar Server backend returns calendar information to Communications Express.

8 Communications Express forwards calendar information to Portal Server.

9 Portal Server sends all channel information to web browser client.



Access Zones

Another way to represent the components of a logical architecture is to place them in access zones that show how the architecture provides secure access. The following figure illustrates access zones for deploying Java Enterprise System components. Each access zone shows how components provide secure remote access to and from the Internet and intranet.

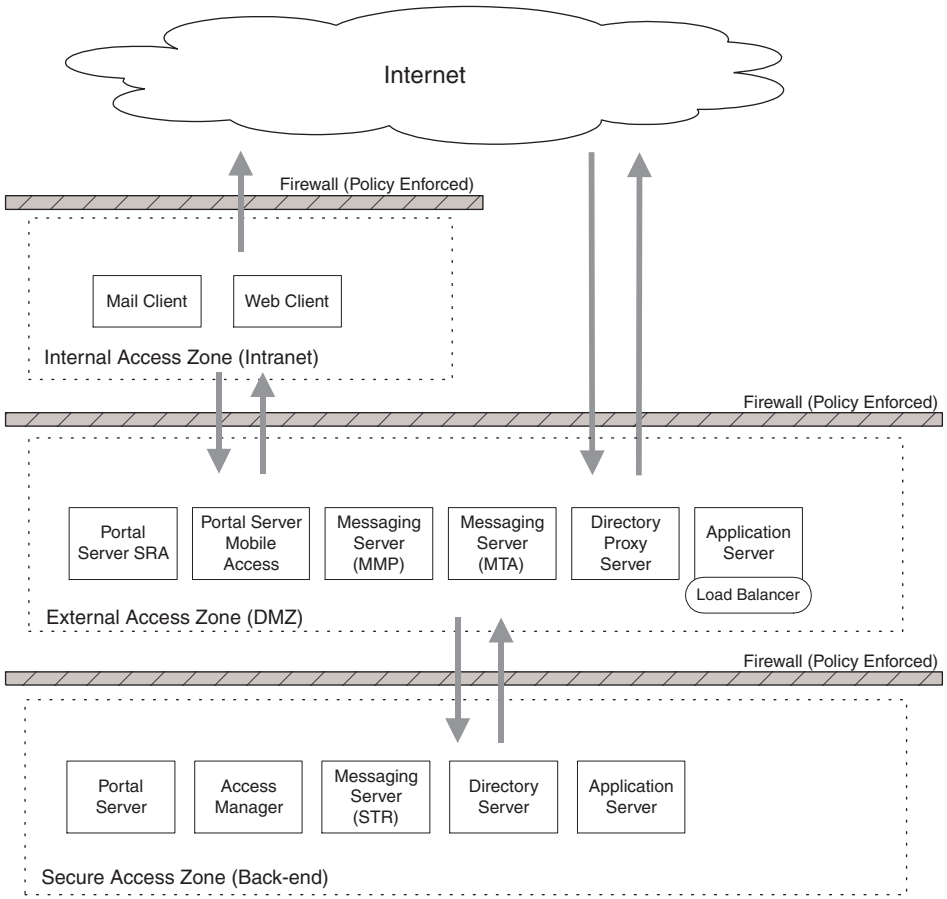


FIGURE 4–5 Logical Components Placed in Access Zones

The following table describes the access zones depicted in "Access Zones" on page 61.

TABLE 4–6 Secure Access Zones and Components Placed Within Them

Access Zone	Description
Internal access zone(Intranet)	Access to the Internet through policies enforced by a firewall between the intranet and the Internet. The Internal access zone is typically used by end users for web browsing and for sending email. In some cases, direct access to the Internet for web-browsing is allowed. However, typically secure access to and from the Internet is provided through the external access zone.

TABLE 4–6 Secure Access Zones and Components Placed Within Them *(Continued)*

Access Zone	Description
External access zone(DMZ)	Provides secure access to and from the Internet, acting as a security buffer to critical back-end services.
Secure access zone(Back-end)	Provides restricted access to critical back-end services, which can only be accessed from the external access zone.

"Access Zones" on page 61 does not illustrate the logical tiers depicted in the previous examples, but instead focuses on which components provide remote and internal access, the relationship of these components to security measures such as firewalls, and a visual depiction of access rules that must be enforced. Use the multi-tier architecture design in combination with the design showing access zones to provide a logical model of your planned deployment.

Deployment Scenario

The completed logical architecture design by itself is not sufficient to move forward to the deployment design phase of the solution life cycle. You need to pair the logical architecture with the quality of service (QoS) requirements determined during the technical requirements phase. The pairing of the logical architecture with the QoS requirements constitutes a deployment scenario. The deployment scenario is the starting point for designing the deployment architecture, as explained in Chapter 5.

◆ ◆ ◆ CHAPTER 5

Deployment Design

During the deployment design phase of the solution life cycle, you design a high-level deployment architecture and a low-level implementation specification, and prepare a series of plans and specifications necessary to implement the solution. Project approval occurs in the deployment design phase.

This chapter contains the following sections:

About Deployment Design

Deployment design begins with the deployment scenario created during the logical design and technical requirements phases of the solution life cycle. The deployment scenario contains a logical architecture and the quality of service (QoS) requirements for the solution. You map the components identified in the logical architecture across physical servers and other network devices to create a deployment architecture. The QoS requirements provide guidance on hardware configurations for performance, availability, scalability, and other related QoS specifications.

Designing the deployment architecture is an iterative process. You typically revisit the QoS requirements and reexamine your preliminary designs. You take into account the

interrelationship of the QoS requirements, balancing the trade-offs and cost of ownership issues to arrive at an optimal solution that ultimately satisfies the business goals of the project.

Project Approval

Project approval occurs during the deployment design phase, generally after you have created the deployment architecture. Using the deployment architecture and possibly also implementation specifications described below, the actual cost of the deployment is estimated and submitted to the stakeholders for approval. Once the project is approved, contracts for completion of the deployment are signed and resources to implement the project are acquired and allocated.

Deployment Design Outputs

During the deployment design phase, you might prepare any of the following specifications and plans:

- **Deployment architecture.** A high-level architecture that depicts the mapping of a logical architecture to a physical environment. The physical environment includes the computing nodes in an intranet or Internet environment, processors, memory, storage devices, and other hardware and network devices.
- **Implementation specifications.** Detailed specifications used as a blueprint for building the deployment. These specifications provide specifics on the computer and network hardware to acquire and describe the network layout for the deployment. Implementation specifications also include specifications for directory services, including details on a directory information tree (DIT) and the groups and roles defined for directory access.
- **Implementation plans.** A group of plans that cover various aspects of implementing an enterprise software solution. Implementation plans include the following:
  - *Migration plan.* Describes the strategies and processes for migrating enterprise data and upgrading enterprise software. The migrated data must conform to the formats and standards of the newly installed enterprise applications. All enterprise software must be at correct release version levels to interoperate.
  - *Installation plan.* Derived from the deployment architecture, specifies hardware server names, installation directories, installation sequence, types of installation for each node, and the configuration information necessary to install and configure a distributed deployment.
  - *User management plan.* Includes migration strategies for data in existing directories and databases, directory design specifications that takes into account replication design specified in the deployment architecture, and procedures for provisioning directories with new content.

- *Test plan.* Describes the procedures for testing the deployed software, including specific plans for developing prototype and pilot implementations, stress tests that determine the ability to handle projected loads, and functional tests that determine if planned functionality operates as expected.
- *Roll-out plan.* Describes the procedures and schedule for moving the implementation from a planning and test environment to a production environment. Moving an implementation into production usually occurs in various phases. For example, the first phase might be deploying the software for a limited group of users and increasing the user base with each phase until the entire deployment is complete. Phased implementation can also include scheduled implementation of specific software packages until the entire deployment is complete.
- *Disaster recovery plan.* Describes procedures on how to restore the system from unexpected system-wide failures. The recovery plan includes procedures for both large scale and small scale failures.
- *Operations plan (Run Book).* A manual of operations that describes monitoring, maintenance, installation, and upgrade procedures.
- *Training plan.* Contains processes and procedures for training operators, administrators, and end users on the newly installed enterprise software.

Factors Affecting Deployment Design

Several factors influence the decisions you make during deployment design. Consider the following key factors:

- **Logical Architecture.** The logical architecture details the functional services in a proposed solution and the interrelationships of the components providing those services. Use the logical architecture as a key to determining the best way to distribute services. A deployment scenario contains the logical architecture paired with quality of service requirements (described below).
- **Quality of service requirements.** The quality of service (QoS) requirements specify various aspects of a solution's operation. Use the QoS requirements to help develop strategies to achieve performance, availability, scalability, serviceability, and other quality of service goals. A deployment scenario contains the logical architecture (described previously) paired with quality of service requirements.
- **Usage analysis.** Usage analysis, developed during the technical requirements phase of the solution life cycle, provides information on usage patterns that can help estimate load and stress on a deployed system. Use the usage analysis to help isolate performance bottlenecks and develop strategies to satisfy QoS requirements.

- **Use cases.** Use cases, developed during the technical requirements phase of the solution life cycle, lists distinct user interactions identified for a deployment, often identifying the most common use cases. Although the use cases are embodied in the usage analysis, when assessing a deployment design you should refer to the use cases to make sure that they are properly addressed.
- **Service level agreements.** A service level agreement (SLA) specifies minimum performance requirements, and when those requirements are not met, the level and extent of customer support that must be provided. A deployment design should easily meet the performance requirements specified in a service level agreement.
- **Total cost of ownership.** During deployment design you analyze potential solutions that address the QoS requirements for availability, performance, scalability, and others. However, for each solution you consider, you must also consider the cost of that solution and how that cost impacts the total cost of ownership. Make sure that you consider the trade-offs embodied by your decisions and that you have optimized your resources to achieve business requirements within business constraints.
- **Business goals.** Business goals are stated during the business analysis phase of the solution life cycle and include the business requirements and business constraints to meet those goals. Deployment design is ultimately judged by its ability to satisfy the business goals.

Deployment Design Methodology

As with other aspects of deployment planning, deployment design is as much an art as it is a science and cannot be detailed with specific procedures and processes. Factors that contribute to successful deployment design are past design experience, knowledge of systems architecture, domain knowledge, and applied creative thinking.

Deployment design typically revolves around achieving performance requirements while meeting other QoS requirements. The strategies you use must balance the trade-offs of your design decisions to optimize the solution. The methodology you use typically involves the following tasks:

- **Estimating processor requirements.** Deployment design often begins with estimating the number of CPUs needed for each component in the logical architecture. Start with the use cases representing the heaviest load and continue through each use case. Consider the load on all components providing support to the use cases, and modify your estimates accordingly. Also consider any previous experience you have with designing enterprise systems.
- **Estimating processor requirements for secure transport.** Study the use cases that require secure transport and modify CPU estimates accordingly.
- **Replicating services for availability and scalability.** Once you are satisfied with the processor estimates, make modifications to the design to account for QoS requirements for availability and scalability. Consider load balancing solutions that address availability and failover considerations.

During your analysis, consider the trade-offs of your design decisions. For example, what affect does the availability and scalability strategy have on serviceability (maintenance) of the system? What are the others costs of the strategies?

- **Identifying bottlenecks.** As you continue with your analysis, examine the deployment design to identify any bottlenecks that cause the transmission of data to fall beneath requirements, and make adjustments.
- **Optimizing resources.** Review your deployment design for resource management and consider options that minimizes costs while fulfilling requirements.
- **Managing risks.** Revisit your business and technical analyses with respect to your design, making modifications to account for events or situations that might not have been foreseen in the earlier planning.

Estimating Processor Requirements

This section discusses a process for estimating the number of CPU processors and corresponding memory that are necessary to support the services in a deployment design. The section includes a walkthrough of an estimation process for an example communications deployment scenario.

The estimation of CPU computing power is an iterative process that considers the following:

- Logical components and their interactions (as indicated by component dependencies in the logical architecture)
- Usage analysis for the identified use cases
- Quality of service requirements
- Past experience with deployment design and with Java Enterprise System
- Consultation with Sun professional services who have experience with designing and implementing various types of deployment scenarios

The estimation process includes the following steps. The ordering of these steps is not critical, but provides one way to consider the factors that affect the final result.

1. Determine a baseline CPU estimate for components identified as user entry points to the system.

   One design decision is whether to fully load or partially load CPUs. Fully loaded CPUs maximize the capacity of a system. To increase the capacity, you incur the maintenance cost and possible downtime of adding additional CPUs. In some cases, you can choose to add additional machines to meet growing performance requirements.

   Partially loaded CPUs allow room to handle excess performance requirements without immediately incurring maintenance costs. However, there is an additional up front expense of the under-utilized system.
2. Make adjustments to the CPU estimates to account for interactions between components.

Study the interactions among components in the logical architecture to determine the extra load required because of dependent components.

3. Study the usage analysis for specific use cases to determine peak loads for the system, and then make adjustments to components that handle the peak loads.

   Start with the most heavily weighted use cases (those requiring the most load), and continue with each use case to make sure you account for all projected usage scenarios.

4. Make adjustments to the CPU estimates to reflect security, availability, and scalability requirements.

This estimation process provides starting points for determining the actual processing power you need. Typically, you create prototype deployments based on these estimates and then perform rigorous testing against expected use cases. Only after iterative testing can you determine the actual processing requirements for a deployment design.

Example Estimating Processor Requirements

This section illustrates one methodology to estimate processing power required for an example deployment. The example deployment is based on the logical architecture for the identity-based communications solution for a medium-sized enterprise of about 1,000 to 5,000 employees, as described in the section "Identity-Based Communications Example" on page 58.

The CPU and memory figures used in the example are arbitrary estimates for illustration only. These figures are based on arbitrary data upon which the theoretical example is based. An exhaustive analysis of various factors is necessary to estimate processor requirements. This analysis would include, but not be limited to, the following information:

- Detailed use cases and usage analysis based on an exhaustive business analysis
- Quality of service requirements determined by analysis of business requirements
- Specific costs and specifications of processing and networking hardware
- Past experience implementing similar deployments

Caution – The information presented in these examples do not represent any specific implementation advice, other than to illustrate a process you might use when designing a system.

Determine Baseline CPU Estimate for User Entry Points

Begin by estimating the number of CPUs required to handle the expected load on each component that is a user entry point. The following figure shows the logical architecture for an identity-based communications scenario described previously in "Identity-Based Communications Example" on page 58.

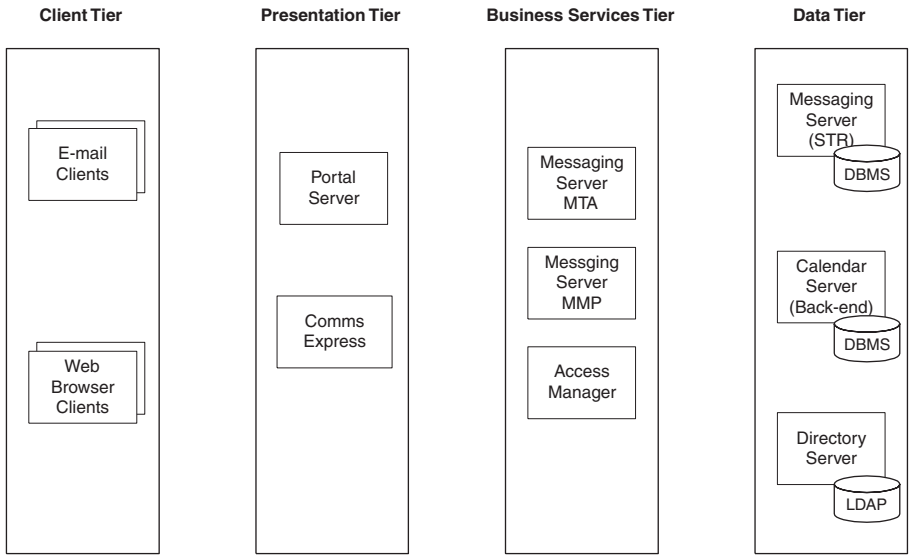


FIGURE 5–1 Logical Architecture for Identity-Based Communications Scenario

The following table lists the components in the presentation tier of the logical architecture that interface directly with end users of the deployment. The table includes baseline CPU estimates derived from analysis of technical requirements, use cases, specific usage analysis, and past experience with this type of deployment.

TABLE 5–1 CPU Estimates for Components Containing Access User Entry Points

Component	Number of CPUs	Description
Portal Server	4	Component that is a user entry point.
Communications Express	2	Routes data to Portal Server messaging and calendar channels.

Include CPU Estimates for Service Dependencies

The components providing user entry points require support from other Java Enterprise System components. As you continue to specify performance requirements, add the performance estimates required for supporting components. The type of interactions among components should be detailed when designing the logical architecture, as described in the logical architecture examples in the section "Example Logical Architectures" on page 53.

TABLE 5–2 CPU Estimates for Supporting Components

Component	CPUs	Description
Messaging Server MTA(inbound)	1	Routes incoming mail messages from Communications Express and e-mail clients.
Messaging Server MTA(outbound)	1	Routes outgoing mail messages to recipients.
Messaging Server MMP	1	Access Messaging Server message store for email clients.
Messaging Server STR(Message Store)	1	Retrieves and stores email messages.
Access Manager	2	Provides authorization and authentication services.
Calendar Server(back-end)	2	Retrieves and stores calendar data for Communications Express, a Calendar Server front-end.
Directory Server	2	Provides LDAP directory services.
Web Server	0	Provides web container support for Portal Serverand Access Manager. (No additional CPU cycles required.)

Study Use Cases for Peak Load Usage

Return to the use cases and usage analysis to identify areas of peak load usage and make adjustments to your CPU estimates.

For example, suppose for this example you identify the following peak load conditions:

- Initial ramp up of users as they log on simultaneously
- Email exchanges during specified time frames

To account for this peak load usage, make adjustment to the components providing these services. The following table outlines adjustments you might make to account for this peak load usage.

TABLE 5–3 CPU Estimate Adjustments for Peak Load

Component	CPUs (Adjusted)	Description
Messaging Server MTAinbound	2	Add 1 CPU for peak incoming email
Messaging Server MTAoutbound	2	Add 1 CPU for peak outgoing email
Messaging ServerMMP	2	Add 1 CPU for additional load
Messaging Server STR(Message Store)	2	Add 1 CPU for additional load

TABLE 5–3 CPU Estimate Adjustments for Peak Load *(Continued)*

Component	CPUs (Adjusted)	Description
Directory Server	3	Add 1 CPU for additional LDAP lookups

Modify Estimates for Other Load Conditions

Continue with your CPU estimates to take into account other quality of service requirements that can impact load:

- **Security.** From the technical requirements phase, determine how secure transport of data might affect the load requirements and make corresponding modifications to your estimates. The following section, "Estimating Processor Requirements for Secure Transactions" on page 74 describes a process for making adjustments.
- **Replication of services.** Adjust CPU estimates to account for replication of services for availability, load balancing, and scalability considerations. The following section, "Determining Availability Strategies" on page 76, discusses sizing for availability solutions. The section "Determining Strategies for Scalability" on page 84 discusses solutions involving available access to directory services.
- **Latent capacity and scalability.** Modify CPU estimates as necessary to allow latent capacity for unexpected large loads on the deployment. Look at the anticipated milestones for scaling and projected load increase over time to make sure you can reach any projected milestones to scale the system, either horizontally or vertically.

Update the CPU Estimates

Typically, you round up CPUs to an even number. Rounding up to an even number allows you to evenly split the CPU estimates between two physical servers and also adds a small factor for latent capacity. However, round up according to your specific needs for replication of services.

As a general rule, allow 2 gigabytes of memory for each CPU. The actual memory required depends on your specific usage and can be determined in testing.

The following table lists the final estimates for the identity-based communications example. These estimates do not include any additional computing power that could have been added for security and availability. Totals for security and availability will be added in following sections.

TABLE 5–4 CPU Estimate Adjustments for Supporting Components

Component	CPUs	Memory
Portal Server	4	8 GB
Communications Express	2	4 GB
Messaging Server(MTA, inbound)	2	4 GB

TABLE 5–4 CPU Estimate Adjustments for Supporting Components *(Continued)*

Component	CPUs	Memory
Messaging Server(MTA, outbound)	2	4 GB
Messaging Server(MMP)	2	4 GB
Messaging Server(Message Store)	2	4 GB
Access Manager	2	4 GB
Calendar Server	2	4 GB
Directory Server	4	8 GB (Rounded up from 3 CPUs/6 GB memory)
Web Server	0	0

Estimating Processor Requirements for Secure Transactions

Secure transport of data involves handling transactions over a secure transport protocol such as Secure Sockets Layer (SSL) or Transport Layer Security (TLS). Transactions handled over a secure transport typically require additional computing power to first, establish a secure session (known as the handshake) and then to encrypt and decrypt transported data. Depending on the encryption algorithm used (for example, 40-bit or 128-bit encryption algorithms), the additional computing power can be substantial.

For secure transactions to perform at the same level as nonsecure transactions, you must plan for additional computing power. Depending on the nature of the transaction and the Sun Java™ Enterprise System services that handle it, secure transactions might require up to four times more computing power than nonsecure transactions.

When estimating the processing power to handle secure transactions, analyze use cases to determine the percentage of transactions that require secure transport. If the performance requirements for secure transactions are the same as for non-secure transactions, modify the CPU estimates to account for the additional computing power needed for the secure transactions.

In some usage scenarios, secure transport might only be required for authentication. Once a user is authenticated to the system, no additional security measures for transport of data is required. In other scenarios, secure transport might be required for all transactions.

For example, when browsing a product catalog for an online e-commerce site, all transactions can be nonsecure until the customer has finished making selections and is ready to “check out” to make a purchase. However, some usage scenarios, such as deployments for banks or brokerage houses, require most or all, transactions to be secure and apply the same performance standard for both secure and nonsecure transactions.

CPU Estimates for Secure Transactions

This section continues the example deployment to illustrate how to calculate CPU requirements for a theoretical use case that includes both secure and nonsecure transactions.

To estimate the CPU requirements for secure transactions, make the following calculations:

1. Start with a baseline figure for the CPU estimates (as illustrated in the previous section, "Example Estimating Processor Requirements" on page 70).
2. Calculate the percentage of transactions that require secure transport, and calculate the CPU estimates for the secure transactions.
3. Calculate reduced CPU estimates for non-secure transactions.
4. Tally the secure estimate and nonsecure estimate to calculate the total CPU estimates.
5. Round up the total CPU estimate to an even number.

"CPU Estimates for Secure Transactions" on page 75 shows an example calculation based on use cases and usage analysis for the Portal Server that assume the following:

- All logins require secure authentication.
- All logins account for 10% of the total Portal Server load.
- The performance requirement for secure transactions is the same as the performance requirement for non-secure transactions.

  To account for the extra computing power to handle secure transactions, the number of CPUs to handle these transactions will be increased by a factor of four. As with other CPU figures in the example, this factor is arbitrary and is for illustration purposes only.

TABLE 5–5 Modifying CPU Estimates for Secure Transactions

Step	Description	Calculation	Result
1	Start with baseline estimate for all Portal Server transactions.	Baseline estimate from "Study Use Cases for Peak Load Usage" on page 72 is 4 CPUs.	- - - - -

2	Calculate additional CPU estimates for secure transactions. Assume secure transactions require four times the CPU power as nonsecure transactions.	Ten percent of the baseline estimate require secure transport:
		`0.10 x 4 CPUs = 0.4 CPUs`
		Increase CPU power for secure transactions by a factor of four:
		`4 x 0.4 = 1.6 CPUs`
			1.6 CPUs

3	Calculate reduced CPU estimates for nonsecure transactions.	Ninety percent of the baseline estimate are non-secure: `0.9 x 4 CPUs = 3.6 CPUs`	3.6 CPUs

TABLE 5–5 Modifying CPU Estimates for Secure Transactions *(Continued)*

Step	Description	Calculation	Result
4	Calculate adjusted total CPU estimates for secure and nonsecure transactions.	Secure estimate + non-secure estimate = total: `1.6 CPUs + 3.6 CPUs = 5.2 CPUs`	5.2 CPUs
5	Round up to even number.	`5.2 CPUs ==> 6 CPUs`	6 CPUs

From the calculations for secure transactions in this example, you would modify the total CPU estimates in "CPU Estimates for Secure Transactions" on page 75 by adding an additional two CPUs and four gigabytes of memory to get the following total for Portal Server.

Component	CPUs	Memory
Portal Server	6	12 GB

Specialized Hardware to Handle SSL Transactions

Specialized hardware devices, such as SSL accelerator cards and other appliances, are available to provide computing power to handle establishment of secure sessions and the encryption and decryption of data. When using specialized hardware for SSL operations, computational power is dedicated to some part of the SSL computations, typically the "handshake" operation that establishes a secure session.

This hardware might be of benefit to your final deployment architecture. However, because of the specialized nature of the hardware, estimate secure transaction performance requirements first in terms of CPU power, and then consider the benefits of using specialized hardware to handle the additional load.

Some factors to consider when using specialized hardware are whether the use cases support using the hardware (for example, use cases that require a large number of SSL handshake operations) and the added layer of complexity this type of hardware brings to the design. This complexity includes the installation, configuration, testing, and administration of these devices.

Determining Availability Strategies

When developing a strategy for availability requirements, study the component interactions and usage analysis to determine which availability solutions to consider. Do your analysis on a component-by-component basis, determining a best-fit solution for availability and failover requirements.

The following items are examples of the type of information you gather to help determine availability strategies:

- How many nines of availability are specified?
- What are the performance specifications with respect to failover situations (for example, at least 50% of performance during failover)?
- Does the usage analysis identify times of peak and non-peak usage?
- What are the geographical considerations?

The availability strategy you choose must also take into consideration serviceability requirements, as discussed in "Designing for Optimum Resource Usage" on page 88. Avoid complex solutions that require considerable administration and maintenance.

Availability Strategies

Availability strategies for Java Enterprise System deployments include the following:

- *Load balancing.* Uses redundant hardware and software components to share a processing load. A load balancer directs any requests for a service to one of multiple symmetric instances of the service. If any one instance should fail, other instances are available to assume a heavier load.
- *Failover.* Involves managing redundant hardware and software to provide continuous access of services and security for critical data if any component fails.

  Sun Cluster software provides a failover solution for critical data managed by back-end components such as the message storage for Messaging Server and calendar data for Calendar Server.
- *Replication of services.* Replication of services provides multiple sources for access to the same data. Directory Server provides numerous replication and synchronization strategies for LDAP directory access.

The following sections provide some examples of availability solutions that provide various levels of load balancing, failover, and replication of services.

Single Server System

Place all computing resources for a service on a single server. If the server fails, the entire service fails.

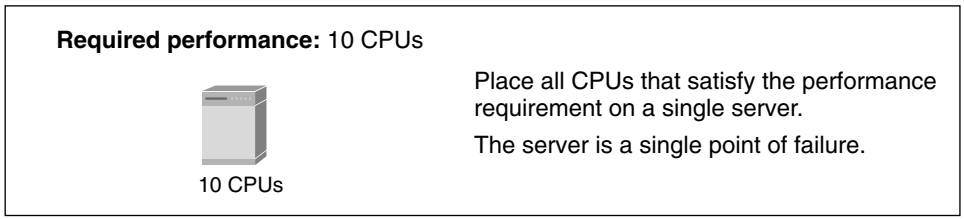


FIGURE 5–2 Single Server System

Sun provides high-end servers that provide the following benefits:

- Replacement and reconfiguration of hardware components while the system is running
- Ability to run multiple applications in fault-isolated domains on the server
- Ability to upgrade capacity, performance speed, and I/O configuration without rebooting the system

A high-end server typically costs more than a comparable multi-server system. However, a single server provides savings on administration, monitoring, and hosting costs for servers in a data center. Load balancing, failover, and removal of single points of failure is more flexible with multi-server systems.

Horizontally Redundant Systems

There are several ways to increase availability with parallel redundant servers that provide both load balancing and failover. The following figure illustrates two replicate servers providing an N+1 failover system. An N+1 system has an additional server to provide 100% capacity should one server fail.

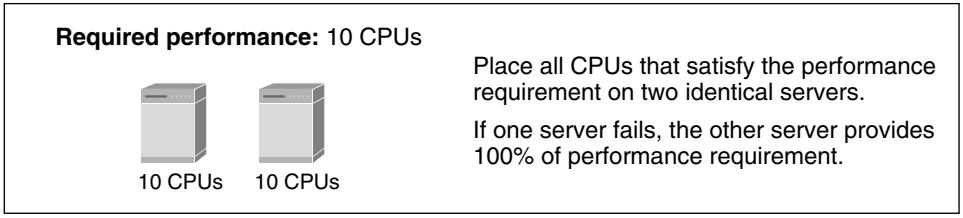


FIGURE 5–3 N+1 Failover System With Two Servers

The computing power of each server in "Horizontally Redundant Systems" on page 78 above is identical. One server alone handles the performance requirements. The other server provides 100% of the performance when called into service as a backup.

The advantage of an N+1 failover design is 100% performance during a failover situation. Disadvantages include increased hardware costs with no corresponding gain in overall performance (because one server is a standby for use in failover situations only).

The following figure illustrates a system that implements load balancing plus failover that distributes the performance between two servers.

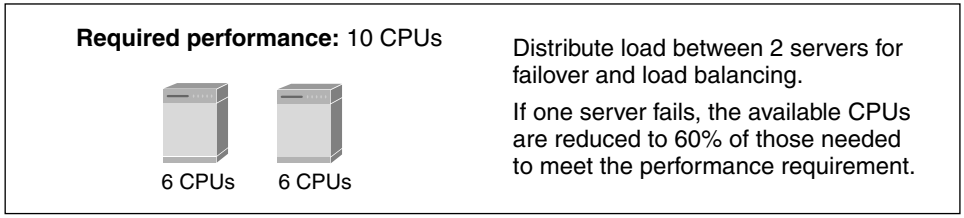


FIGURE 5–4 Load Balancing Plus Failover Between Two Servers

In the system depicted in "Horizontally Redundant Systems" on page 78 above, if one server fails, all services are available, although at a percentage of the full capacity. The remaining server provides 6 CPUs of computing power, which is 60% of the 10 CPU requirement.

An advantage of this design is the additional 2 CPU latent capacity when both servers are available.

The following figure illustrates a distribution between a number of servers for performance and load balancing.

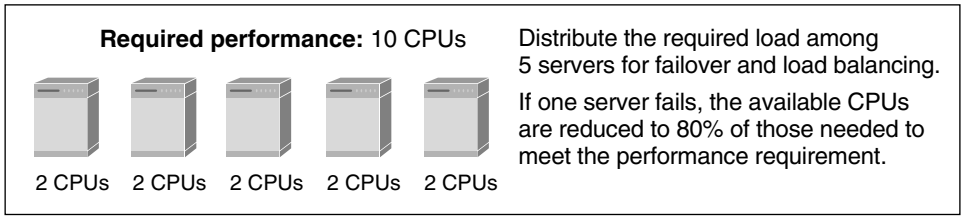


FIGURE 5–5 Distribution of Load Between *n* Servers

Because there are five servers in the design depicted in "Horizontally Redundant Systems" on page 78, if one server fails the remaining servers provide a total of 8 CPUs of computing power, which is 80% of the 10 CPU performance requirement. If you add an additional server with a 2-CPU capacity to the design, you effectively have an N+1 design. If one server fails, 100% of the performance requirement is met by the remaining servers.

This design includes the following advantages:

- Added performance if a single server fails
- Availability even when more than one server is down
- Servers can be rotated out of service for maintenance and upgrades
- Multiple low-end servers typically cost less than a single high-end server

However, administration and maintenance costs can increase significantly with additional servers. You also have to consider costs for hosting the servers in a data center. At some point you run into diminishing returns by adding additional servers.

Sun Cluster Software

For situations that require a high degree of availability (such as four or five nines), you might consider Sun Cluster software as part of your availability design. A cluster system is the coupling of redundant servers with storage and other network resources. The servers in a cluster continually communicate with each other. If one of the servers goes offline, the remainder of the devices in the cluster isolate the server and fail over any application or data from the failing node to another node. This failover process is achieved relatively quickly with little interruption of service to the users of the system.

Sun Cluster software requires additional dedicated hardware and specialized skills to configure, administer, and maintain.

Availability Design Examples

This section contains two examples of availability strategies based on the identity-based communications solution for a medium-sized enterprise of about 1,000 to 5,000 employees, as described previously in "Identity-Based Communications Example" on page 58. The first availability strategy illustrates load balancing for Messaging Server. The second illustrates a failover solution that uses Sun Cluster software.

Load Balancing Example for Messaging Server

The following table lists the estimates for CPU power for each logical Messaging Server component in the logical architecture. This table repeats the final estimation calculated in the section "Update the CPU Estimates" on page 73 .

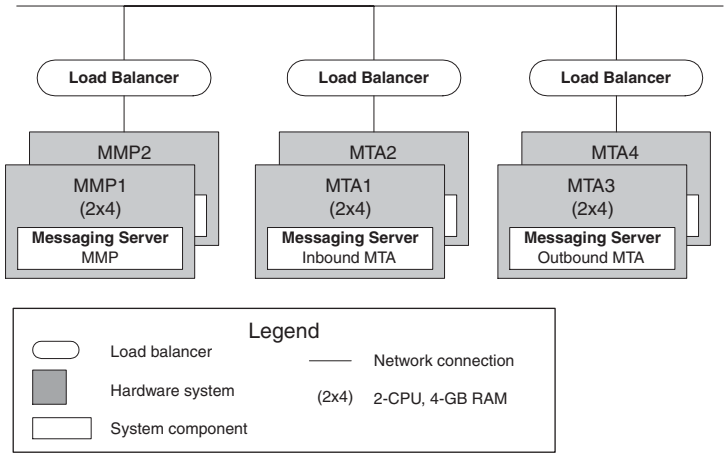
TABLE 5–6 CPU Estimate Adjustments for Supporting Components

Component	CPUs	Memory
Messaging Server(MTA, inbound)	2	4 GB
Messaging Server(MTA, outbound)	2	4 GB
Messaging Server(MMP)	2	4 GB
Messaging Server(Message Store)	2	4 GB

For this example, assume that during technical requirements phase, the following quality of service requirements were specified:

- **Availability.** Overall system availability should be 99.99% (does not include scheduled downtime). Failure of an individual computer system should not result in service failure.
- **Scalability.** No server should be more than 80% utilized under daily peak load and the system must accommodate long-term growth of 10% per year.

To fulfill the availability requirement, for each Messaging Server component provide two instances, one of each on separate hardware servers. If a server for one component fails, the other provides the service. The following figure illustrates the network diagram for this availability strategy.



In the preceding figure the number of CPUs has doubled from the original estimate. The CPUs are doubled for the following reasons:

- In the event one server fails, the remaining server provides the CPU power to handle the load.
- For the scalability requirement that no single server is more than 80% utilized under peak load, the added CPU power provides this safety margin.
- For the scalability requirement to accommodate 10% increased load per year, the added CPU power adds latent capacity that can handle increasing loads until additional scaling would be needed.

Failover Example Using Sun Cluster Software

The following figure shows an example of failover strategy for Calendar Server back-end and Messaging Server messaging store. The Calendar Server back-end and messaging store are replicated on separate hardware servers and configured for failover with Sun Cluster software. The number of CPUs and corresponding memory are replicated on each server in the Sun Cluster.

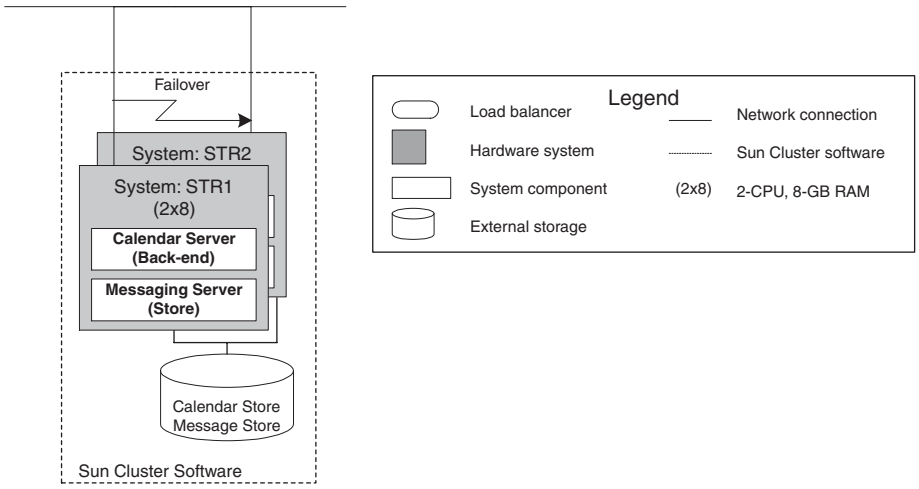


FIGURE 5–6 Failover Design Using Sun Cluster Software

Replication of Directory Services Example

Directory services can be replicated to distribute transactions across different servers, providing high availability. Directory Server provides various strategies for replication of services, including the following:

- **Multiple databases.** Stores different portions of a directory tree in separate databases.
- **Chaining and referrals.** Links distributed data into a single directory tree.
- **Single master replication.** Provides a central source for the master database, which is then distributed to consumer replicas.
- **Multi-master replication.** Distributes the master database among several servers. Each of these masters then distributes their database among consumer replicas.

Availability strategies for Directory Server is a complex topic that is beyond the scope of this guide. The following sections, "Single Master Replication" on page 82 and "Multi-Master Replication" on page 83 provide a high-level view of basic replication strategies. For detailed information see Chapter 12, "Designing a Highly Available Deployment," in *Sun Java System Directory Server Enterprise Edition 6.0 Deployment Planning Guide*.

Single Master Replication

The following figure shows a single master replication strategy that illustrates basic replication concepts.

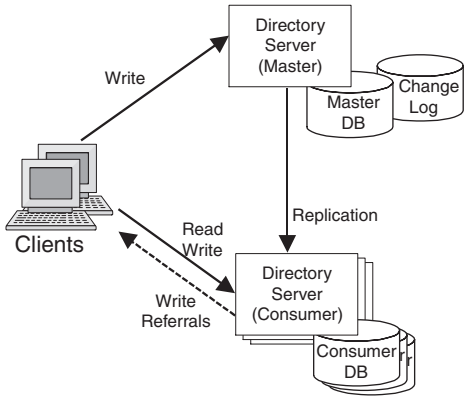


FIGURE 5–7 Single Master Replication Example

In single master replication, one instance of Directory Server manages the master directory database, logging all changes. The master database is replicated to any number of consumer databases. The consumer instances of Directory Server are optimized for read and search operations. Any write operation received by a consumer is referred back to the master. The master periodically updates the consumer databases.

Advantages of single master replication include:

- Single instance of Directory Server optimized for database read and write operations
- Any number of consumer instances of Directory Server optimized for read and search operations
- Horizontal scalability for consumer instances of Directory Server

Multi-Master Replication

The following figure shows a multi-master replication strategy that might be used to distribute directory access globally.

In multi-master replication, one or more instances of Directory Server manages the master directory database. Each master has a replication agreement that specifies procedures for synchronizing the master databases. Each master replicates to any number of consumer databases. As with single master replication, the consumer instances of Directory Server are optimized for read and search access. Any write operation received by a consumer is referred back to the master. The master periodically updates the consumer databases.

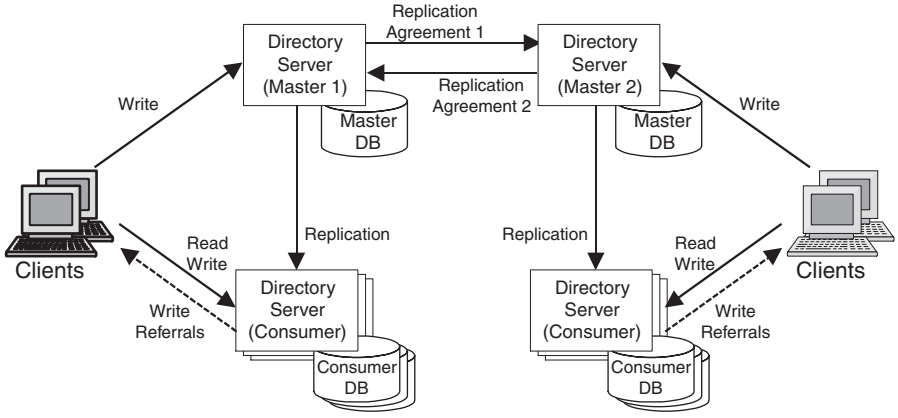


FIGURE 5–8 Multi-master Replication Example

Multi-master replication strategy provides all the advantages of single master replication, plus an availability strategy that can provide load balancing for updates to the masters. You can also implement an availability strategy that provides local control of directory operations, which is an important consideration for enterprises with globally distributed data centers.

Determining Strategies for Scalability

Scalability is the ability to add capacity to your system, usually by the addition of system resources, but without changes to the deployment architecture. During requirements analysis, you typically make projections of expected growth to a system based on the business requirements and subsequent usage analysis. These projections of the number of users of a system and the capacity of the system to meet their needs are often estimates that can vary significantly from the actual numbers for the deployed system. Your design should be flexible enough to allow for variance in your projections.

A design that is scalable includes sufficient latent capacity to handle increased loads until a system can be upgraded with additional resources. Scalable designs can be readily scaled to handle increasing loads without redesign of the system.

Latent Capacity

Latent capacity is one aspect of scalability where you include additional performance and availability resources into your system so the system can easily handle unusual peak loads. You can also monitor how latent capacity is used in a deployed system to help determine when to scale the system by adding resources. Latent capacity is one way to build safety into your design.

Analysis of use cases can help identify the scenarios that can create unusual peak loads. Use this analysis of unusual peak loads plus a factor to cover unexpected growth to design latent capacity that builds safety into your system.

Your system design should be able to handle projected capacity for a reasonable time, generally the first 6 to 12 months of operation. Maintenance cycles can be used to add resources or increase capacity as needed. Ideally, you should be able to schedule upgrades to the system on a regular basis, but predicting needed increases in capacity is often difficult. Rely on careful monitoring of your resources as well as business projections to determine when to upgrade a system.

If you plan to implement your solution in incremental phases, you might schedule increasing the capacity of the system to coincide with other improvements scheduled for each incremental phase.

Scalability Example

The example in this section illustrates horizontal and vertical scaling for a solution that implements Messaging Server. For vertical scaling, you add additional CPUs to a server to handle increasing loads. For horizontal scaling, you handle increasing loads by adding additional servers for distribution of the load.

The baseline for the example assumes a 50,000 user base supported by two message store instances that are distributed for load balancing. Each server has two CPUs for a total of four CPUs. The following figure shows how this system can be scaled to handle increasing loads for 250,000 users and 2,000,000 users.

Note – "Scalability Example" on page 85 shows the differences between vertical scaling and horizontal scaling. This figure does not show other factors to consider when scaling, such as load balancing, failover, and changes in usage patterns.

Identifying Performance Bottlenecks

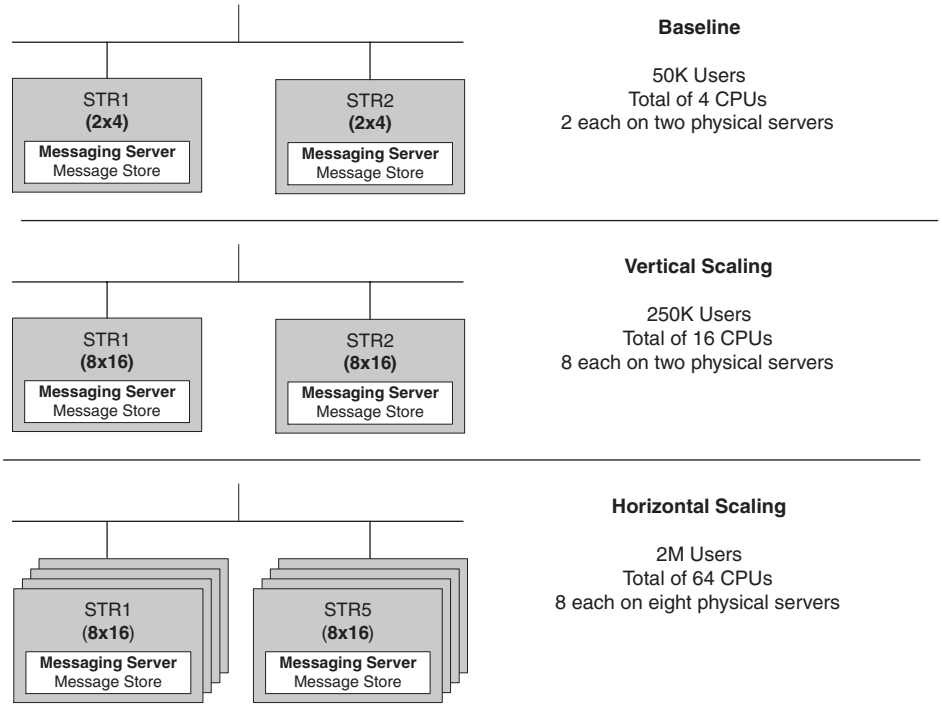


FIGURE 5–9 Horizontal and Vertical Scaling Examples

One of the keys to successful deployment design is identifying potential performance bottlenecks and developing a strategy to avoid them. A performance bottleneck occurs when the rate at which data is accessed cannot meet specified system requirements.

Bottlenecks can be categorized according to various classes of hardware, as listed in the following table of data access points within a system. This table also suggests potential remedies for bottlenecks in each hardware class.

TABLE 5–7 Data Access Points

Hardware Class	Relative Access Speed	Remedies for Performance Improvement
Processor	Nanoseconds	Vertical scaling: Add more processing power, improve processor cache Horizontal scaling: Add parallel processing power for load balancing

TABLE 5–7 Data Access Points *(Continued)*

Hardware Class	Relative Access Speed	Remedies for Performance Improvement
System memory (RAM)	Microseconds	Dedicate system memory to specific tasks
		Vertical scaling: Add additional memory
		Horizontal scaling: Create additional instances for parallel processing and load balancing
Disk read and write	Milliseconds	Optimize disk access with disk arrays (RAID)
		Dedicate disk access to specific functions, such as read only or write only
		Cache frequently accessed data in system memory
Network interface	Varies depending on bandwidth and access speed of nodes on the network	Increase bandwidth
		Add accelerator hardware when transporting secure data
		Improve performance on nodes within the network so the data is more readily available

Note – "Identifying Performance Bottlenecks" on page 85 lists hardware classes according to relative access speed, implying that slow access points, such as disks, are more likely to be the source of bottlenecks. However, processors that are underpowered to handle large loads are also likely sources of bottlenecks.

You typically begin deployment design with baseline processing power estimates for each component in the deployment and their dependencies. You then determine how to avoid bottlenecks related to system memory and disk access. Finally, you examine the network interface to determine potential bottlenecks and focus on strategies to overcome them.

Optimizing Disk Access

A critical component of deployment design is the speed of disk access to frequently accessed datasets, such as LDAP directories. Disk access provides the slowest access to data and is a likely source of a performance bottleneck.

One way to optimize disk access is to separate write operations from read operations. Not only are write operations more expensive than read operations, read operations (lookup operations for LDAP directories) typically occur with considerably more frequency than write operations (updates to data in LDAP directories).

Another way to optimize disk access is by dedicating disks to different types of I/O operations. For example, provide separate disk access for Directory Server logging operations, such as transaction logs and event logs, and LDAP read and write operations.

Also, consider implementing one or more instances of Directory Server dedicated to read and write operations and using replicated instances distributed to local servers for red and search access. Chaining and linking options are also available to optimize access to directory services.

Chapter 6, "Tuning System Characteristics and Hardware Sizing," in *Sun Java System Directory Server Enterprise Edition 6.0 Deployment Planning Guide* discusses various factors in planning for disk access. Topics in this chapter include:

- **Minimum memory and disk space requirements.** Provides estimates for disk and memory needed for various sizes of directories.
- **Sizing physical memory for cache access.** Provides guidance on estimating cache size according to planned usage of Directory Server and on planning total memory usage.
- **Sizing disk subsystems.** Provides information on planning disk space requirements according to directory suffixes and Directory Server factors that affect disk use. and distributing files across disks, including various disk array alternatives.

Designing for Optimum Resource Usage

Deployment design is not just estimating the resources required to meet the QoS requirements. During deployment design you also analyze all available options and select the best solution that minimizes cost but still fulfills QoS requirements. You must analyze the trade-off for each design decision to make sure a benefit in one area is not offset by a cost in another.

For example, horizontal scaling for availability might increase overall availability, but at the cost of increased maintenance and service. Vertical scaling for performance might increase computing power inexpensively, but the additional power might be used inefficiently by some services.

Before completing your design strategy, examine your decisions to make sure that you have balanced the use of resources with the overall benefit to the proposed solution. This analysis typically involves examining how system qualities in one area affect other system qualities. The following table lists some system qualities and corresponding considerations for resource management.

TABLE 5–8 Resource Management Considerations

System Quality	Description
Performance	For performance solutions that concentrate CPUs on individual servers, will the services be able to efficiently use the computing power? (For example, some services have a ceiling on the number of CPUs that can be efficiently used.)

TABLE 5–8 Resource Management Considerations *(Continued)*

System Quality	Description
Latent capacity	Does your strategy handle loads that exceed performance estimates?
	Are excessive loads handled with vertical scaling on servers, load balancing to other servers, or both?
	Is the latent capacity sufficient to handle unusual peak loads until you reach the next milestone for scaling the deployment?
Security	Have you sufficiently accounted for the performance overhead required to handle secure transactions?
Availability	For horizontally redundant solutions, have you sufficiently estimated long-term maintenance expenses?
	Have you accounted for the scheduled downtime necessary to maintain the system?
	Have you balanced the costs between high-end servers and low-end servers?
Scalability	Have you estimated milestones for scaling the deployment?
	Do you have a strategy to provide enough latent capacity to handle projected increases in load until you reach the milestones for scaling the deployment?
Serviceability	Have you taken into account administration, monitoring, and maintenance costs into your availability design?
	Have you considered delegated administration solutions (allowing end-users to perform some administration tasks) to reduce administration costs?

Managing Risks

Much of the information on which deployment design is based, such as quality of service requirements and usage analysis, is not empirical data but data based on estimates and projections ultimately derived from business analyses. These projections could be inaccurate for may reasons, including unforeseen circumstances in the business climate, faulty methods of gathering data, or simply human error. Before completing a deployment design, revisit the analyses upon which your design is based and make sure your design accounts for any reasonable deviations from the estimates or projections.

For example, if the usage analysis underestimates the actual usage of the system, you run the risk of building a system that cannot cope with the amount of traffic it encounters. A design that under performs will surely be considered a failure.

On the other hand, if you build a system that is several orders more powerful than required, you divert resources that could be used elsewhere. The key is to include a margin of safety above the requirements, but to avoid extravagant use of resources.

Extravagant use of resources results in a failure of the design because underutilized resources could have been applied to other areas. Additionally, extravagant solutions might be perceived by stakeholders as not fulfilling contracts in good faith.

Example Deployment Architecture

The following figure represents a completed deployment architecture for the example deployment introduced earlier in this white paper. This figure provides an idea of how to present a deployment architecture.

Caution – The deployment architecture in the following figure is for illustration purposes only. It does not represent a deployment that has been actually designed, built, or tested and should not be considered as deployment planning advice.

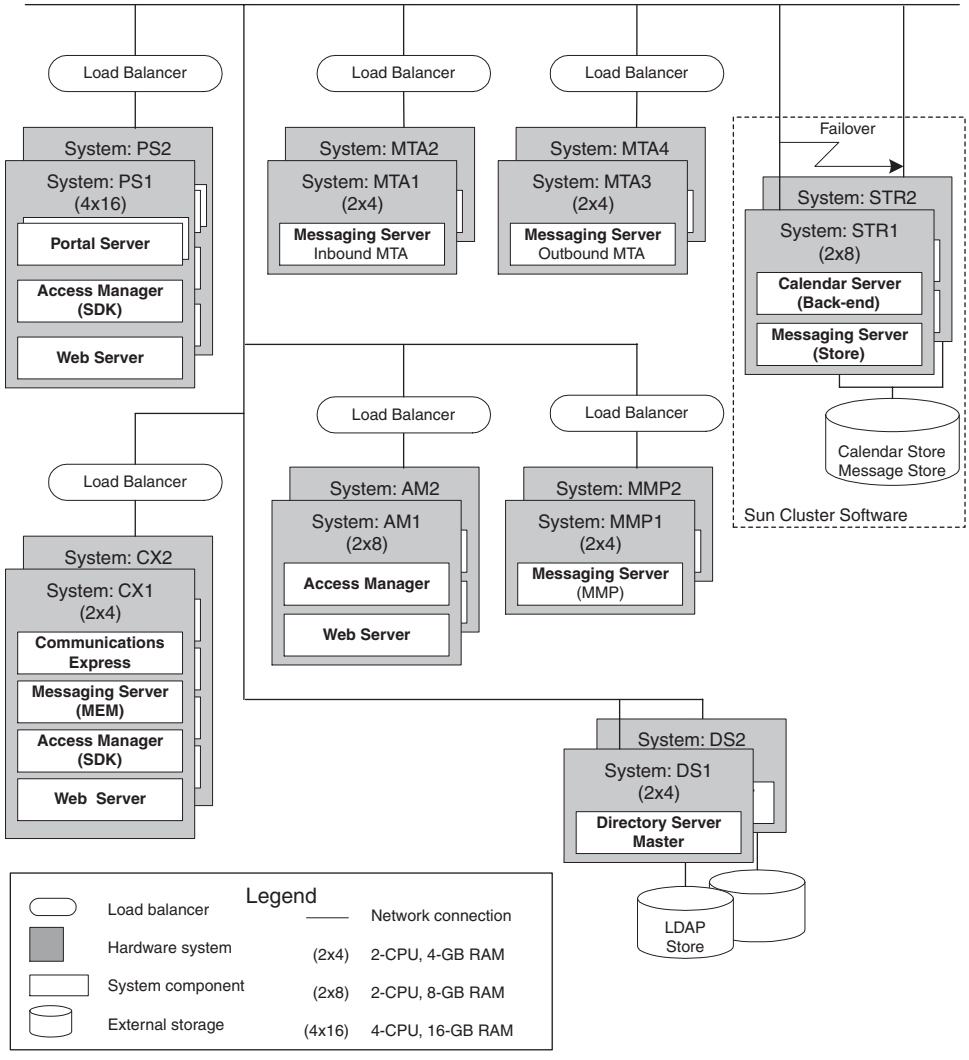


FIGURE 5–10 Example Deployment Architecture

◆ ◆ ◆ CHAPTER 6

Implementation of a Deployment Design

During the implementation phase of the solution life cycle you work from specifications and plans created during deployment design to build and test the deployment architecture, ultimately rolling out the deployment into production. Implementation is beyond the scope of this guide, however this chapter provides a high-level view of this phase.

This chapter contains the following sections:

About Implementing Deployment Designs

After the deployment architecture has been approved and implementation specifications and plans have been completed, you enter the implementation phase of the solution life cycle. Implementation is a complex set of processes and procedures that requires careful planning to ensure success. Implementation includes the following tasks:

- Building the network and hardware infrastructure
- Installing and configuring software according to an installation plan
- Migrating data from existing applications to the current solution
- Implementing a user management plan
- Designing and deploying pilots or prototypes in a test environment according to a test plan
- Designing and running functional tests and stress tests according to a test plan
- Rolling out the solution from a test environment to a production environment according to a rollout plan
- Training administrators and users of the deployment according to a training plan

Details of implementation are beyond the scope of this guide. However, the following sections provide overview information for some of these tasks.

Installing and Configuring Software

The installation and configuration of Sun Java™ Enterprise System for a distributed enterprise application requires the planning and coordination of many tasks and procedures. During the deployment design phase, you create an installation plan based on the high-level deployment architecture that provides installation and configuration information needed to install Java Enterprise System software.

Highlights of this installation plan include:

- Determining the sequence and type of installation
- Surveying hosts for previously installed software and installation readiness
- Gathering configuration information for each Java Enterprise System component you are installing

The *Sun Java Enterprise System 5 Installation Planning Guide* provides details on how to gather information for an installation plan. The *Sun Java Enterprise System 5 Installation Reference for UNIX* provides detailed configuration information and worksheets you can use to document this information. The *Sun Java Enterprise System 5 Installation Guide for UNIX* provides guidance on common installation scenarios that involve multiple Java Enterprise System components. For more information, refer to Chapter 1, "Preparing for Installation," in *Sun Java Enterprise System 5 Installation Guide for UNIX*.

Developing Pilots and Prototypes

Java Enterprise System deployments typically fall into two categories, those based primarily on services provided with Java Enterprise System and those that require a significant number of custom services that are integrated with Java Enterprise System services. You can think of the former type of deployment as an 80:20 deployment (80% of the services are provided by Java Enterprise System) and similarly, the former as a 20:80 deployment.

For 80:20 deployments, during the implementation phase, you typically develop a pilot deployment for testing purposes. Because 80:20 deployments use mature Java Enterprise System services that provide "out-of-the-box" functionality, pilot deployments move relatively quickly from development, testing, and modification steps, to production deployments. A pilot deployment verifies the functionality of a solution, but also provides information on how well the system performs.

20:80 deployments, on the other hand, introduce new, custom services that do not have the history of interoperability that comes with 80:20 deployments. For this reason, you create a prototype, which is a proof-of-concept deployment that typically requires a more rigorous

development, testing, and modification cycle before going into production. A prototype lets you determine how well a proposed solution solves the problem in a test environment. Once the prototype demonstrates the functionality is sufficient, you can move on to more rigorous testing and then to a pilot deployment.

Note – Actual enterprise deployments can vary greatly in the amount of custom development of services they require. How you use pilot and prototype deployments for testing purposes depends on the complexity and nature of your deployment.

Testing Pilot and Prototype Deployments

The purpose of testing pilot and prototype deployments is to determine as best as possible under test conditions whether the deployment satisfies the system requirements and also meets the business goals.

Ideally, functional tests should model scenarios based on all identified use cases and a set of metrics should be developed to measure compliance. Functional testing can also involve a limited deployment to a select group of beta users to determine if business requirements are being satisfied.

Stress tests measure performance under peak loads. These tests typically use a series of simulated environments and load generators to measure throughput of data and performance. System requirements for the deployment are typically the basis for designing and passing stress tests.

Note – Functional and stress tests are particularly important for large deployments where system requirements might not be well-defined, there is no previous implementation on which to base estimates, and the deployment requires a significant amount of new development.

Testing can indicate problems with the deployment design specification and might involve several design, build, and test iterations before you can roll out the deployment to a production environment. When testing prototype deployments, you might discover problems with the deployment design, in which case you can iterate back to earlier phases in the solution life cycle to address the problems.

Make sure you have thoroughly tested your deployment design before proceeding to a pilot deployment. A pilot deployment indicates you have already verified the deployment design with earlier series of tests. Problems you uncover during the testing of a pilot deployment must generally be addressed within the parameters of the deployment design.

Because testing never completely simulates a production environment, and also because the nature of a deployed solution can evolve and change, you should continue to monitor deployed systems to identify any areas that require tuning, maintenance, or service.

Rolling Out a Production Deployment

Once the pilot or proof-of-concept deployment passes the test criteria, you are ready to roll out the deployment to a production environment. Typically, you roll out to a production environment in stages. A staged rollout is especially important for large deployments that affect a significant number of users.

The staged deployment can start with a small set of users and eventually expand the user base until the deployment is available to all users. A staged deployment can also start with a limited set of services and eventually phase in the remaining services. Staging services in phases can help isolate, identify, and resolve problems a service might encounter in a production environment.

Index

W