Contents

1 Introduction

Learning Goals

After completing this chapter, you will be able to:

- Define enterprise applications.
- Classify enterprise applications.
- Explain challenges in raising enterprise applications.
- Determine the key characteristics and applicability of software engineering methodologies.
- Explain the lifecycle of raising enterprise applications.
- Identify the stakeholders of enterprise applications.
- Describe the three key determinants of successful enterprise applications.
- Outline knowledge and skill areas required to raise enterprise applications.
- Determine the success of enterprise applications.

Welcome readers! Welcome to the stupendous journey of raising typical enterprise applications from their inception to rollout! In the last few decades, the software industry has traversed a long trail in greenfield (custom-built) enterprise applications development and witnessed various software engineering paradigms and methodologies. There is no fit-all universal solution to raise an enterprise application. This book is an attempt to take the readers through the various processes, life cycle stages, patterns, frameworks, tools, technologies and many more things which are required to raise enterprise applications that can cater to the business needs of today's enterprise.

This chapter lays the foundation for the rest of the book. Readers will understand the objectives, challenges and different types of enterprise applications. It introduces various software engineering paradigms and how the life cycle of enterprise applications progresses. Readers will learn a step-by-step approach for raising successful enterprise applications. Parameters to measure the success of enterprise applications are also discussed.

1.1 Enterprise Applications

Every software application is not designated as an enterprise application. The software applications, which are the DNA of organizations and imbibe the business functionalities of the enterprises to catalyze their growth, are termed as *enterprise applications*. They not only enhance the efficiency and productivity of the organization but also help in ensuring business continuity. The software application, which imbibes complex business logic, expected to be high on performance, fortified from vulnerabilities and attack vectors, is expected to handle large volumes of data and concurrent

users and scalable on need basis, easily maintainable and extendable and able to orchestrate with the overall enterprise application landscape of the organization is typically designated as enterprise application.

A typical enterprise application landscape has varieties of enterprise applications. They can be categorized from multiple perspectives. One of the categorizations is based upon the placement of the enterprise application in an organization in terms of their visibility to customers.

- Customer facing enterprise applications or front-end systems of an organization are referred to as *upstream* enterprise applications. For example, an "order capture" application that captures online orders of customers is an upstream enterprise application.
- Likewise, the back-end enterprise applications that work behind the scenes in an organization to fulfill the customers' or end users' needs are called *downstream* enterprise applications. For example, an "order provisioning" application can be considered as a downstream application, as it helps in fulfilling and provisioning the online orders captured through the "order capture" upstream enterprise application.
- There is a third category of applications in an organization that fulfills the general organizational needs, and can be referred to as *business enabler* enterprise applications, for example payroll and human resource management applications. Figure 1-1 represents a collection of upstream, downstream and business enabling enterprise applications in a typical financial enterprise landscape.

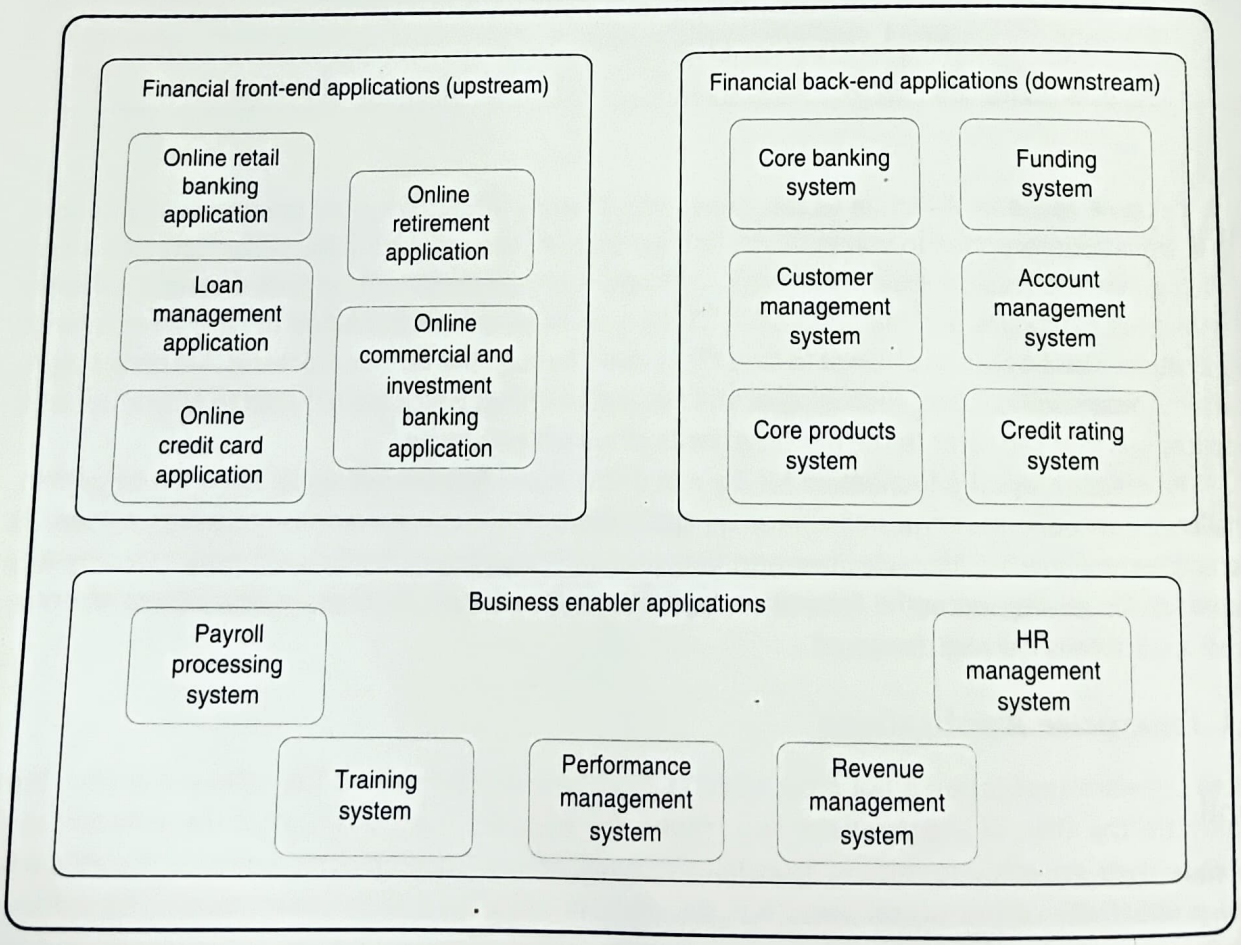


Figure 1-1 Upstream, downstream and business enabler enterprise applications.

Enterprise applications can also be categorized on the basis of industry they cater to. For example, a billing application for telecom is different from billing application for the retail industry. Enterprise applications can also be categorized based on the enterprise business functions they fulfill. For example, enterprise applications catering to order management functions or order fulfillment functions, etc.

Enterprise applications can be categorized based on parameters like nature of processing they perform. For example, enterprise applications may fall under categories such as batch processing, online transaction processing (OLTP) or online analytical processing (OLAP) applications or decision support systems (DSS). Enterprise applications can also be categorized as custom-built enterprise applications or readymade commercial off-the-shelf (COTS) packages. Enterprise applications can also be categorized as host-centric or distributed applications.

This is the era of "extended enterprises" and realization of the power of "selfservice." The business services and offerings are converging, markets are flexible, fluid and unpredictable, customers are getting more demanding and technologies are ever changing. This leads to complexities and challenges to enterprises for differentiating themselves from their competitors. Enterprise applications play a key role in meeting the business objectives. Let us explore the challenges faced by enterprise applications in order to keep up to the promises of today's enterprise.

Business processes automation

Enterprises are busy in achieving "flow through" by automating the enterprise business processes. "Flow through" is a mechanism to orchestrate the business processes spread across the organization to achieve the business process simplification. This helps organizations achieve quicker time-to-market, improved productivity and optimization of the resources. At the same time, enterprise applications are getting more and more complex to achieve such an integration and orchestration.

Data harmonization

Enterprises want to harmonize the data, which is a herculean task in itself depending upon the current state of the enterprise data. The biggest challenge is the lack of a "single source of truth" due to multiple versions, copies and representations of the same data in existence within the systems at any given point of time. This warrants the unification and standardization of data from the perspective of bringing together the disparate data and integration of systems in order to orchestrate the underlying enterprise business processes.

Application integration

In the last few decades, various enterprise applications are developed as silos. These applications are usually developed in isolation without considering the entire enterprise application landscape. In this context, application integration is one of the key challenges in front of the enterprises. As the enterprises are looking out for automating business processes, individual applications that host parts of these business processes are required to be integrated to realize the automation. The underlying diversity of platforms, different technologies and extent of integration pose challenges to application integration.

Application security

The attack surface of applications is broadening with more and more applications using the public Internet to expose themselves for access by users and other applications across the enterprise boundaries. With new attack vectors and security vulnerabilities creeping in, application security perimeter is ever increasing. The challenge is to fortify the security perimeter of enterprise application. This is

not only required to be compliant with the government regulations but also to maintain the enterprise brand value in the market.

Internationalization

Businesses are getting flatter, and the enterprises are on globalization spree. Enterprises want to reach out to their customers across the geographies using online mechanisms. The challenge is to build enterprise applications in order to serve the customers in a localized manner. Applications need to be built to serve customers in their local language and provide localized standards such as time, date and currency.

Transaction management

The online applications are ruling the enterprises and are contributing as the core infrastructure of the organizations. Vendor partners, suppliers, customers—both internal and external—and other stakeholders are accessing these applications to accomplish the business processes. Typically, a business process level transaction involves multiple system level transactions. The challenge is to consider these system level transactions as an atomic unit for identifying the success or failure of the overarching business process.

Rich user experience

Customers are getting more demanding and expect desktop like rich experience over the Internet. Providing such an engaging experience to the end user on browser is one of the challenges for enterprise applications.

Quality of Service (QoS)

The enterprise applications play a "mission-critical" role in the smooth functioning of the enterprises. The quality attributes like scalability, reliability, security, maintainability, availability etc. are inherent non-functional requirements for any enterprise application. These quality attributes are also termed as *ilities* of an enterprise application. As the complexity of applications is increasing, the challenge is to focus on core functionality along with the ilities of enterprise applications. QoS is to ensure the applications are "fit for purpose" and satisfy the stated and non-stated needs of end users and businesses. Applications that do not confirm to QoS typically result in lost productivity, revenue and goodwill.

Technology selection

Enterprise application development encompasses various platforms, frameworks and tools to select from the entire gamut of technologies—both commercial and open source ones. This comes with a challenge of not only keeping the team abreast with the technology spectrum but also to select the right set of technologies to build enterprise applications.

Governance and team productivity

Managing diverse teams and ensuring their productivity is one of the key challenges while raising successful enterprise applications. Strong governance is required to manage the team diversity. Teams have to develop the right skill sets and ensure effective reuse of existing solution elements in all the phases of application development to shorten the time-to-production of enterprise applications.

1.2 Software Engineering Methodologies

Software engineering is an engineering discipline that comprises of methodologies to develop, manage and maintain the software with focus on software quality, software requirements, software design, software development, software testing, software configuration management, etc.

Enterprise applications are developed predominantly using *iterative* methodology of software development. The traditional approach of software development is the *waterfall* methodology. It typically comprises of a sequence of phases—requirements, analysis, design, build and testing—wherein each phase output acts as input to the next phase. It typically takes more time-to-production, and the business value is realized only towards the end of a software development phase as against the iterative and agile approaches that are in vogue today, where the business values are realized in an incremental manner and time-to-production is relatively quicker.

Iterative software development approach is a cyclic approach to build software components in an incremental manner. Its planning involves choosing a subset of requirements to be implemented in each iteration. This way of development leverages the learning of the previous iterations in the next cycle to improve upon the target software component. The biggest benefit of using the iterative approach is risk mitigation. The waterfall methodology assumes the complete clarity of requirements. Consequently, the waterfall methodology follows the "big-design-upfront" (BDUF) approach which could lead to premature design decisions that may need to be reworked later. In reality, requirements evolve over a period of time which could also extend into the development stages of the life cycle. The design can also evolve in tune with the requirements and this ensures the better alignment of the design to the requirements.

IBM Rational Unified Process (RUP)[1] is one of the iterative software development processes. RUP has assembled the iterations in four phases: inception, elaboration, construction and transition. In each of the iterations, the unit of work is divided into nine disciplines. Six out of those nine are engineering disciplines namely business modeling, requirements, analysis and design, implementation, test and deployment. Other three are supporting disciplines namely configuration and change management, project management and environment.

Agile software development is an extension to the iterative approach to build applications in a nimble fashion with a light weight process. Typically in an agile project the iterations' duration is relatively smaller. These iterations are coupled with the short feedback loops to accommodate the changes quickly and appropriately guide the project forward. Rather than full-fledged design upfront, agile methodologies favor the incremental design improvements through techniques such as code refactoring and test-driven development. Code refactoring refers to improving the quality of the design through code modifications based on hindsight. Agile approach assumes a close involvement of the customer in the development process to provide continuous feedback as an application shapes up.

Extreme Programming (XP) and Scrum are two of the agile software development methodologies.[2] XP uses user stories for gathering requirements, CRC cards for just-in-time design and Pair Programming for coding. Scrum follows shorter iterations with 30-day release cycles, daily stand up meetings and concept of product back log.

The agile approaches are generally considered more suitable for small and mid-sized projects with teams consisting of more mature set of developers due to certain informal tendencies of the process

---

[1] To know more about IBM Rational Unified Process, visit http://www-01.ibm.com/software/awdtools/rup/

[2] To know more about XP methodology and Scrum methodology, visit http://www.extremeprogramming.org/ and http://scrummethodology.com/, respectively.

followed. In contrast, methodologies such as RUP are considered to be preferable for enterprise applications and projects involving large teams due to the higher rigor associated with the process.

In practice, most projects follow a hybrid approach which fosters the idea of fusing agile practices within the framework of more disciplined approaches like RUP.

1.3 Life Cycle of Raising Enterprise Applications

Raising an enterprise application exhibits a life cycle. Whatever be the software engineering paradigm used to raise an enterprise application, typically the life cycle of an enterprise application follows four phases—incepting, architecting and designing, constructing and testing, before it is rolled out for the enterprise users.

In a typical scenario, incepting an enterprise application starts as a result of enterprise analysis and business modeling activities. This is followed by the requirements engineering which typically include elicitation of requirements, and thereafter analysis of requirements, with respect to the enterprise application. Elicitation of requirements is done using use cases, prototypes or user stories. These requirements are validated to ensure their feasibility with respect to various factors including business requirements, budget, technology, etc. Incepting an enterprise application concludes with casting the plan and estimation of the project.[3]

Architecting and designing is the next phase in the life cycle of an enterprise application which takes key inputs from the enterprise architecture initiatives of an organization. Enterprise architecture defines the overall business architecture, data architecture, applications architecture and technology architecture of an organization.[4] An enterprise application is one of the pieces to be fitted in the entire jigsaw puzzle of the applications landscape. Architecture of the enterprise application is laid out from various perspectives that include logical, integration, solution, data, technology and security, to name a few, key ingredients. These perspectives are further boiled down into the respective detailed design using various design patterns, frameworks, technologies and tools. The design includes several iterations to reach the final one, which is baselined to start the next phase—the construction phase.

Constructing an enterprise application starts with building the application framework components. This is followed by construction of application components, which implements the use cases identified during the requirements engineering of an enterprise application. Thereafter the software components are unit-tested. In addition, code review is an essential activity in the construction phase. Code review happens using both static and dynamic code analysis. Static code analysis includes type checking, style checking, security review, software quality checking, etc. Dynamic code analysis typically includes code coverage and code profiling.[5]

Testing an enterprise application is the most vital phase of the enterprise application life cycle that ensures the requisite ilities in enterprise applications for a successful application rollout. Various kinds of testing are done to ensure these and typically include integration testing, system testing and user acceptance testing. System testing facilitates to test various ilities of the applications. System testing typically includes performance testing, interface testing, globalization testing, compatibility testing, usability testing and penetration testing of the enterprise application.[6] Eventually, the enterprise application is rolled out in the production environment for end users and is maintained and supported till the end of life cycle of enterprise application.

---

[3] More about incepting an enterprise application is discussed in Chapter 2.

[4] More about architecting and designing an enterprise is discussed in Chapter 3.

[5] More about constructing an enterprise application is discussed in Chapter 4.

[6] More about testing an enterprise application is discussed in Chapter 5.

Every organization expects successful enterprise applications to be associated with certain qualities of service (QoS). The predominant ones are performance, security, maintainability, reliability and scalability. These qualities of service cut across the life cycle phases of enterprise applications.

Let us take the example of security. Security requirements are captured during incepting the enterprise application. Afterwards, threat modeling is done as part of architecting and designing phase of the enterprise application and, in turn, the outputs of threat modeling are used further in the same phase to have secure architecture and design. During construction of an enterprise application, secure coding principles and best practices are applied to have vulnerability free code and further it is tested from the security vulnerability perspective.

Performance is no different. Performance requirements are captured in the incepting phase. Sizing of infrastructure and performance test strategy is laid down during architecture and design phase. Although not necessary, preliminary load testing may be done in the construction phase. During testing phase, the enterprise application is subjected to load testing, stress testing, volume testing, endurance testing and other application assessments related to scalability and performance.

Throughout the life cycle of an enterprise application, it witnesses various stakeholders. The key stakeholders for incepting enterprise applications are the sponsors and the customers. Stakeholders from the process groups, product groups, service groups, IT application groups, IT infrastructure groups, analysts groups and vendors, etc. also play an important role during incepting enterprise applications. Architecting and designing phase is carried out by enterprise architects, data architects, integration architects, solution architects and application architects. Respective designers also join hands with the architects in their respective area of architecting and designing to accomplish this phase. Construction phase is primarily carried out by the programmers to build and review the code of application. Testing phase is accomplished by various kinds of testing teams such as integration testing team, performance testing team, application security testing team, interface testing team and user acceptance testing team. Finally, an enterprise application is rolled out with the help of release team and the representatives of the team that has raised the enterprise application.

1.4 Three Key Determinants of Successful Enterprise Applications

Business and IT alignment is the key for successful enterprises. It happens as a result of conceiving and raising successful enterprise applications that are in complete alignment with the business needs of the organization. There are broadly three aspects to raising successful enterprise applications:

(a) Business case readiness
(b) Strategy to execute
(c) Excellence in execution

The foundation of a successful enterprise application is laid down by an appropriate *business case* that is supported by the organizational objectives, vision and strategy. The business case should be supported from the financial perspective and have an acceptance of all the key stakeholders. An enterprise application should have well-defined parameters derived based on the business case to measure its success.

The next thing required to raise successful enterprise application is to have an *execution strategy*. The execution strategy is a comprehensive plan that is required to manifest budget, resources, timelines and availability of subject matter experts (SMEs) in a consistent manner to realize a successful enterprise application.

Once the execution strategy is in place, *excellence in execution* is required to realize a successful enterprise application. Enterprise applications typically warrant substantial time to get completed and during which business objectives, circumstances and environments may change. Continuous assessment of the objectives, having robust change management mechanisms and stake holder management is required to keep the target of raising enterprise applications intact. Robust traceability is also required to make sure the objectives are met. On-going communications to create stable baselines, sign off or formal hand shaking among stakeholders and reverse feedback mechanism in each life cycle phase of raising enterprise application are critical to the success of enterprise applications.

The stakeholders involved in the life cycle phases of raising enterprise applications should be equipped with the appropriate and required skill sets—both technical and soft—for their successful delivery. There are various ingredients that are required to raise successful enterprise applications, which you will go through in this book. Figure 1-2 represents these ingredients as an unstructured honeycomb. Although these ingredients are the predominant ones, other aspects such as organizational constraints, team dynamics, and soft skills come together to glue all these ingredients to consummate successful enterprise applications.

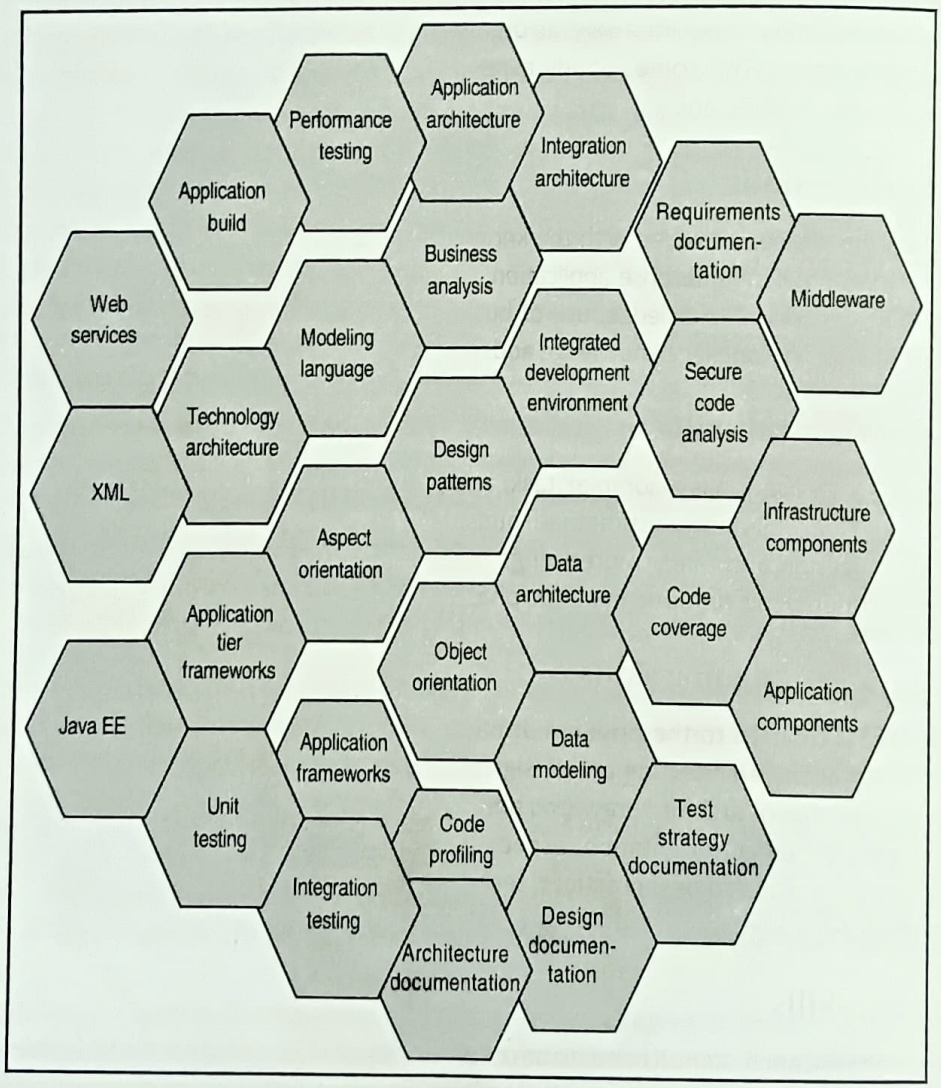


Figure 1-2 Ingredients of enterprise application.

Let us explore the typical knowledge and skill areas required by a team engaged in raising enterprise applications:

Knowledge of organizational dynamics

The most important thing required by the enterprise application team is to understand the organizational business and business needs of end users. This understanding provides several input to an enterprise application team, such as functionalities that should be packaged together, the kind of intuitive user interfaces required, components that can be reused, the positioning of a particular enterprise application in the overall application landscape and the interfaces required to be exposed or consumed by an enterprise application. This comes handy, especially during inception and architecting phase of enterprise applications.

Domain knowledge

The industry context and industry specific best practices are very important to build a competitive and cutting-edge enterprise application. It comes with the domain understanding. This provides input to the enterprise application team on the prescriptive business processes, information and data models etc. This provides valuable input to arrive at organization specific input on the processes, data and information management. This comes handy especially during inception, architecting and design and testing phase of such applications.

Business analysis skills

Skill of analyzing the business, coupled with the knowledge of domain and organizational dynamics, is very important for inception of an enterprise application. Business analysis skills are typically conglomeration of domain knowledge, technical knowledge, use of business analysis related tools and practice of soft skills. Business process modeling language knowledge adds to a robust documentation of business analysis results.

Program management skills

Raising enterprise application is a complex task, and it needs to be program-managed. Program management skills include planning, estimation, budgeting, talent management, change management, positive communication and many more things that are required to manage the program of raising enterprise applications. It is required in all the phases of the life cycle of an enterprise application.

Architecting and designing skills

Finding an optimal solution to the problem at hand requires architecting and designing skills. These skills are required in the architecting and designing phase of an enterprise application. This typically includes the knowledge of architecture views and view points, architectural patterns, design patterns, design paradigms like object orientation, aspect orientation and service orientation, usage of design tools, architectural and design best practices, technical frameworks, knowledge of modeling languages like Unified Modeling Language, etc.

Programming skills

To realize an enterprise application, it needs to be coded or programmed. Programming skills not only includes knowledge of a programming language but also the knowledge of the underlying platform, knowledge of an Integrated Development Environment (IDE) tool, programming best practices, code review skills, knowledge

of unit testing tools, configuration management and build tools, static code analysis tools and dynamic code analysis tools. Above all, "algorithmic" thinking is the key ingredient for a good programming.

Testing skills

To ensure an enterprise application is ready to use, it has to pass through multiple types of tests. Multiple types of testing skill sets are required for testing an enterprise application, which includes integration testing, performance testing, load testing, stress testing, application security testing, interface testing and user acceptance testing. Knowledge of various kinds of tools used to perform testing is also essential.

Knowledge of tools

Automation is the key to enterprise applications' quicker time-to-production and teams' productivity enhancement. This is achieved through usage of tools. Tools are used across all life cycle phases of enterprise applications. Requirements gathering and requirement analysis tools are required in the inception phase. Requirements management tools are required across the life cycle phases of enterprise applications. Architecting, design and modeling tools are required in architecting and design phase. IDEs, unit testing tools, build tools, code analysis tools are required in the construction phase and various testing tools like penetration testing, performance testing and regression testing tools are required during testing and rollout phase of enterprise application.

1.5 Measuring the Success of Enterprise Applications

For an organization, it's not only important to raise successful enterprise applications but also to measure the success of enterprise applications. Success is viewed from different perspectives by various stakeholders who have diverse roles and interests in raising enterprise applications.

Irrespective of the varieties of contexts and stakeholders objectives, typical parameters that are attributed to the success of enterprise applications are as follows:

- *Effectiveness of the solution* to the business problem at hand is the first and foremost parameter that determines the success of enterprise application. This parameter is important for all the stakeholders of enterprise application. One of the ways to measure the success is by measuring the productivity gain acquired by the end users of the enterprise application in performing their job function. Acceptance of enterprise application by end users is also a qualitative measure for this parameter.
- *Quality of enterprise application* is one of the key parameters to measure the success of enterprise applications and is important for all class and categories of stakeholders. Quality is attributed to the conformance to the functional requirements, defect-free code and adherence to the ilities by the enterprise applications. For example, to measure the success in terms of achieving performance and scalability, organizations use standardized benchmarks like online transaction processing benchmark (TPC-C). To measure security, organizations look for things like whether the Top-10 security vulnerabilities by the Open Web Application Security Project (OWASP) are being taken care or not while raising the enterprise applications. There are several other software metrics that are also used to quantify the software code quality. For example, cyclomatic complexity is used to measure the complexity of the source code, code coverage is used to measure the degree of source code tested, and defect ratio is to measure the density of defects.
- *Time-to-production* is a very important parameter from the perspective of the sponsors of an enterprise application in measuring the success of enterprise applications. In this fast moving and competitive world, business enterprises need to adapt to the new business environment

and roll out enterprise applications in the quickest possible time to be at the leading edge of the business. Time-to-production for an enterprise application is the key to success, which defines the total time taken to roll out the enterprise application since its inception.

- *Cost effectiveness* of enterprise application is another important parameter from the perspective of the sponsors of the enterprise application. It determines the expenditure versus the benefits of the enterprise application. Return on Investment (RoI), due to use of the enterprise application determines its cost effectiveness.
- *Budget and schedule adherence* in raising an enterprise application is one of the key parameter from the perspective of the enterprise application development teams. Overruns of budgets and exceeding the timelines have a direct impact on the time-to-production and cost effectiveness parameters. The enterprise application's program management team has the bottom-line responsibility for fulfilling this success parameter.
- *Productivity* is another important parameter to measure the success of enterprise applications from the perspective of enterprise application development teams. Reuse, use of best practices, frameworks and tools, and automation of software processes, etc. act as catalysts to improve productivity of the teams, which are responsible for raising enterprise applications.

Above all, one must remember that success of an enterprise application is purely a contextual thing, and it has to be vetted against the organizational vision, strategy and objectives behind raising enterprise applications.

Summary

This chapter has introduced the concept of enterprise applications, types of enterprise applications and typical challenges posed in raising today's enterprise applications. The iterative software engineering methodologies are introduced from the perspective of key characteristics and applicability. This chapter has also introduced the typical life cycle phases of raising enterprise applications, and the stakeholders who interface and participate in these life cycle phases. The three key determinants to raise successful enterprise applications are described, along with key knowledge areas and skill sets that are required from an application's stakeholders. Towards the end of the chapter, the key parameters, determining the success of enterprise applications are also elucidated.

Review Questions

1. Identify the enterprise applications in your domain area and categorize them.
2. List three most important challenges in raising an enterprise application.
3. Perform a SWOT analysis of the two software engineering methodologies.
4. Find out the names of life cycle phases in the Scrum approach.
5. Explore the 4 + 1 view of IBM Rational Unified Process.

Further Readings

Application security: http://www.owasp.org/index.php/OWASP_Top_Ten_Project
IBM patterns for e-business: https://www.ibm.com/developerworks/patterns/
IBM Rational Unified Process: https://www-01.ibm.com/software/awdtools/rup/
Performance benchmarks: http://www.tpc.org/
Scrum methodology: http://scrummethodology.com/
XP methodology: http://www.extremeprogramming.org/

Incepting Enterprise Applications

2

Learning Goals

After completing this chapter, you will be able to:

- Define enterprise analysis and business modeling.
- Apply requirements elicitation and analysis.
- Determine actors, use cases, relationships among them and use case specification.
- Explain prototypes.
- Identify various types of non-functional requirements.
- Create system requirement specification.
- Define requirements validation and requirements traceability.
- Describe project plan.
- Determine different project estimation techniques.
- List tools used for business modeling and requirements engineerings.

In the last 10 years, we have seen companies like Amazon, eBay and Google having tremendous growth. Also, we have seen other "brick-and-mortar" companies, such as Walmart and Tesco, adapting multichannel commerce to service customers and be competitive in the current environment. Even service industries like banks and telecom use technology as their backbone to serve end-customers' need with a very short time-to-market. In all these companies, we are talking about real complex business processes and systems.

To run such complex and high volume businesses, enterprises require raising complex software systems, which cater to the need of customers and other stakeholders. Incepting an enterprise application is the first step to raise it. This is required to determine the problem at hand, identify the need for the enterprise application, establish the scope, analyze the feasibility, and cast a roadmap for further phases of lifecycle of enterprise application development.

Incepting an enterprise application primarily consists of the following activities:

- Enterprise analysis
- Business modeling
- Requirements elicitation and analysis
- Requirements validation
- Planning and estimation

Let us further explore these activities in the following sections.

2.1 Enterprise Analysis

Business continuity and sustained growth are of paramount importance to any enterprise. Enterprise analysis deals with questions such as "Why is a given system needed?", "Why does a given system need change?", "Why is it important to invest at this time?" As part of enterprise analysis, a business analyst performs the following activities:

- Identifies the business opportunities and business changes
- Identifies stakeholders across business units
- Collects and prioritizes the business requirements
- Defines the business roadmap with scope and exclusions
- Determines the high-level investment needed for the enterprise
- Conducts feasibility study for any changes proposed
- Conducts risk analysis and competitive analysis
- Decides on build-or-buy strategy
- Creates "Return on Investment" (RoI) business cases with proper justifications
- Gets the necessary approvals from the sponsors

Xp

Not all enterprise applications initiatives are preceded by the enterprise analysis activity. This is typically an activity that is undertaken as part of enterprise architecture exercise and provides a roadmap for the complete set of enterprsie applications. This activity leads to fulfillment of the business goals of an enterprise,.

As part of enterprise analysis, or just after it, most organizations do a complete business modeling exercise.

2.2 Business Modeling

Enterprise analysis could lead to two forms of programs/projects:

(a) Creating something new
(b) Extension/change to something which already exists

In normal engineering terms, the first type is commonly referred to as *development program/project* and the second type as *reengineering program/project*. In a development project, the entire business requirements are arrived at from scratch. For example, to create an e-commerce platform for an existing brick-and-mortar business, one has to define the entire business process and transactions afresh.

An example of a reengineering project could be rewriting the existing brick-and-mortar business from a legacy system written in COBOL to a new modern system based on Java. In this approach, we may get the entire business process and the validations from the existing system and may have only a few new requirements. While there is no standard definition of what a reengineering project is, it is most widely accepted to mean that there are no major changes to requirements, and hence does not involve a "requirements" or "inception" phase. Requirements are derived from the existing system and implemented with little or no change.

To raise a successful enterprise application, it is important to understand the business and business imperatives in detail. These business details and processes have to be well documented and a business user should be able to verify/validate it easily. Business modeling helps one understand the business information and the business processes, which an enterprise uses to fulfill its business goals. Business modeling is done by business analysts.

To understand the business problem at hand it's essential to have an *as-is* and *to-be* modeling of the business processes. The "as-is" business process model reflects the existing business process, and

the "to-be" business process model reflects the desired business process. "As-is" and "to-be" business modelings catalyze the understanding of the deficiencies of the existing process and provide insights about how they could be overcome.

Tt Tools for business modeling can be as simple as using a whiteboard or can be of varying degrees of sophistication from a simple diagramming tools like Microsoft® Visio® to a full-blown modeling notation-aware tool such as Enterprise Modeler®.

Let us further understand more about business modeling using a case study in the loan-banking domain. The same case study will be followed throughout this book so that readers can understand and appreciate how an enterprise application is raised.

2.3 Loan Banking Business: Case Study of EM Bank

EM Bank (Easy Money Bank) is a leading global bank, headquartered in Europe, with presence in more than 50 countries across continents. EM Bank offers services in core banking, investment banking, mortgage, foreign exchange and wealth management.

The core banking segment of EM Bank has been offering loans to its retail customers through its branches, since 1970. In order to keep up with the competition and current business trends, the bank management decided to expand the loan offerings to a wider customer base using multiple channels to reach the prospective and existing customers.

As part of the business expansion exercise, EM Bank's business analysts analyzed the "as-is" business process of the loan management services. Figure 2-1 depicts the "as-is" business process for loan management.

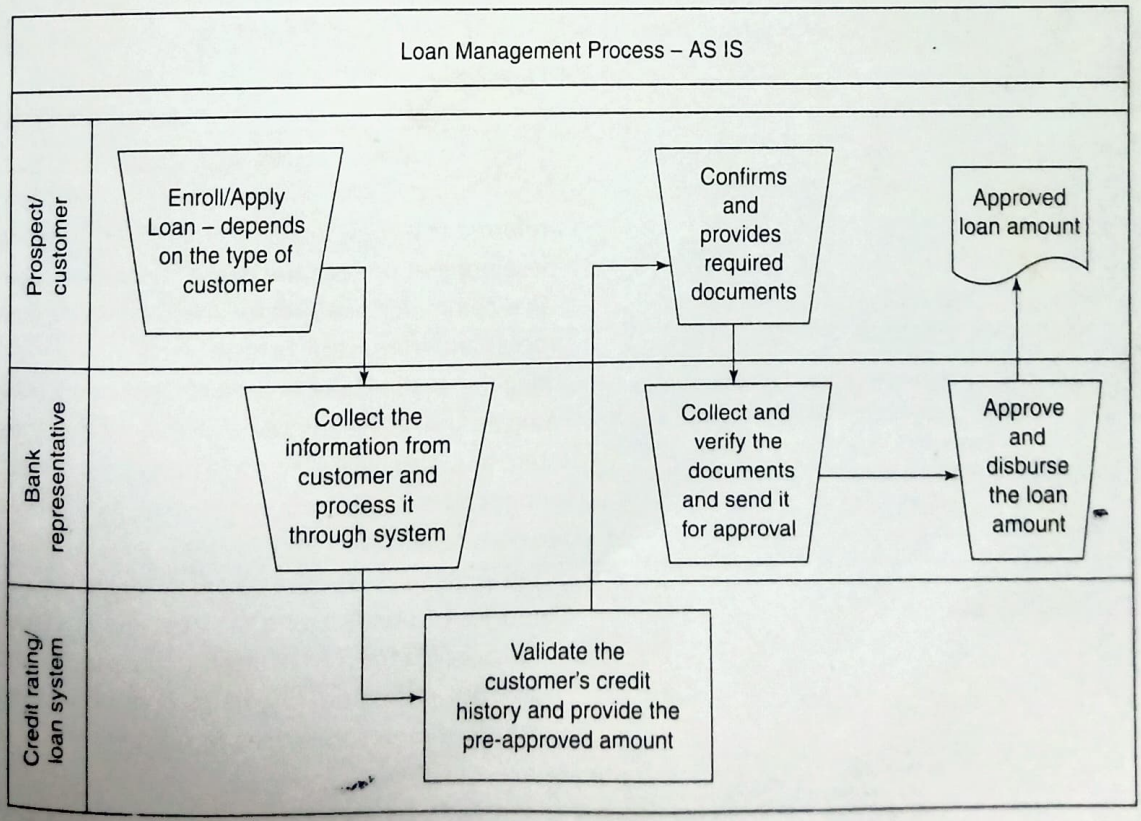


Figure 2-1 Loan management—"as-is" business process.

As depicted in Figure 2-1, EM Bank has a single channel (only through bank branch) to offer loans. The preapproved loan amount is decided based on customer relationship with the bank. All the verification and approval processes are done by the bank representatives. There is no external agency interface to provide customer credit rating.

After analyzing the "as-is" business process, the following "to-be" process (as illustrated in Figure 2-2) is identified in order to meet the objective of expanding loan offerings:

- Automate loan offerings for existing customers as well as new customers (prospects) via bank branches and the Internet.
- For the prospect to do business with the bank and get a loan, a prospect has to register (enroll), either through the Internet, or by meeting with one of the bank's representatives. On successful completion of the registration process, the prospect will be treated as an existing customer.
- During customer loan initiation process, the customer fills in the particulars either through the Internet or by contacting a bank representative. Information about the preapproved loan amount will be made available to the customer at the time of loan initiation. The customer can either choose to go with the preapproved loan amount (or an amount less than it), or opt for a higher amount by placing a request for enhancement of the loan amount.

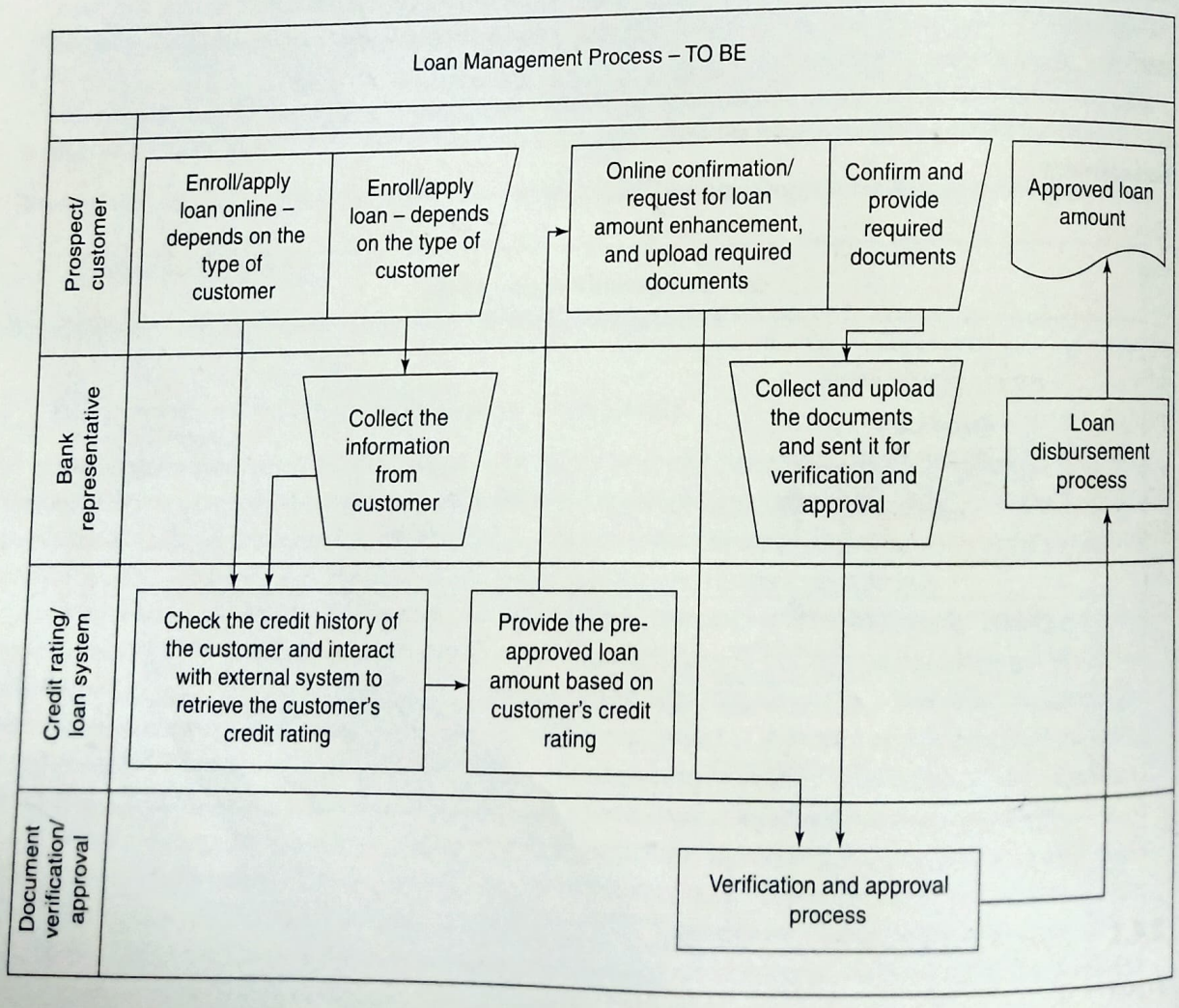


Figure 2-2 Loan management—"to-be" business process.

- The preapproved loan amount is based on the customer's credit rating. Customer credit rating facility is offered to EM Bank by an external agency.
- Upon loan initiation, the customer submits the necessary documents (such as employment proof, address proof and social security number) by either uploading them on the system, or by providing them physically to a bank representative. Physical documents are digitized, uploaded and linked to the loan application by a bank representative.
- Once the documents are uploaded onto the system, a verifier verifies the application and the related documents.
- If all the given documents are appropriate, and verified by the verifier, the loan request is sent to the approver for approval. Once the loan is approved, the customer is notified of the approval and amount. The approved loan amount is then disbursed to the customer.

To accomplish the identified "to-be" business process, a new loan management system (LoMS) needs to be developed. LoMS system will have functionalities from loan initiation to disbursement of the loan. This system will be used by both the bank customers and the bank staff. In all the subsequent chapters, LoMS application will be taken as an example to understand and appreciate the life cycle of raising enterprise applications. In practice, an enterprise application is much more complex and big, but the example application (LoMS) is sufficient to understand the key principles, paradigms and practices to raise an enterprise application.

After business modeling, the next step in incepting the enterprise applications would be to detail out the requirements. These detailed requirements would then be analyzed for completeness and validated for feasibility and correctness.

2.4 Requirements Elicitation and Analysis

Once the operation of the business is specified in business process models, it can be used as an input to perform more detailed requirements elicitation and analysis for enterprise applications.

Requirements elicitation and analysis is a systematic approach of capturing client requirements, analyzing them and documenting the problem domain. There are various ways to approach it. Review of existing systems and relevant documentation, interviewing IT personnel of the organization, and prototyping are few of them. Business analysts facilitate the requirements elicitation and analysis.

Requirements can be of two types: good requirements and bad requirements. Bad requirements are introduced primarily by making assumptions while writing requirements, or not restricting focus to the problem domain through mixing up aspects of the solution domain. Needless to say, one should always avoid bad requirements. Remember, good requirements are always SMART (that is, specific, measurable, attainable, realizable and testable/traceable/time bound). There are various kinds of requirements that need to be elicited and are broadly divided into two categories:

- *Functional requirements*. Functional requirements capture what the system is expected to do. To depict functional requirements, "use cases" and prototypes are used.
- *Nonfunctional requirements* (*NFRs*). NFRs capture how the system does what it is expected to do with respect to its constraints and expected qualities of service (QoS) such as reliability, scalability, portability, usability, availability, security and performance.

2.4.1 Actors and Use Cases

Functional requirements elicitation and analysis starts with enlisting the entities that use the system, and the functional reasons why they will use the system. The entities are termed as *actors* and the

functional reasons are termed as *use cases*. However, if one is using Extreme Programming methodology, requirements are stated as "user stories", rather than use cases.

In

"A picture is worth a thousand words." This old proverb defines the soul of Unified Modeling Language (UML) that is the de-facto visual language of modeling business processes and software systems. UML2.0 is the latest specification by Object Management Group (OMG) that supports 13 modeling diagrams. Use case diagrams are one of these diagrams to represent the functional *requirements view*.

The entities or actors, as termed in UML[1] jargon, shouldn't be assumed to consist of only human users. Apart from human users, actors can be external systems or external devices. To find out the actors, one should ask a simple question "Who is going to use the system?" Customer, reviewer and approver are the human user actors of LoMS. Accounting system and credit rating system are the system actors.

While identifying actors, following few points may be of help:

- An actor typically represents a role, and not a user.
- For bigger applications, typically a "system admin" actor is required. But it purely depends on the requirements.
- Only direct actors of a system should be considered.

A use case is initiated by an actor for a particular functional reason. To determine use cases, one should ask a simple question "Why does an actor need to interact with the system under consideration?" For example, in LoMS, why does a customer want to interact with the system? Customer wants to initiate the loan. Why does the credit rating system interact with the system? A credit rating system provides the credit rating of the customer. Therefore, "initiate loan" and "provide credit rating" are two different use cases. The complete list of use cases will provide the intended behavior, and hence the scope of the system.

After identification of actors and use cases, the next step is to identify the relationships among them. The simplest kind of relationship that exists among use cases and actors is the *association relationship*, which is depicted with a straight line between actor and use case, as shown in Figure 2-3.

Once the actors, the use cases and the relationships among them are identified, the next step is to create the use case diagrams. Use case diagrams are one of the tools to capture the functional requirements. The first level of use case diagram, which represents all the actors, high-level use cases, the system boundary and the relationship among the actors and use cases, is represented in Figure 2-3. The level of use cases diagrams, one should delve into, is purely dependent on the project complexity as well as the use case complexity. The thumb rule is to stop when the implementation of a given use case can be estimated.

Tt

There are several tools available to create use case diagrams. Few popular ones are Rational Software Architect, Visio and Together Architect. Other open source choices like AgroUML are also available

[1] For more information on UML, please refer to the Object Management Group's official Web site for UML, http://www.omg.org/gettingstarted/what_is_uml.htm

As shown in Figure 2-3, customer and bank representative are two human actors who participate in the "initiate loan" use case. Credit rating system and accounting system are two external system actors that participate in the same use case. Thus "initiate loan" effectively has associations with four actors.

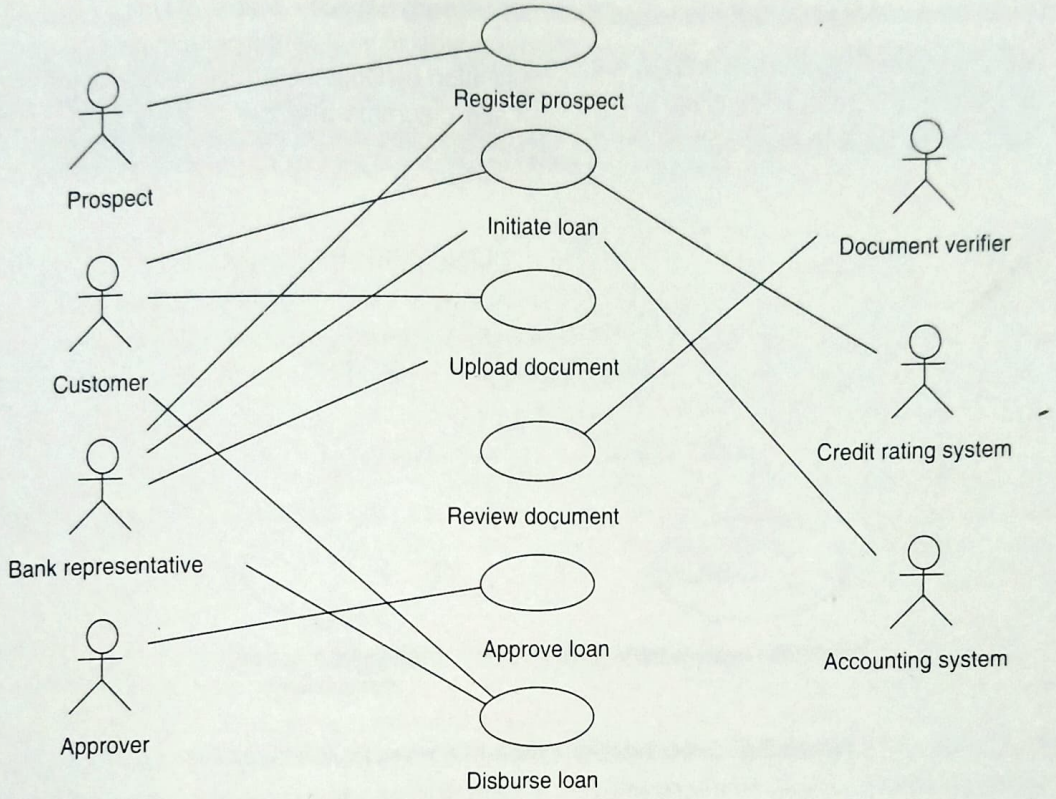


Figure 2-3 LoMS use case diagram.

Let us explore a few more relationship types that exist among actors and within use cases. As depicted in Figure 2-4, the bank representative, approver and the document verifier actors are the types of bank executive actor. This means that bank executive actor generalizes the bank representative, approver and the document verifier actor. This type of relationship is referred to as *generalization relationship* and is the only type of relationship that exists among actors.

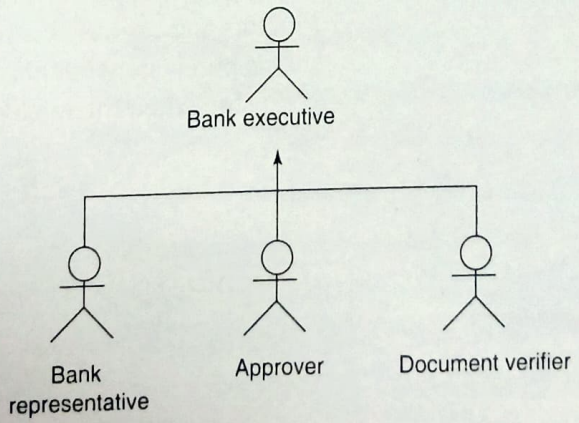


Figure 2-4 Generalization relationship among actors.

In UML2.0 does not allow associations among actors. But, the generalization relationship is useful in depicting common characteristics among actors.

There can be three types of relationships that can exist among use cases.

(a) *Generalization relationship* among use cases exists, if a particular use case is a specialized form of another use case, as depicted in Figure 2-5. For example, Initiate Loan by Customer and Initiate Loan by Bank Representative are the specialized use cases of Initiate Loan use case.

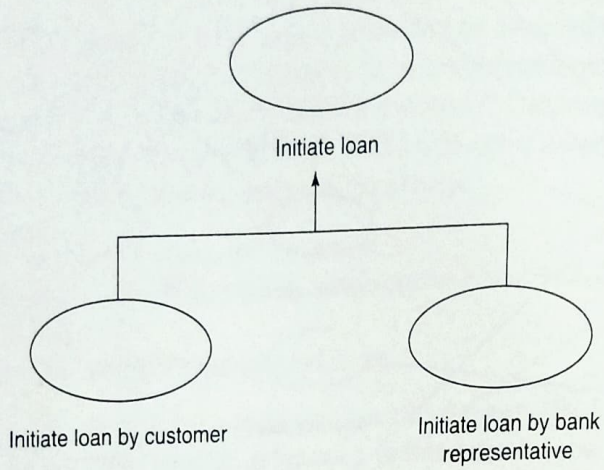


Figure 2-5 Generalization relationship among use cases.

(b) *Extends relationship* among use cases exists, if behavior of one of the use cases (extension use case) may be conceptually inserted in the base use case (extended use case) under some condition(s), as depicted in Figure 2-6. Provide Credit Rating is an extension use case of Initiate Loan, because as per loan initiation requirement, preapproved loan amount will be fetched along with the credit rating from the credit rating system in case the preapproved amount is not updated in the last six months. It's represented as a dashed arrow pointing from extension use case towards the extended use case.

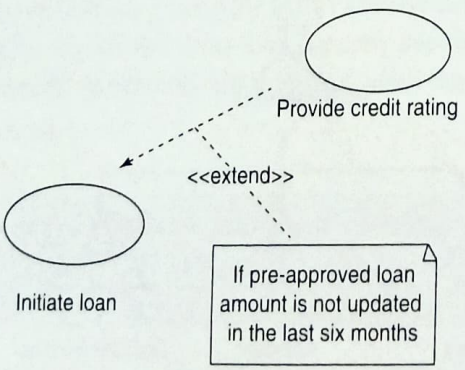


Figure 2-6 Extends relationship.

(c) *Uses relationship* among use cases exists, if one of the use cases (including use case) always includes another use case (included use case), as depicted in Figure 2-7. It is also known as *include relationship*. The Provide Customer Information use case is always used by the Initiate Loan use case, because as per loan initiation requirement the customer information is always processed. The Initiate Loan use case is the including use case, and the Provide Customer Information use case is the included one. It is represented as a dashed arrow pointing from the including to the included use case. It makes sense to extract a use case as an "included" use case only if that use case is used by more than one use case.

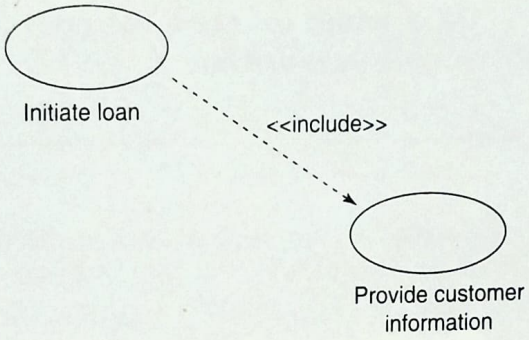


Figure 2-7 Uses/include relationship.

A use case diagram does not provide a comprehensive description of use case functionality. Typically, a textual description of the use case functionality is also documented, and is commonly referred to as *use case specification* (UCS) document. It primarily consists of actors, preconditions, post-conditions, primary flow and alternate flows of a use case. Table 2-1 provides a template for the details that are generally captured as part of a UCS document.

Table 2-1 Elements of use case specification

UCS elements	Description
Actors	List of participating actors in the use case
Description	A brief description of the use case
Preconditions	List of prerequisites to start the use case
Post-conditions	List of things which are outcome of the use case
Priority	Business criticality is quantified and presented in a relative manner
Trigger	The event that initiates the use case
Primary scenario	The primary flow of the use case. Primary scenario is the most frequently occurring scenario in the entire use case
Alternate scenario # number	List of alternate flows. These typically occur less frequently than the primary flow

(*Continued*)

Table 2-1 (*Continued*)

Field definitions	The "significant" nouns are collected from various scenarios and are jotted down along with the specifications such as size of the field, type of the field, possible values and default values. These act as an input to design the data models in the successive phase
Exceptions	List of exceptional flows
Assumptions	Assumptions, if any
User Interface	Reference to the prototype(s)/wireframe(s)
Related use cases	List of related use cases. Use cases that are included or having extension are listed here
Nonfunctional requirements	Capture NFRs specific to this particular use case

Tt UCS documents are prepared typically using a standard document editor like Microsoft Office Word®. UML tools like Rational Rose, Together® Architect are used to attach this document with the respective use cases modeled with these tools.

Primary or alternate scenarios can be sometimes quite complex and be supported with even better and lucid explanations in the form of activity diagrams. Activity diagrams are UML diagrams that are similar to flow charts, and are good at visually representing the flow of activity of a use case. Designing activity diagrams is purely optional and typically exercised only for those scenarios/use cases that are really complex to comprehend in a textual representation.

2.4.2 User Prototypes

The human user actors identified in the previous steps need to interact with the system. This boils down to creation of sophisticated User Interfaces (UI), but typically starts with creating prototypes or *wireframes* of these user interfaces. Wireframes are the bare minimum visual elements laid down to depict the basic user interface design.

User prototypes or wireframes serve the following purposes:

- They can be used to allow business users to validate the user interface, which helps in managing expectations and getting "buy-in" from them to avoid last minute surprises.
- UI designers can attain a comfort level from the look-and-feel perspective of wireframes. It also serves as a blueprint that helps in managing the UI design consistency.
- Business analysts and UI designers are able to find out usability requirements like navigability, which is one of the key nonfunctional requirements.
- Business analysts get a better aid in understanding the exceptions and variations in use cases.

Tt Dreamweaver is one of the favorite tools of wireframe designers. Microsoft Visio® or Adobe® Illustrator can also be used to create wireframes,

The following are some of the points that need to be taken care while designing the user prototypes:

- Tools like Microsoft PowerPoint® can be used for prototyping, but it's a good practice to use HTML as the basis for prototyping.
- Sample data used in prototype should be indicative of real data.
- Prototypes should concentrate on the primary scenario of a use case. If alternate scenarios are critical, then only it is advisable to consider them for prototyping.
- A few tools, like Dreamweaver, have the capability of creating wireframes that can be further converted into a skeletal code.

In the era of Web 2.0, where collaboration is the key, annotated prototypes are also used. A team developing prototype can easily share and add notes to it. One such prototyping collaboration tool is Protonotes.[2] Tools such as Wiki and shared portal are also used for requirements gathering in a collaborative mode.

2.4.3 Nonfunctional Requirements

In contrast to the functional behavior of an enterprise application, there are characteristics and constraints associated with the application, termed as *nonfunctional requirements* (or NFRs). These characteristics and constraints are of paramount importance for any enterprise application.

Please remember, the NFRs are not accidental; they must be captured, documented and considered very carefully during development of the system.

Referring to the sample system LoMS, it should be able to process 1000 loan applications in a day. It should be able to support 20 percent of average growth in the volume of loan applications every year. It should be supported by Internet Explorer version 6 and 7. Context-sensitive help of the system should be available. It should have a maintainable code base. It should be able to avert faults due to the underlying applications servers and database servers. It should not be prone to injection attacks. These are a few of the examples depicting the characteristics and constraints expected of LoMS, in particular, and of any enterprise application, in general.

The NFRs of an enterprise application have a very high impact on the way the system has to be architected, designed and deployed. In fact, NFRs play a pivotal role in the validation of the architecture and design of enterprise applications.

A customary question is "What is to be captured as NFRs?". There are myriad of them, but the typical ones that need to be captured in an NFR document are as depicted in Table 2-2.

[2] You can refer to Protonotes at http://www.protonotes.com/

Table 2-2 Key NFRs

Key NFRs	Description
Performance requirements	Constraints like throughput, response time and refresh time under normal and peak loads
Usability requirements	Effective use of screen real estate, navigability, localization and internationalization, browser support and accessibility requirements
Scalability requirements	Resource utilization, data storage requirements and projected growth metrics could be used for better capacity planning and management.
Interface requirements	Dictate the mechanisms followed for application's interoperability or integration with existing enterprise applications
Operating requirements	Security, maintainability and reliability requirements have to be laid down as part of operating requirements. Recovery time objective (RTO), recovery point objective (RPO) and mean time between failures (MTBF) are few of the metrics captured for reliability operating requirements
Lifecycle requirements	Testability, reusability, portability and installability requirements
Regulatory requirements	Legal and licensing requirements

Let us now discuss these nonfunctional requirements.

Performance requirements

Performance requirements are the performance constraints that need to be identified for each critical use case of the system under a given set of conditions. There are three key determinants of the performance of any enterprise application:

(a) Workload of a system
(b) The server software, hosting the enterprise application
(c) Underlying hardware of the enterprise application

The number of user requests to the system, think time of the user to perform an action in the system and arrival rate of requests are the key performance factors in the specification of the workload. This eventually boils down to the system throughput. *Throughput* is defined as the amount of work that a system can perform in a given time period. The server software typically involves the operating systems, application servers and database servers. The underlying hardware of the application typically involves processors, disks, memory and networks. An application uses resource utilization as one of the key performance metrics.

Performance requirements of an enterprise application can be categorized into two perspectives:

(a) *Business perspective* captures the requirements like types of transactions, types of users, their usage pattern and request arrival pattern. These are required to measure the workload and throughput of the system.

(b) *Technology perspective* captures the requirements like operating environment details, system and interface details and performance boundaries with respect to the external systems. These are required to measure the resource utilization and the response time of the system.

In case of LoMS, the performance requirements state that the system should be able to process a peak of 60 and an average of 40 concurrent transactions per second. It should be able to support 250 concurrent user sessions and process 1000 loan applications in a day. The maximum page-refresh time involving online transactions should not be more than 5 seconds.

Xp

User interaction transactions, batch transactions and report generation transactions are three different types of transactions. The metrics for throughput and response time for different types of transactions should be captured separately.

The nonfunctional requirements gathered from performance perspective of enterprise applications are used to perform various performance related tests like load test, volume test, stress test and endurance test on applications. Procedures followed to test and tune the performance and how enterprise applications conform to the performance requirements are elaborated in the later chapters.

Usability requirements

Usability requirements dictate the user experience. This is one of the key NFRs that can make or mar an end user's perception of the application. These are one of the most important requirements from the perspective of acceptance of an application by end users.

In case of LoMS, the usability requirements state that the system requires customers and prospects to be self guided through the loan initiation process. For existing customers, the loan information should be pre-populated, if available. For all of the human actors, the system should facilitate easy navigation combined with easy workflow. The LoMS application should support internationalization as it needs to be localized as per the user base. Context-sensitive help should be available to the user.

The following simple thumb rules can really help in smooth acceptance of enterprise application by the end user:

- Proper usage of the screen real estate is very important. Logical grouping of the items on the user screen or placing the relevant contents at the appropriate place helps the end user to understand the overall screen in a better manner and in turn decreases the think time of the user to take an action on the screen.
- Easy to use and intuitive navigability is the key to usability of an application. Having excessive number of dialog boxes is not a good practice. Too many switches between mouse and keyboard should be avoided. The tab sequences should be defined appropriately, in case application supports user inputs through keyboard.
- Wherever possible, provide auto populate and auto complete of data in forms' fields to save the time of user. This results in less number of data input mistakes.
- Interacting with an application in one's preferred language provides more "easy-to-use" and "personalized" touch to the end user. Careful internationalization of the enterprise application is very important so that it can be localized as per the end users' chosen locale and language requirements.
- Support for popular browsers is also necessary.

In

End users with disabilities require accessibility support in applications. Accessibility requirements help such users access the application easily and effectively. W3C® has launched Web Accessibility Initiative (WAI) to strategize the guidelines for accessibility.[3]

Scalability requirements

Scalability requirements represent the ability of the system to sustain its performance with growing number of users and data of the application, and require careful and elaborate capacity planning and management.

In a typical enterprise application, the number of users will keep increasing over time. Corresponding to the user population growth, the data volume also grows. This might have an impact on system throughput and performance.

In case of LoMS, the scalability requirements state that the system should be scalable to support an average annual growth of 20 percent in the volume of loan applications. To gauge the data storage requirements in LoMS, the size of data corresponding to a loan application needs to be identified. If on an average, the size of a loan application data is 1 MB and 1000 loan applications need to be processed in a day, the per day data storage requirement is 1 GB. Additional data storage requirements for attachments (verification documents in LoMS) also need to be captured to arrive at the final requirements. The data storage requirements, along with the projected growth factor, help in identifying the optimum database size of an enterprise application.

To achieve the required scalability in an application, either resources are topped up in the existing system, or separate physical systems are added to the existing ones. The former approach is referred to as *vertical scaling* and the latter one as *horizontal scaling*.

Interface requirements

Interface requirements pronounce the interoperability and integration requirements of the enterprise applications. Enterprise applications shouldn't exist in silos. They have to coexist in the enterprise ecosystem. They need to provide services or consume services for automating the business processes. To achieve this, they need to be interoperable and integrated within the overall applications' landscape.

Enterprises follow various mechanisms for integration such as:

- Integration at application's User Interface level
- Integration at business data level
- Integration at business process level

Xp

The "Application bridging" approach is followed when high performance is chosen over the integration flexibility. This leads to the tight coupling among applications.

The following simple questions can help in gathering the interface related requirements:

- What are the trigger events for communication to happen between applications?

[3] For more information about WAI, please refer to http://www.w3.org/WAI/

- What are the methods to facilitate communication between applications? Whether there is a constraint of using a specific communication protocol or a specific device or format?
- What is the structure in which data has to be passed between applications?
- What are the mechanisms to handle errors during communication?
- Whether the interface is crossing the technology domain?

In case of LoMS, the interface requirements state that the system needs to interact with the credit rating system that exists outside the organization boundaries and the accounting system that exists within the organization enterprise applications landscape. You will learn more on integration mechanisms and technologies in the next chapter.

Operating requirements

Operating requirements are the constraints on the enterprise application that lay down the requirements related to security, maintainability and reliability of enterprise applications.

Security requirements are typically categorized into following two types:

- *Network security requirements*: The network security requirements may include requirements related to the network firewalls, routers and intrusion detection systems, etc. These help in designing the application's deployment infrastructure.
- *Application security requirements*: The application security requirements cater to the implementation of security's AAA. Security's AAA stands for Authentication, Authorization and Auditing. In addition to AAA requirements, the enterprise applications, like LoMS that are Internet facing, need to be safeguarded against the Web security vulnerabilities like Cross Site scripting, SQL injection, etc. User session related requirements like configurable session time out, denying the repeated unsuccessful attempts to create a session by a source and rejection of concurrent login by the same user, etc., may also be captured.

Maintainability of an application is related to its manageability and analyzability. The required monitoring facilities and application's configuration capability is documented as manageability requirements. Requirements like need of log files, audit trails, monitoring support, etc., are captured as analyzability requirements.

Reliability requirements of enterprise applications include the system's availability and recoverability requirements. LoMS availability requirement of 99.999 percent leads to the need for fault tolerance and load balancing mechanisms for the application. Recovery time objective (RTO), recovery point objective (RPO) and MTBF should be captured to illustrate the recoverability requirements of the system. RTO signifies the amount of time taken by a system to recover from its previous known state and be available for use. RTO requirement for LoMS is within 10 minutes. RPO requirement is that in case of failure, the system shall recover no less than 30 minutes old transaction data before the failure and within 10 minutes of failure. MTBF is defined as an average time between failures of the system. LoMS shall support MTBF of more than three months.

Life cycle requirements

Life cycle requirements relate to needs that an application should fulfill at one or more stages of its life cycle, and can be captured in terms of testability, reusability, portability and installability to name a few.

- *Testability requirements* capture the quality of an application, whereby it lends itself to being tested in a comprehensive manner without requiring too much effort.

- *Reusability requirements* capture the aspects of a system that should be designed for reuse, and those that should be developed with reusable components.
- *Portability requirements* capture the hardware and operating system platforms on which the system should be capable of being deployed.
- *Installability requirements* capture the requirements like mode of installation-automated or manual, requirement of installation guide and scripts. These requirements help in planning the rollout of an application.

Regulatory requirements

Regulatory requirements capture the restrictions and the legal requirements related to certain categories of sensitive data, and ways in which they can be processed. For example, some countries have regulations that require sensitive data to be maintained within the country. Some have regulations that require a specific data to be archived for a specified duration.

In Requirement elicitation and analysis phase involves finding and capturing the functional and nonfunctional requirements of a software system. The elicited requirements are documented as a system specification or functional specification document. A software requirement specification (SRS) document is derived from the system specification document by considering how the functional requirements will be met by the system that is under development.

SRS document is the master document for all the further artifacts that are generated during the life cycle of raising enterprise applications. SRS serves as a starting point for the enterprise application development team. It establishes assurance between the client and the team designated for raising the enterprise application. A template for SRS is described by IEEE Standard 830-1998.

Table 2-3 depicts the typical elements captured in an SRS document.

Table 2-3 Elements of software requirement specification

SRS elements	Description
Business overview	Brief overview of what an enterprise does
System overview	Overview of the system under consideration
Functionality list	The list of business functionality in an unambiguous form
Use cases	The list of identified use cases and attached respective use case specification document
Prototypes	Wireframes of the primary functionalities
NFR list	List of nonfunctional requirements

2.5 Requirements Validation

Requirements validation is an exercise that is typically facilitated by business analysts to ensure that the requirements stated during requirement elicitation and analysis are meeting the business objectives. This is really an important exercise, as it helps in ensuring the adoption and usage of the enterprise application by the business users.

Although requirements analysis and requirements validation activities go hand in hand, the entire activity of requirements validation typically comprise of the following three activities:

(a) Ensuring the coverage of all business needs identified during enterprise analysis and requirements elicitation phase
(b) Ensuring the requirements documentation sanctity by subject matter experts (SMEs) and end users
(c) Ensuring the feasibility of requirements to the extent possible

The requirements that are elicited during requirements elicitation need to be validated from the viewpoint of completeness so that target solution addresses all the business needs. It is typically ascertained by establishing a *requirements traceability matrix* (RTM). A sample (partial) RTM for LoMS has been depicted in Table 2-4.

Table 2-4 Requirements traceability matrix

Use case	Functional requirement	Design element	Code	Test case
UC01-Initiate Loan	Req1.1- Existing customers can initiate the loan	Loan initiation design	Class LoanInitiation	LoanInitiation Test#1 LoanInitiation Test#2 LoanInitiation Test#3
UC02-	–		–	–

Tt RTM can be set up using plain and simple spreadsheets or by using specialized software like Rational Requisite Pro for capturing and managing requirements.

RTM helps not only in tracking requirements from a coverage point of view but also in managing the requirements throughout the software development life cycle (SDLC) by mapping requirements to the further phases of SDLC like design, development and deployment. RTM in table 2-4 depicts the 'Initiate Loan' use case, and how it is mapped to other deliverables/components of the software delivered in the further stages of SDLC. (We will explore these deliverables/components in further chapters.) RTM is tedious to maintain, but it actually helps in precisely locating the related software elements, and it can really save a lot of time during impact analysis and maintenance activities of the enterprise application.

In A requirement that is inconsistent, incorrect, ambiguous, unacceptable and non-testable is called as an *invalid requirement.*

Another important way of validating the requirements is to ensure the proper documentation of requirement specifications and its thorough review. Requirement reviews are typically done by SMEs. A specific requirement, such as User Interface (UI) requirement, is usually validated by end users with the help of UI prototypes or wireframes.

Another important technique to validate requirements is to prepare *user acceptance test* (UAT) cases. The review of these cases helps end users validate the requirements.[4]

[4] More about UAT will be discussed in Chapter5.

Requirements validation also involves ensuring the feasibility of requirements. Feasibility of requirements is more focused towards analyzing the feasibility from business angle and at times from the technical angle. This is to ensure whether the target solution is compliant with the organizational technical standards and guidelines or the target solution can coexist within the existing enterprise landscape.

Proof of Concepts (PoC) is also used to validate the feasibility of a requirement. PoC developed during incepting an enterprise application helps in articulating functional specifications and converting these specifications to software requirement specifications. It is a minimal solution to validate requirements from several perspectives such as how business process flows are implemented in the system. Although PoC development is a time-consuming and effort-intensive activity, it is worthwhile to do a PoC to validate the feasibility of requirements elicited, especially in case of complex situations.

2.6 Planning and Estimation

As discussed earlier, the enterprise analysis phase leads to the development of a business case. The business case acts as the primary input for preparing the highest level planning document, called *project charter*. Planning is primarily done by a project manager, and business analysts and architects play an important role in providing the necessary information for project planning.

Once the project charter is laid down, bringing clarity to the project scope, detailed project planning is done. Detailed planning involves multiple stakeholders taking decisions with project manager facilitating the exercise. Answers to questions like "Who is going to involve in the project?," "When will the project start?," "What are the dependencies?," "What are the efforts involved?," etc., are required for planning. Identifying the cost, schedule, size and effort involved upfront is called *estimation*.

Project estimation strengthens the planning, and planning needs estimation as one of its key components. Project estimation is usually done by project managers along with the help of business analysts and technical architects. Estimation is done at various stages for planning the future phases of SDLC. Earlier the life cycle phase, less the information available, and less the accuracy of estimation!

How to estimate the enterprise application efforts and timelines during incepting stage? At this point in time, documented requirements in terms of use cases along with detailed use case specifications and prototypes/wireframes are available for estimation.

What exactly are we estimating when we say "estimation"? We are estimating the size of the project and the efforts required for the project execution. Size of the software project is estimated in lines of code (LOC) or function points (FP). A project effort is typically measured in person-months. They are related to each other by a "productivity factor," which could vary based on team and organization capabilities.

There are several estimation techniques that are followed for estimating the schedule, time, effort, etc. A few of the popular estimation techniques are:

- Ballpark estimation
- Use Case Point estimation
- Function Point estimation

Ballpark estimation is typically "order of scale" estimation. This is usually applied in the initial phases of inception of an enterprise application when requirements are not crystal clear, or implementation details are not available. This estimation technique involves decisions based on heuristics. Data available in the organization from similar projects, similar domains or industry standards are used for estimation.

Use Case Point estimation is one of the popular software sizing and estimation techniques, which primarily depends on the use-case model prepared in the requirements elicitation and analysis phase. The target functionality that is depicted in the form of use case models is the basis for doing Use Case Point estimation.

Function Point estimation, popularly known as FP estimation, is based on FP counts. A function point (or FP) is the unit to measure the functional size of the software under measurement. To perform FP estimation, a fully documented functional specification of the target software solution is needed.

FP estimation in itself is a very vast area to explore and requires expertise. The way of applying FP to estimate a software solution is dependent on the nature of the project, such as whether it is an enhancement project or a green field development project. It also depends on whether estimation is being done for client/server systems or Graphic User Interface intensive systems or object-oriented systems.[5]

Whatever be the estimation methodology, ultimately the project effort estimation is used for arriving at the project duration and schedule. The duration is a function of the effort, availability of resources, various constraints and minimum elapsed time for certain activities. The schedule is typically prepared by the project manager with assistance from business analysts, technical architects and SMEs.

Now, we are all set with requirements elicited, analyzed and validated. We have planned and estimated the resources and time to raise the enterprise application. We are ready to leap into next phase of SDLC, that is architecting and designing enterprise applications.

Summary

This chapter has introduced the concept of enterprise analysis and business process modeling. The concept of "as-is" and "to-be" process modeling is introduced. The sample EM Bank's LoMS case study is introduced, which will be used throughout the book for raising an enterprise application.

This chapter has introduced the concept of requirement elicitation and analysis for both functional and nonfunctional requirements. Use case diagrams, use cases, actors and the various kinds of relationships that exist in use case diagrams are explained in the context of LoMS application. Importance of prototypes and the best practices followed for prototyping are also touched upon. Various kinds of NFRs are introduced briefly. An SRS template is also briefly discussed.

This chapter has introduced the concept of requirements validation. Several ways to validate the requirements are also discussed. Establishing a requirements traceability matrix, reviewing and ensuring the proper documentation of requirements, preparing the user acceptance test cases and preparing proof of concepts are the key ways to ensure requirements validation.

This chapter has also introduced the concept of planning and estimation of the project that undertakes the job of raising an enterprise application. The readers are also introduced to some of the popular estimation techniques like Use Case Point estimation and Function Point (FP) estimation.

Review Questions

1. Explore business process modeling notation (BPMN) and tools to perform the business process modeling using BPMN.
2. List down all the 13 diagrams supported by UML2.0. Write down the significance of each diagram.

[5] To explore more about the FP estimation, please visit the International Function Point User Group (IFPUG) portal, http://www.ifpug.org/. IFPUG is an organization that promotes the use of Function Point analysis.

Architecting and Designing Enterprise Applications 3

Learning Goals

After completing this chapter, you will be able to:

- Describe architecture, views and viewpoints.
- Explain enterprise applications in perspective of enterprise architecture.
- Create logical architecture of application.
- Create technical architecture and design of application.
- Apply object-oriented analysis and design.
- Design technical architecture layers.
- Formulate data architecture and design.
- Distinguish different kinds of data representations and modeling.
- Describe infrastructure architecture, design and building blocks.

Ever heard the Winchester Mystery House story? Way back in the 19th century, one Mrs. Winchester started constructing a grand Victorian mansion. It took approximately 40 years to build the humungous 160-room mansion. Each room had been designed one at a time with lots of modifications along the way. This led to many bizarre features. "Few staircases and doors in the mansion lead to nowhere!" was one of them. The reason is that such a huge project was built without a master design blueprint.

An enterprise application without a master design blueprint can be as mysterious (and as futile in purpose!) as the Winchester Mystery House. Needless to mention, architecting and designing are the foundation stone for raising a successful enterprise application.

The previous chapter on incepting enterprise applications focused purely on the problem domain. After eliciting and validating the problem, the next step to raise enterprise applications is to focus on the solution domain. The solution for an enterprise application starts with defining the strategic design or the master blueprint of the application—the architecture. As the solution progresses in the software life cycle, the design for enterprise application is arrived at, to start the construction for a specific planned iteration. The design and construction activities often happen in an iterative fashion.

Enterprise applications are going to solve business problems, and for that the entire business context needs to be understood. This big picture is required and is called as *enterprise architecture*. You will explore enterprise architecture and its various facets in this chapter. This chapter will also introduce the concept of application architecture and design. The concepts of logical architecture, technical

architecture and the detailed design of an enterprise application are discussed. You also will learn about the building blocks, frameworks, design patterns, tools, standards and technologies used to architect and design enterprise applications.

This chapter also introduces the concept of data architecture, various kinds of data stores and data modeling. You will also explore the building blocks of infrastructure architecture and design—networking, internetworking and communication protocols, IT infrastructure and policies and an overview of middleware—in this chapter.

Towards the end, we will explore the key elements of architecture and design documentation.

3.1 Architecture, Views and Viewpoints

ANSI/IEEE Std 1471 defines "architecture" as "the fundamental organization of a system embodied in its components, their relationships to each other and to the environment and the principles guiding its design and evolution." In simple terms, the architecture is the collection of related components or structures that coexist in a given context for a purpose. Architecture serves as the blueprint for guiding the design of an application or a product.

Architecture in itself is purely conceptual. To raise an enterprise application, what matters is to understand the architecture description. *Architecture description* is the concrete artifact to depict the architecture and is organized in the form of "views" and "viewpoints".[1]

In UML is an architecture description language (ADL) in the field of software architecture among other ADLs like ACME and C2.

There are various viewpoints to an enterprise application. *Viewpoints* in general represent stakeholder's view of the enterprise application. A *view* is how the enterprise application looks like from a selected viewpoint. Each view reflects an aspect of the architecture description and is the guiding blueprint in making decisions on that aspect in all the stages of software development life cycle. For example, use cases described in the previous chapter are considered as viewpoint that helps in eliciting and analyzing requirements of an enterprise application from the perspective of its users. This viewpoint focuses on aspects like various business processes and workflows that are required by an enterprise application. This view is called the *usecase* view.

There are other architecture views and viewpoints that focus on several other aspects of an enterprise application. All these views and viewpoints together constitute a comprehensive architecture description of the enterprise application. More views, viewpoints and their applicability are explored in this chapter.

An architecture view, in turn, consists of one or more models. These models are used to depict the design of a part of the enterprise application from a certain aspect as represented by the view. Taking the same example of usecase view, it consists of usecase models that are depicted by usecase diagrams as described in Chapter 2.

---

[1] To know more about architecture and architecture description, you can refer to http://www.iso-architecture.org/ieee-1471/index.html.

Putting it simply, the architecture description of an enterprise application consists of various aspects. These aspects are depicted by various views and, in turn, each view is represented as one or more appropriate models using languages like UML.

Xp

While views are interdependent, initially each view should be modeled independently to ensure "separation of concerns". Subsequently the inconsistencies, interdependencies and conflicts across the different views are reconciled to optimize the structure and behavior of the system.

3.2 Enterprise Application—An Enterprise Architecture Perspective

3.2.1 Enterprise Triangle and Enterprise Architecture

An enterprise can be represented as an enterprise triangle, as depicted in Figure 3-1. The enterprise triangle represents the coexistence of users, business applications and data, and their interactions. End users in enterprises use business applications to access and manipulate the enterprise data.

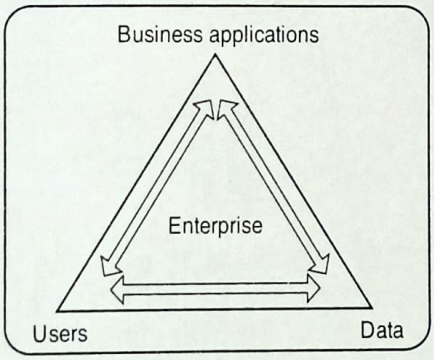


Figure 3-1 Enterprise triangle.

Enterprise architecture helps conceptualize, represent, understand and analyze these business scenarios, keeping business applications, user and data under consideration. Enterprise architecture, as depicted in Figure 3-2, in turn, consists of several kinds of architectures and their building blocks. It can also be understood as the architecture description of the entire enterprise business and the enterprise-wide IT systems as a whole. You will explore these architectures and their building blocks throughout this chapter.

Enterprise architecture is the holistic architecture of the enterprise that explains the building blocks, or elements, of an organization and the synergy between these building blocks. It is the mechanism to facilitate the IT-business alignment with a tight tab on the Return on Investment (RoI) by enabling the organization to figure out redundancies in the business and the IT alike.

Enterprise architecture acts as the key to be at the helm of business through alignment between business and IT. It throws light on the interrelationship between the resources, capabilities and service offerings of the enterprise, helps in managing the business complexities and acts as a catalyst in formalizing the business and IT roadmap for the enterprise.

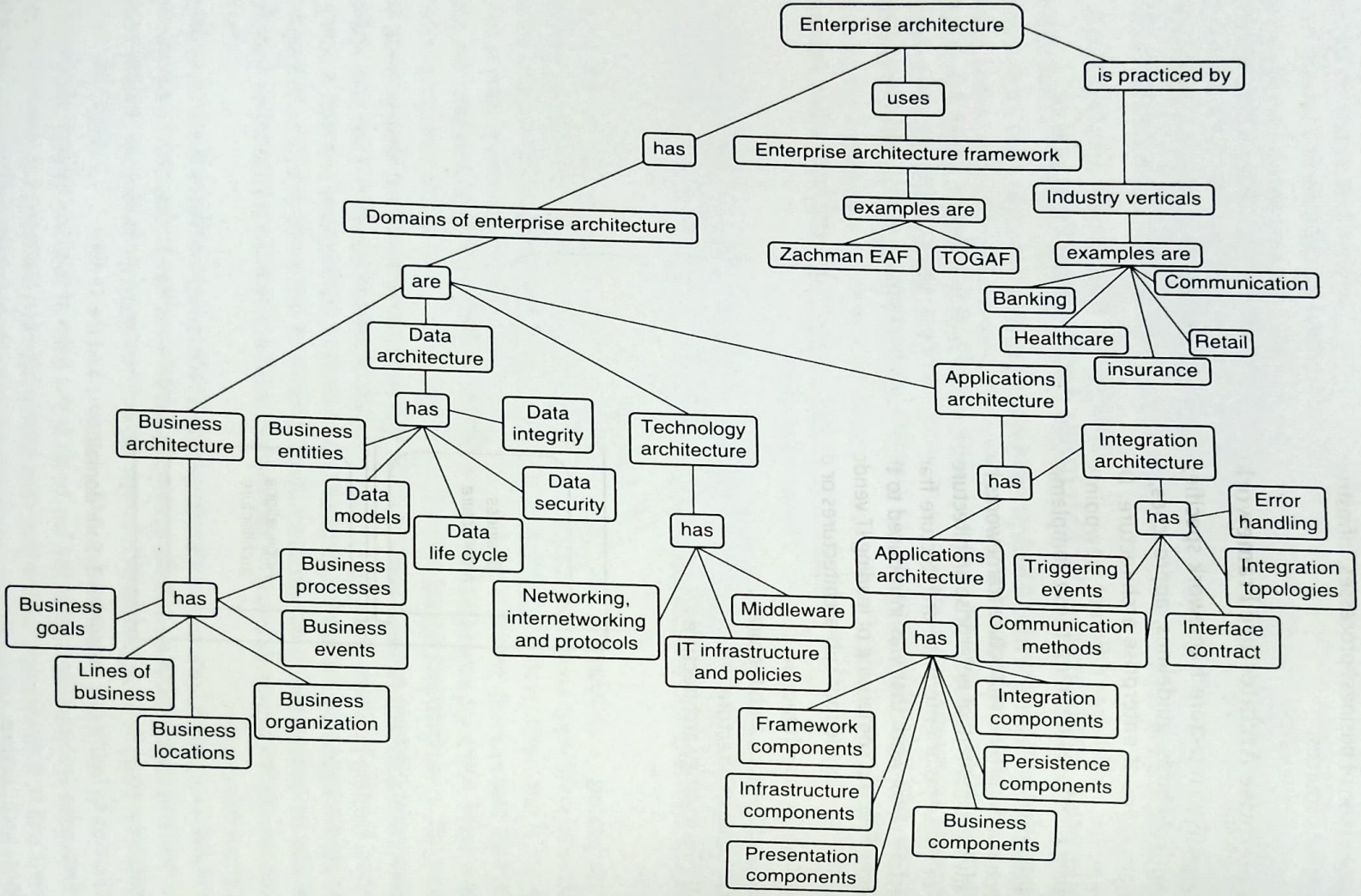


Figure 3-2 Enterprise architecture—an overview.

Enterprise architecture plays a key role in managing the complexity of several aspects such as applications' landscape, business processes mapped to the applications, varieties of technologies used in implementation of business processes, infrastructure to deploy the applications and management of enterprise data sources.

3.2.2 Enterprise Architecture Frameworks

An enterprise architecture framework specifies a set of views and viewpoints and comprises of principles, standards, guidelines, approaches, services, design concepts and models to facilitate the development of enterprise architecture. Different stakeholders in the organization have different purposes and hence different viewpoints to view the business. The enterprise architecture framework facilitates defining these complementary enterprise views for various stakeholders in the organization.

Several enterprise architecture frameworks have evolved over a period of time to govern the definition and management of enterprise architectures. The Open Group Architecture Framework (TOGAF) is one of the popular enterprise architecture frameworks. TOGAF is an open and generic enterprise architecture framework that was introduced to the industry in the mid nineties by the Open Group's Architecture Forum comprising of leading IT vendor organizations and customers.

TOGAF defines the four sub-architectures or domains of enterprise architecture (see Figure 3-3):

(a) Business architecture
(b) Applications architecture
(c) Data architecture
(d) Technology architecture

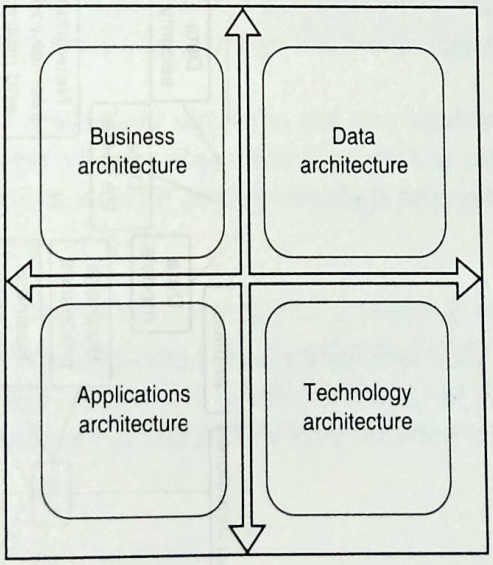


Figure 3-3 Four domains/architectures of TOGAF.

Business architecture

Business architecture focuses on the overall strategy and governance of the business processes and other key elements of the business such as business goals, lines of business, business locations,

business organization and business events. It provides the basis for defining the enterprise applications that will achieve the goals of the business. You may recall from the previous chapter the role of business modeling of the enterprise in the context of conceptualization of one or more enterprise applications.

To understand an enterprise application one should start with understanding the business processes it supports or maps to. From the perspective of an enterprise application, business architecture is restricted to the description of related business processes and modeling them to represent the business architecture. Typically, related business processes that need to be automated are mapped to an enterprise application or a suite of related enterprise applications catering to a particular functional area of the enterprise business.

Enterprises that belong to a specific industry vertical or domain typically have many common elements in their business architectures. These similarities are generally captured as part of the industry specific enterprise architecture frameworks.

In TM Forum's New Generation Operations Systems and Software (NGOSS) is the enterprise architecture framework for the telecommunications domain. Enhanced telecom operations map (eTOM), which is a component of the NGOSS framework, is a repository of standard telecommunications service providers' related business processes.

Applications architecture

Applications architecture (as distinct from *application architecture* that represents the structure of an individual application) is the structure of interconnection and interactions of all the applications in an enterprise and maps to the underlying enterprise business processes. This typically comprises of both the architecture of individual applications and the integration architecture, binding these applications together.

Applications architecture deals with the enterprise applications landscape and how individual applications are mapped to the underlying business process of the organization. It describes the synergies among these set of enterprise applications acting together to implement the objectives of the enterprise.

Data architecture

Data architecture focuses on the overall data assets—both logical and physical—that an organization possesses. It typically includes the business entities, data models, data life cycle management, data security, and data integrity mechanisms. Data architecture is sometimes referred to as *information architecture*.

Technology architecture

Each enterprise has to ensure consistent user experience, scalability, availability, security, disaster recovery and various other things to ensure business effectiveness and continuity. These are determined by the IT infrastructure, networking, internetworking, communication protocols and middleware capabilities of the enterprise. This viewpoint of enterprise architecture is termed as *technology architecture* and at times used interchangeably with the term *infrastructure architecture*.

The technology architecture consists of building blocks that include the middleware components used to glue the applications, IT infrastructure hardware and software and networking and communication protocols.[2]

Some other popular enterprise architecture frameworks are as follows:

- Zachman Enterprise Architecture Framework
- EAF from Gartner
- C4ISR for Department of Defense (DoD), USA Government

In addition to the above generic enterprise architecture frameworks, a few domain-specific frameworks, NGOSS mentioned earlier being one of them, exist. There are a myriad of tools and resources available for architecting the enterprise and creating the blueprint of enterprise architecture. The tools are predominantly based on one or more enterprise architecture frameworks.

Tt Some of the popular resources related to enterprise architecture are: (a) Enterprise Architect: http://www.sparxsystems.com/, (b) Microsoft: www.msdn.microsoft.com/architecture and (c) Gartner: http://www.gartner.com/it/products/research/asset_129493_2395.jsp

3.2.3 Blueprint of an Enterprise Application

An enterprise application exists in the ecosystem of various other applications and multiple data sources to fulfill a certain business role in the enterprise. Enterprise architecture helps understand how an application fits within the existing enterprise landscape, and guides arriving at the blueprint of an application. The blueprint of an enterprise application consists of the following viewpoints or perspectives:

- Logical architecture
- Technical architecture
- Data architecture
- Infrastructure architecture

Each perspective or viewpoint plays an important role for a particular focus group/team in the software development life cycle (SDLC). For example, an in-depth understanding of data architecture is required by the database team, or an in-depth understanding of infrastructure architecture is required by the deployment team. Let us explore the four perspectives of an enterprise application's blueprint.

Logical architecture

Elements of a business architecture define the functionality of a given enterprise application. This functionality can be mapped to a set of logical building blocks, implementing the enterprise application. These logical building blocks together define the logical architecture of an enterprise

[2] To know more about TOGAF, please visit http://www.opengroup.org/togaf/.

application. The logical architecture serves as the blueprint to the architects to arrive at an optimal enterprise application solution that fulfills the objectives laid out by the business architecture.

Technical architecture

The next step is to realize the logical architecture as a technical implementation of the solution. The technical architecture serves as the starting point of the implementation and consists of application frameworks, design patterns, APIs and programming languages. Appropriate mechanisms for integration of the applications are also identified as part of technical architecture to achieve the integration of enterprise applications within the applications' landscape.

The primary goal of technical architecture is to provide simple, maintainable and robust design building blocks that can be easily extensible to accommodate the future changes in the business rules of the enterprise. Technical architects take input from the logical architecture to lay down the technical architecture of the application. Later architects work with the software design team to define the detailed design of the enterprise application.

Data architecture

Data architecture, as part of the enterprise architecture, deals with the universe of data within the entire enterprise and focuses on the business significance of the data. The data architecture in the context of a given enterprise application has its focus primarily on identifying, modeling and defining the data which needs to be processed by the enterprise application. Data architecture is typically looked at from three viewpoints or perspectives:

- *Conceptual viewpoint:* defines all the business entities in an enterprise related to the given problem domain
- *Logical viewpoint:* defines the logical representation of all the business entities and their relationships with respect to a specific enterprise application
- *Physical viewpoint:* defines the physical implementation of the entities and their relationships within an enterprise application

A business analyst typically starts with identifying the business entities from the requirements gathered. Architects and designers map these business entities to the logical design of the data that eventually gets realized as physical database design.

Infrastructure architecture

An enterprise application can be deployed on the targeted hardware and network infrastructure in various ways depending upon the deployment requirements of the application. The infrastructure architecture describes the physical hardware and the supporting systems software that supports the deployment of the enterprise application.

The infrastructure architects identify the types of servers, for example Web servers, file servers, application servers, groupware servers and database servers, the physical locations of deployment, backup plan, storage, connectivity points, firewalls, alternate storage locations and middleware requirements for the smooth and uninterrupted operation of an enterprise application.

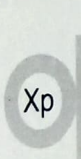
3.3 Logical Architecture

Logical architecture is used to represent the business processes in the form of logical building blocks and map these building blocks to the software application or group of software applications that an enterprise needs to raise as the solution to the business problems identified.

The logical architecture provides a bridge between the requirements captured by the business analysts and the technical solution that will be implemented by the development team. Architects define the logical architecture by identifying and putting together the logical components of the solution. Logical architecture is not the technical architecture. It is independent of any technology and doesn't dictate the tools, frameworks or technical components required to realize the enterprise application.

The logical architecture and its components vary based on the problem domain they address. Several structures are possible for organizing the enterprise application solution, but logical architecture is typically represented in a layered fashion wherein each layer hosts a related set of logical components.

The layered architecture pattern is about dividing the services into modular layers such that the interactions are limited to the adjacent layers. The concept of dividing an enterprise application into layers is very important and proven one. This enables the concept of "the separation of concerns" that leads to desirable properties such as loose coupling, scalability, reusability and modularity in the application. One of the representative examples of the layered architecture pattern is the networking models like ISO-OSI and TCP/IP.



While layers at higher levels normally access functionality belonging to the immediately lower layer, in a physical implementation, higher layers could request any of the lower layers for services to implement their functionality.

The logical architecture of a typical enterprise application is depicted in Figure 3-4. It is comprised of the following layers:

- Presentation layer
- Service access layer
- Business layer
- Data access layer
- Integration layer
- Enterprise information systems layer
- Infrastructure services layer

Let us start with understanding the *business layer*, which is the core layer of the enterprise application. Business layer can be considered to comprise of three sub layers to represent business processes, business services and business components. *Business process* or *service orchestration layer* is the highest level of abstraction of the business layer. It represents the business processes implemented in the application. Business processes, in turn, are composed from one or more business services and are represented in the *business service layer*. *Business component layer* represents the lowest level of abstraction of business layer in terms of underlying business entities.

The enterprise application has to connect with the database or other storage mechanisms to manage persistence of the application's data across invocations. In Figure 3-4, it is represented as *data layer*. Certain business services implemented by an enterprise application may require data from external applications, which necessitates their integration with external applications. *External systems layer* is used to represent the external applications with which the target application has to integrate with. These two layers are often combined together and are referred to as the *enterprise information systems* (EIS) *layer*.

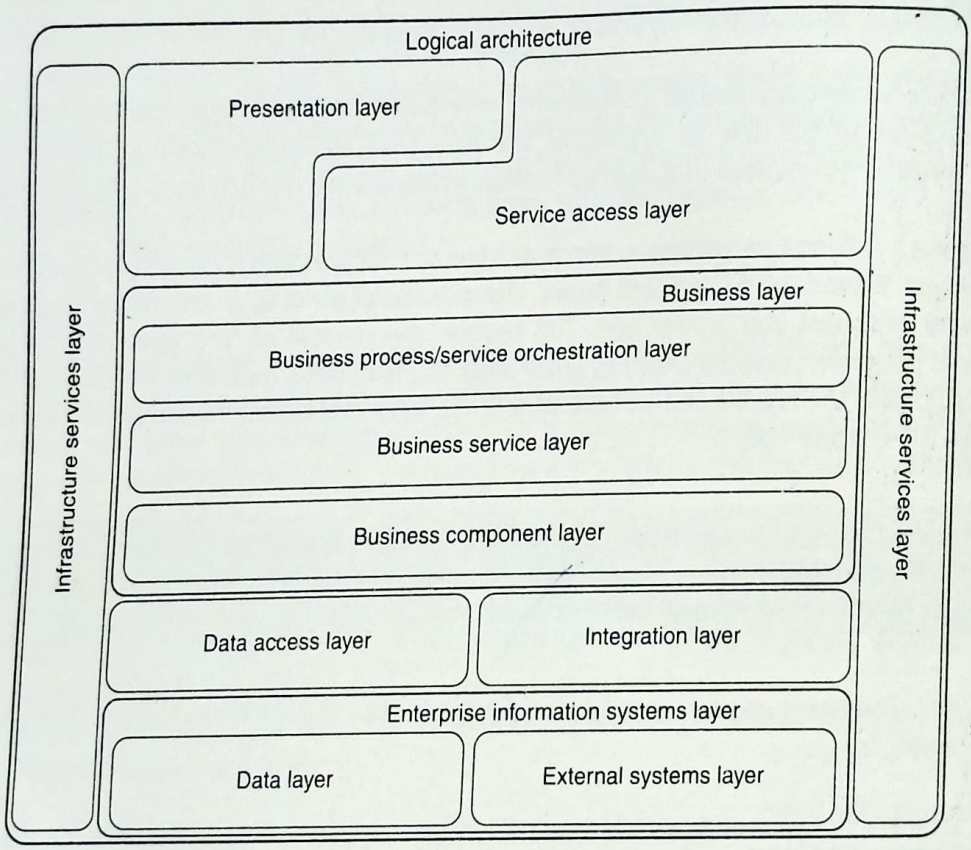


Figure 3-4 Generic logical architecture of an enterprise application.

The *data access layer* and the *integration layer* provide the abstractions that hide the physical details of storage and access mechanisms from the business layer. This insulates the business layer from the implementation specifics and any changes that could be introduced at a later point in time.

The business processes or business services can be exposed to the human users or the external systems. *Presentation layer* is used by the human users, and the *service access layer* is used by the external systems to access the business processes or business services hosted by the enterprise application.

The *infrastructure services layer* provides the reusable and general purpose components like logging, caching, auditing, etc., which are required by various other application layers. In principle, some of the services provided by the data access layer and the integration layer may be considered as part of the infrastructure services layer.

Let us apply this generic architecture to define the logical architecture for LoMS. Based on the requirements gathered, the logical architecture of LoMS is depicted in Figure 3-5. In practice, there may not exist a strict one-to-one mapping between the requirements and the components of the logical architecture.

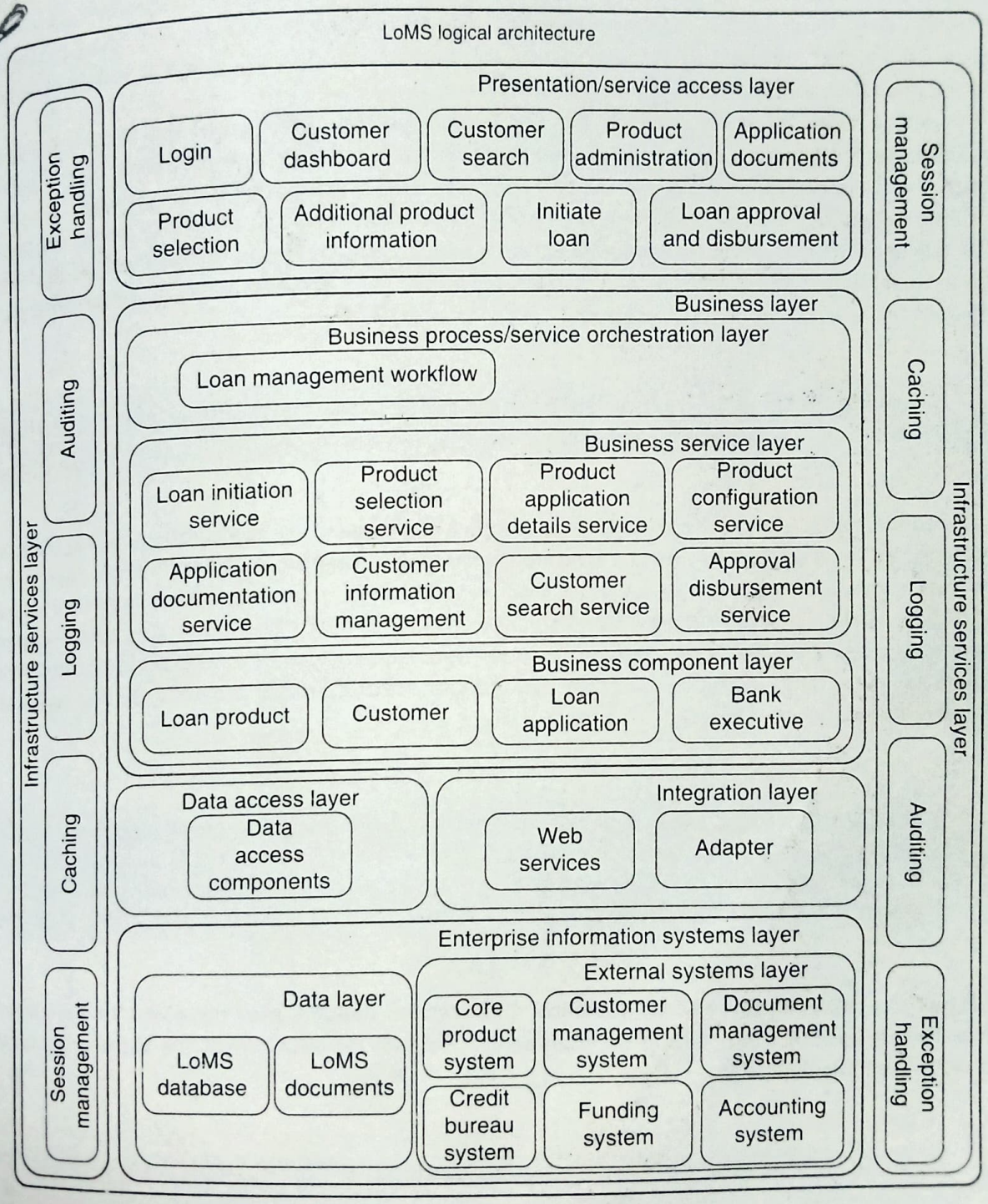


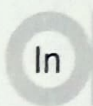
Figure 3-5 LoMS logical architecture.

3.4 Technical Architecture and Design

3.4.1 Mapping Logical Architecture to Technical Architecture

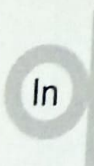
As mentioned earlier, the logical architecture serves as the blueprint for implementing the enterprise application solution. The architects and design team use logical architecture, along with other artifacts, created in the requirement engineering phase—use case specifications, nonfunctional requirements and system requirements specification—to define the technical architecture. Technical architecture translates the logical architecture into a technical solution in terms of specifying the architecture layers using the chosen technologies, APIs and frameworks.

The *software architecture patterns* are widely used in the definition of the technical architecture. They are proven solutions to widely recurring software architectural problems. These patterns describe the fundamental structure of the software elements participating in the software system and the relationships that exist among these elements. These patterns act as reference points from which the technical solution of an enterprise application is arrived at and used in further analysis of the nonfunctional qualities of the system such as performance, reliability and availability. As mentioned earlier, LoMS uses the layered architecture pattern.



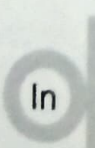
Apart from the layered architecture pattern, pipe and filter architecture pattern and broker architecture pattern are a few of other software architecture patterns.

The conceptualization of the technical architecture of an enterprise application typically starts with identifying the technical layers and the common components that are required in these layers. A key trend that is invariably noticeable in most enterprise applications implementations is the use of *application frameworks.* The application frameworks are the standard mechanism to structure the generic components used in the implementation. *Application components* are the realization of the usecases achieved by extending and/or customizing the generic functionalities provided by the application framework components.



An application framework is seldom designed as part of the enterprise application development itself. It typically the case that it is already developed as part of the organizational mandate to use the common application framework across all application development projects in a specific technology.

Figure 3-6 represents the LoMS technical architecture. Readers may observe that the technical architecture is derived from the logical architecture through introduction of the technology components into the individual layers.



Layers and tiers are generally used interchangeably in the technical architecture context. Layers are logical while tiers are considered to be physical. Hence, tiers may be distributed across multiple physical or virtual machines based on performance and availability considerations.

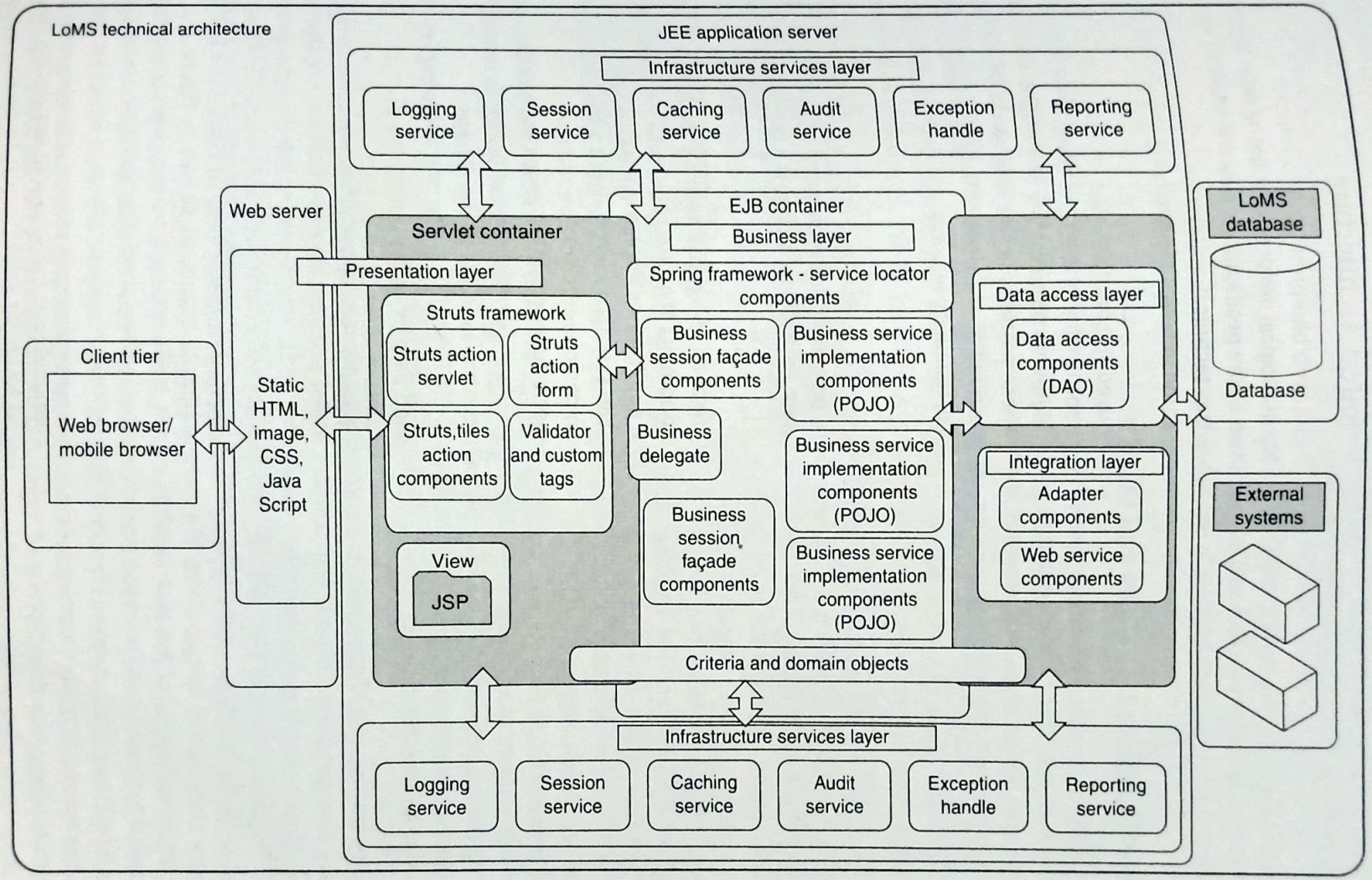


Figure 3-6 LoMS technical architecture.

In parallel to the presentation layer, there may exist a service access layer whose function is to expose the business services/processes to the nonhuman users, that is external applications via mechanisms like Web services.

For implementing these layers, several paradigms may be used—the most widely followed today is object-oriented paradigm—which will be examined in the next section. You will explore the technical architecture in a layer-by-layer manner in the following sections.

3.4.2 Object-Oriented Analysis and Design

Object-oriented techniques, which are extensively used in modern-day software development, touch several stages of the SDLC. Object-oriented analysis and design (OOAD) technique used during the architecture and design phase helps model the software entities as counterparts of the real-world objects. To implement the technical solution, object-oriented programming (OOP) is used during the construction phase.

Objects and classes

Objects and *classes* are the primary concepts that underlie OOAD. Definition of the technical solution starts with identification of objects and classes. An object mirrors a real world entity in the domain of the solution being built. It encapsulates the entity's state at runtime. A class is a template or a blueprint for a type of objects and has a set of attributes and behaviors. For example, in LoMS, `Customer` is a class from which `customer` objects are instantiated.

Abstraction and *encapsulation* are two key concepts that are central to OOAD and are closely related. Abstraction defines what an object does, and encapsulation defines how an object works and hides the internal details. *Inheritance* and *polymorphism* are two other distinguishing characteristics that help in arriving at solutions that are maintainable and extensible in tune with changing requirements, and are heavily exploited in OOAD.

OOAD proceeds in distinct steps to provide a stable foundation from which the solution can evolve in a systematic fashion, as explained here. The design team identifies critical usecases among the ones listed during the requirements engineering phase and starts identifying the classes.

Considering the various use cases documented in the inception phase, the following key categories of classes may be identified:

- **Entity classes.** The classes that encapsulate the core business entities are called *entity classes*. They emerge directly from the vocabulary of the problem domain. Domain entities and other key domain concepts along with their attributes and behavior are modeled by the entity classes. It is important to note that only domain aspects are emphasized and implementation aspects are carefully avoided in specifying entity classes. For example, in LoMS, `LoanProduct` is an entity class, which encapsulates the loan products' details, as shown in Figure 3-7, which is a UML class diagram. You may recall that UML is introduced as a visual modeling language in Chapter 2. Usecase diagrams were used to represent requirements, as experienced by the users of the system. *Class diagrams*, another type of UML diagrams, are used to model the software entities of the solution. Each entity is represented as a rectangle that has three sections: the first section represents the class name, second section represents the attributes (characteristics) and the third section represents the methods (behavior).

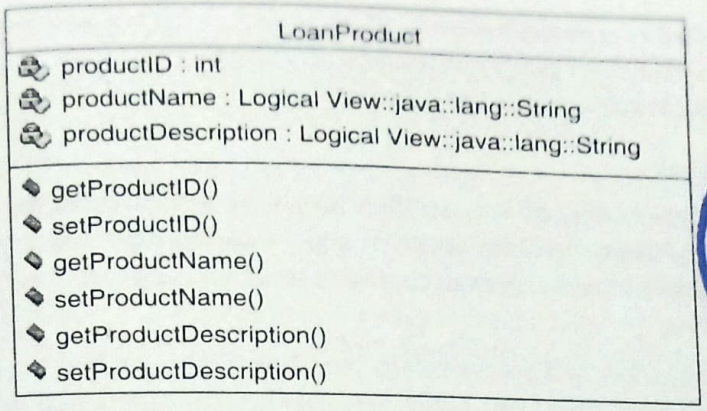


Figure 3-7 LoanProduct—entity class.

- **Boundary classes.** The classes that encapsulate all aspects of interactions with the actors are called *boundary classes*. Boundary classes are primarily created to model user interface interactions. For example, elements of data entry forms are encapsulated as the boundary classes. In LoMS, `InitiationForm.jsp` is a boundary class, which encapsulates the interaction of 'Bank Representative' actor who initiates the loan on behalf of a customer. On the other hand, in case of external system actors, boundary classes typically encapsulate the logic of exchanging data with the other systems. Similarly, a device interface boundary class may be required in cases where the system interacts with an external device.
- **Data store classes.** *Data store classes* encapsulate the design decisions like how enterprise applications store and retrieve data from the application's data stores. For example, `InitiateLoanDAO` in LoMS, as depicted in Figure 3-8, is a data store class which encapsulates the logic of accessing and persisting the customer details in the underlying database.

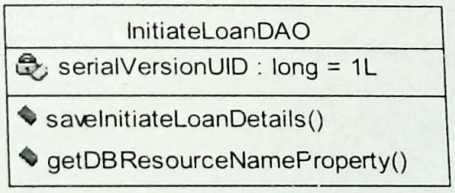


Figure 3-8 InitiateLoanDAO—data store class.

Xp

Typically for one set of entity classes there may be multiple sets of boundary classes and data store classes, or vice versa.

Designing classes is not a onetime affair. It's an iterative process and the classes get refined as the design phase continues. It starts with identifying a name for the class along with the attributes and operations of the class. Attributes are being associated with properties such as data types, access scopes, access specifiers and access modifiers. As the design phase continues, operations are also associated with the classes. Properties such as name, parameters and their description, visibility, scope, default value and data ranges are also identified with respect to the individual operations. As the design

phase progresses, classes may get aggregated, re-factored or associated via new relationships. Initial class design focuses on the domain perspective, but as the design process matures, nonfunctional considerations also need to be factored into the class design.

Relationships among classes

The realization of the functionality of the application is achieved through collaboration among the classes. Hence it becomes important to identify the relationships among classes and among the instances of the same or other classes. Primarily, there are two types of class relationships: association and generalization.

(a) **Association.** *Association* is a relationship that exists among the objects of the classes where they are aware of each other for fulfilling a particular functional requirement. For example, `Customer` can initiate a loan. Therefore, `Customer` and `LoanProduct` objects are associated with each other. A customer can initiate loan for many loan products, whereas a particular loan product type can be associated with multiple customers. Figure 3-9 depicts the association relationship between `Customer` and `LoanProduct` classes.

In an association relationship, *multiplicity* and *navigability* impact the class design. Multiplicity represents the number of objects of a particular class that may participate in a given association. For example, each customer can apply for multiple loan product types and a given loan product type can be applied for by one or more customers. This is represented in the class diagram by placing multiplicity indicators, `1..n`, on each side of the association. Navigability depicts the direction of navigation in a relationship. If the requirement says 'Given a loan product, find out the name of the customers who have applied for this loan product', the navigability requirement is that the relationship should be navigable from `LoanProduct` to `Customer`. Likewise, if there exists a requirement to be able to 'find out the number of loan products applied for by a given customer', the relationship should be navigable also in the other direction, as shown in Figure 3-9.

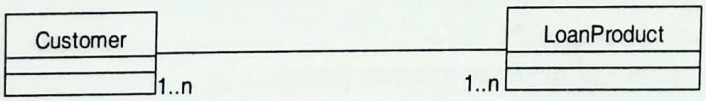


Figure 3-9 Association between `Customer` and `LoanProduct`.

Variations of the association relationship exist like aggregation and composition. Aggregation relationship represents the "whole-part" relationship between the objects of the classes. For example, `Customer` and `LoanDetails` exhibit the aggregation relationship, as the customer's loan details are part of the `Customer`, as shown in Figure 3-10.

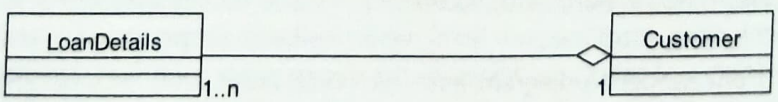


Figure 3-10 Aggregation between `Customer` and `LoanDetails`.

Composition relationship is a complete "whole-part" relationship between classes, where part object cannot exist without the whole object, and it is always the whole object that creates the part object. For example, the *inner classes* of the Java language are a mechanism to realize the composition relationship in an elegant fashion.

(b) **Generalization.** *Generalization relationship* represents "is a" or "a kind of" relationship among classes. It is more commonly referred to as *inheritance*. To identify this relationship, the design team usually starts by finding similarities among dissimilar things. The similarities are grouped in the parent or the base class, and the dissimilarities are extracted into the child or sub classes. For example, mortgage loan and personal loan are two types of specialized loan products. As shown in Figure 3-11, `LoanProduct` is the generalized class, whereas `MortgageLoan` and `PersonalLoan` are the specialized classes of the `LoanProduct`. Contrary to the association relationship, multiplicity and navigability do not exist in case of generalization relationship.

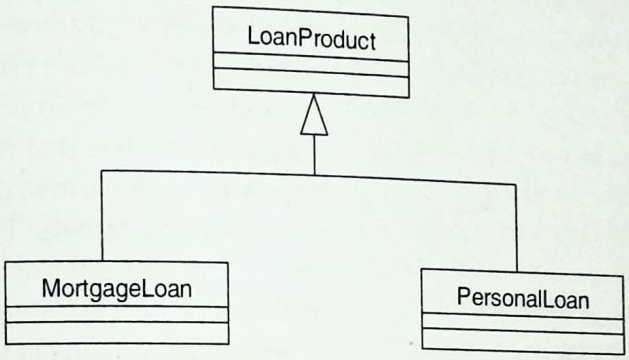


Figure 3-11 Generalization relationship in loan products.

In this section, class diagrams (Figures 3-7 through 3-11) were created to represent a partial view of the system. There are two aspects to the model of an application—static and dynamic. The class diagrams model the static aspect of the application. To design the dynamic view of the system, other UML diagrams are used. A *sequence diagram* (discussed later in the chapter) is used to visualize the sequence of actions, and is an important tool for a designer to convey the sequence of message passing or method calls among classes in a given use case. Contrary to the sequence diagram, *collaboration diagram*—another UML diagram to capture the dynamic view—depicts how classes collaborate rather than the sequence of actions.

Tt While UML diagrams can be drawn by hand, they are often created using modeling tools like Rational Rose, Rational Software Architect, Rational Software Modeler and Together Architect to name a few. Further, tools can be limited to just drawing (e.g., Microsoft® Visio®) or they can be more full blown and support features such as validation and analysis of the model. Going a step further, some tools may also allow generation of code and test cases from the model.

Apart from sequence diagram and collaboration diagram, there is another important UML diagram, called *state chart diagram* that is used to model various states that a particular object may go through during the execution of the application.

Xp

State chart diagrams are useful only when modeling objects that can potentially assume a large number of state changes based on conditions that arise during the execution of the application.

With this primer on OOP, let us move on to demystify the design of technical architecture layers. While delving into each of the layers, we will explore the building blocks, components, technologies, tools, tips, tricks and best practices followed to design the technical solution of enterprise applications.

3.4.3 Infrastructure Services Layer

In general, each layer builds upon the layer below. However, an enterprise application may require basic services like authentication, authorization, logging, exception handling and session management across the layers. *Infrastructure services layer* is the layer that provides the common services for all the other layers in an enterprise application so that all the designers and developers can use it in a uniform manner. This layer is also known as *common services layer*.

All the basic services required by an enterprise application, as shown in Figure 3-12, should be kept separate from all other layers to reduce coupling and promote reuse of code. Many a time, these services are used as-is in other applications of the enterprise with or without minimal changes. In general, as part of designing the technical solution, the architecture team establishes this layer first, as the components of this layer may be used across the application.

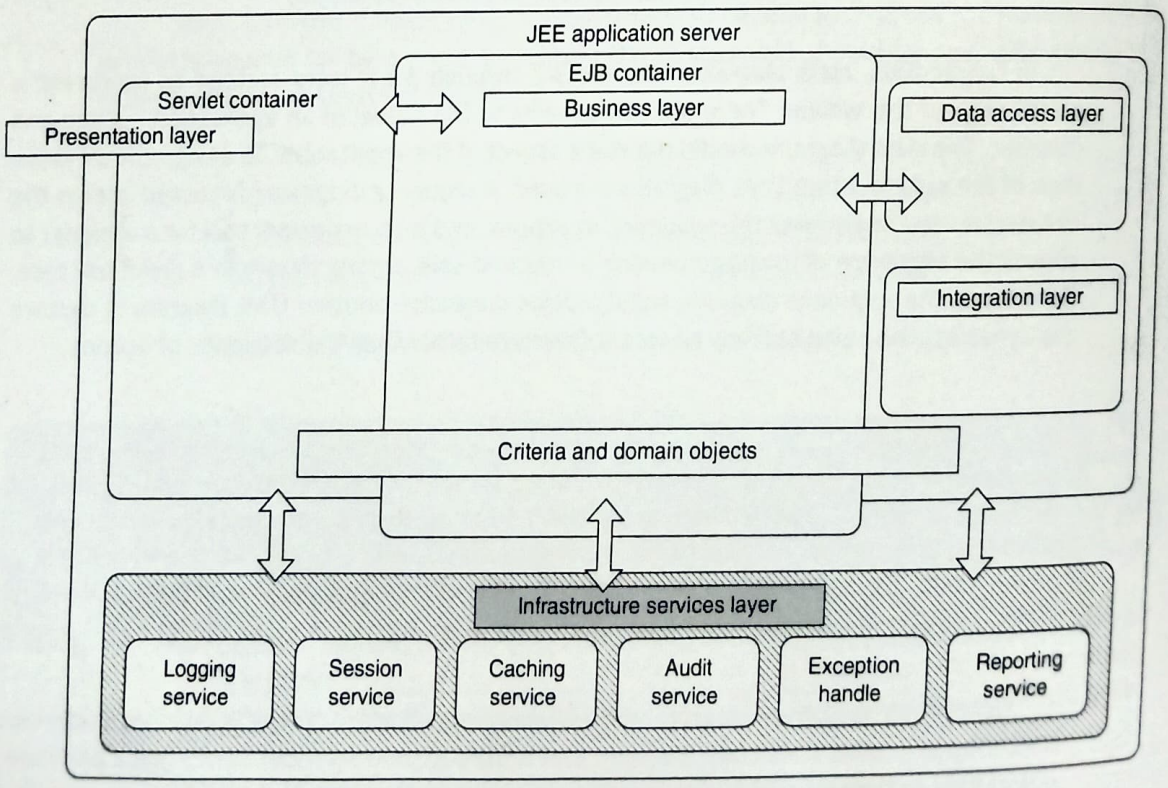


Figure 3-12 Infrastructure services layer.

Let us explore a few of the most common components of an infrastructure services layer.

Logging

For an enterprise application development, *logging* is one of the primary needs both for auditing and debugging purposes. Audit information in the logs enable application administrator to keep the track of usage of various resources by the users of the enterprise application. This might be required to satisfy the security and regulatory NFRs. Trace information generated during the course of the application execution helps the developers and the administrators diagnose various conditions and events and, if necessary, use them to debug the code by understanding the context of failures.

Tt

Logging, being a widely required functionality, several open source and commercial components are extensively available, and often the logging functionality in an enterprise application is implemented with such components. Apache Log4J is an open source component for logging and commonly used in Java and JEE applications.

Typically, the log statements are coded along with the source code of various layers of application. Logging components, such as Log4J, enable runtime configuration of several features of the logging mechanism without having to modify the application code. Excessive amount of logging may also have a down side in that it may result in performance slow down of the application. The logging components mitigate this by allowing configuration of the amount of log information to suit the requirements.

To illustrate the use of logging, consider the `InitiateLoan` use case that logs events like getting the preapproved loan amount based on social security number (SSN), or saving the loan details after validation. LoMS uses Log4J as the underlying logging mechanism. However, in order to isolate the application from any specific third party component, LoMS application framework encapsulates the Log4J API by providing a wrapper that hides the specific details of Log4J.[3]

Xp

Though logging and exception handling are often interrelated, it is possible that they may be used independently in some places in an application. Hence it is considered a good practice to decouple their design and implementation.

Exception handling

Exceptions occur whenever the normal flow of program instructions is disrupted due to an event, for example, passing a null argument in a method call may raise a `NullPointerException`. Such exceptions need to be handled appropriately to either resume execution of the application after necessary remedial action or to gracefully terminate the application where recovery is not possible.

Exception handling is an elegant and robust mechanism by which an application can catch and handle disruptive events during the execution of the code. Exceptions in a Java application may be either thrown by the underlying Java virtual machine or the application itself. The strategy for exception handling should be well thought of by the architects. A few of the points to ponder upon are as follows:

[3] Chapter 4 provides more details on the implementation of logging.

- There are various exceptions predefined by Java API packages such as `java.lang` and `java.io`. These exceptions should be used in preference to user-defined exceptions, wherever applicable. For example, it is better to use the predefined `StringIndexOutOfBoundsException`, rather than create a user-defined exception to check when the program attempts to access a character beyond the bounds of a string.
- A meaningful message helps easy interpretation or diagnosis of the condition that caused the exception. An exception should contain enough information to indicate the type, location and description of the problem. The common practice of novice developers of just printing the stack trace while handling the exception not only makes the application messages meaningless to the end user but may also expose unnecessary details about the internal structure of the application, which may also sometimes result in compromising the security of the application.
- Nesting of try/catch blocks may pose code maintenance problems. It is always a recommended practice to eliminate nested try/catch blocks by restructuring the code.

As an example of exception handling, the `InitiateLoan` use case in LoMS should provide a strategy to handle input of an invalid SSN number of a customer.[4]

Session management

Support for multiple concurrent users in an enterprise application necessitates tracking the activity of the users in the course of their interactions to ensure the integrity of various shared resources. In a typical *n*-tier Internet or intranet application, the end users generally use HTTP protocol as a medium to communicate. HTTP, being a stateless protocol, does not keep track of the state of the application across multiple requests. However, many application scenarios require remembering the series of state changes during the interactions. Web applications use the notion of a "session" to track the state changes. A session is a series of requests originating from the same user (agent) that enables the application to keep track of the activities of an individual application user.

Session management can be implemented in several ways. One of the ways is to track session by using a session ID, which is a unique identifier associated with the session. Information associated with a given user's session is stored on the server and keep track of using the session ID as a key. Session information typically includes data such as user ID, credentials, locale (language preference) and session context data, resulting from earlier requests. The session ID is typically generated upon successful authentication of an application user.

A few points to consider for effective session management are as follows:

- **State management and performance.** State management has often impact on performance and hence needs to be managed in an efficient manner. Session data is often stored in primary memory for instant access. However, where audit requirements necessitate persisting the session data, database or file system may be used as storage.
- **Clustering.** Normally enterprise applications use clustering of servers to provide redundancy for fault tolerance and load balancing. In such a case, the session state will have to be consistent and requires it to be replicated across the servers in the cluster. This is achieved by declaring session attributes as persistent and serializable so that they can be processed across servers in the cluster. You will learn more on clustering in the infrastructure architecture section of this chapter.

[4] Chapter 4 provides details on the implementation of exceptions and their handling.

In LoMS, session data is stored in primary memory and is created for each user when the user is successfully authenticated. It uses JBoss POJO cache solution to implement this feature.[5]

Caching

Caching is a proven technique for improving the performance of an application and is one of the most important components of the infrastructure services layer. Usually, the application's data is stored in persistent storage mediums like file system or relational database. Frequent access to these mediums, in order to perform any required processing, is an overhead and may cause performance bottlenecks in an application. The application may slow down significantly or even crash on receiving too many concurrent requests. Caching is one of the techniques that is effective in overcoming such problems.

Caching involves bringing frequently used data to the high speed primary memory from the secondary store where it is persisted so that the application can readily access it. This eliminates the need of repeatedly creating the connection to the underlying storage mediums. Such data may also be shared across processes and the users requiring it. This simple technique helps improve the performance of the application several fold in most scenarios. However, caching requires additional effort to manage cache size in an optimal fashion and guarding against the possibility of sometimes serving up stale data.

During design phase, application designers decide about the "mostly static" components of data that rarely change at run time of the application and needs to be accessed frequently. These data need to be cached. There are different ways of caching frequently used data. Two of the approaches are described below:

(a) One approach is to *pre-fetch* the data at application startup and store it in the cache. Upon user requests during the execution of the application, the data is fetched from the cache instead of the data source in secondary storage.
(b) The second approach is *lazy loading*, wherein the data is retrieved from the data source at the time of the first request, and thereafter the data may remain stored in the cache.

The cached data may be replaced when it is no longer needed and is controlled by a suitable caching algorithm based on several factors such as expiration time, size of cache and access patterns. To implement a cache, a data structure such as a `hashtable` is commonly used in which a key is associated with an object to store it in cache and later used to lookup the object.

Tt JBoss Caching is an open source POJO cache solution, which provides object-oriented solution for implementing caching services in an enterprise application.

In LoMS, JBoss Caching is used to build the cache component. JBoss Caching supports clustered deployment and the instances may be distributed across the servers. This enables sessions to run across the cluster by replicating the session data across all the instances.[6]

[5] Chapter 4 provides more details on the implementation of a session and its management.
[6] Chapter 4 provides more details on the caching service in enterprise applications.

Security

Security is an important component in a distributed enterprise application, considering that parts of it may run on publicly accessible servers and networks. Users of the application need to be authenticated before being provided access and their actions have to be verified for suitable authorization based on their roles. Further, it may be necessary to maintain a track of users' actions for audit purposes. Security components of the infrastructure services layer comprise of classes, which enable the authentication, authorization and auditing requirements, more popularly known as *AAA security requirements*, of an enterprise application.

A complex enterprise often comprises of multiple systems with their own authentication mechanisms and databases of users. User IDs and credentials may be different for the same user across the systems. A typical usage may involve a user accessing one or more of these systems in a single session and, in the process, may have to provide his credentials multiple times. This could become annoying and tedious to the frequent users of these systems. To cater to this problem, enterprises typically prefer to use *single sign-on* (SSO) solutions to eliminate the need for multiple authentications of the same user within a single session. They are enterprise class security solutions that use several ways for authentication such as form based, biometrics or smart card based mechanisms. Also, *directory servers* are used as a common store for user data including credentials across the enterprise and the SSO solutions leverage these for their operation.

Tt Java Open Single Sign-On (JOSSO) is one of the Java based open source single sign-on solutions that can communicate with LDAP stores (directory servers) to authenticate the users of enterprise applications.

Other infrastructure services layer components

Besides the infrastructure services layer components discussed in the previous paragraphs, there are various other infrastructure components that might be useful across the layers of an enterprise application—auditing services, reporting services and notification services to name a few. The auditing services component enables the monitoring of an enterprise application with the help of monitoring tools. Reporting services help generate periodic or ad hoc reports on activities and operations of significance to business. Notification services take care of creating and sending personalized notifications to users based on a triggering event.

3.4.4 Presentation Layer

Presentation layer is the face of an enterprise application from the perspective of an end user. It primarily dictates the end user experience, provides the content and data in a user-friendly format and responds to end user generated events. Presentation tier serves static and dynamic content to the end user.

Static content is usually represented in HTML. Dynamic content in a Web application is generated using scripting technologies (such as JavaScript) or server side mechanisms, such as Java Server Pages (JSP) as in the case of JEE applications. JSP technology leverages the power of embedding Java code in static HTML to generate the dynamic portion of the content. Prior to JSP technology, servlets were used to generate dynamic content, but now they are predominantly used to design controllers for coordinating the model and view in a typical model-view-controller (MVC) pattern.

Tt JSP and Servlet are the key Java Enterprise APIs that form the prime components of the presentation layer.

The MVC pattern deals with decoupling the business logic ("model") from the user interface ("view"). This pattern allows all interactions among view and model to be controlled by a central entity called "controller". This approach, as depicted in Figure 3-13, helps in generating views from the underlying model.

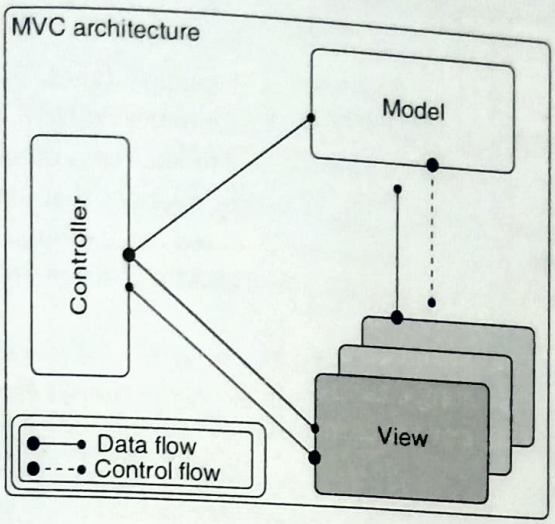


Figure 3-13 MVC architecture.

The MVC pattern allows introducing a new view without changing the underlying model. It allows flexibility in terms of changing the interaction (view navigation) model without significantly altering the User Interface code. Some may argue that the down side of MVC architecture pattern is that it introduces complexity in the solution as more abstractions and classes are required to be created in order to realize this pattern, but this is more than offset by the benefits.

Presentation layer acts like a bridge between the user and the enterprise application, as depicted in Figure 3-14. The key to the presentation layer design is its flexibility and extensibility since the presentation layer is the component that is most prone to change.

Frameworks and design patterns

In a typical *n*-tier Internet or intranet enterprise application, MVC pattern defines the blueprint for the overall presentation tier. The reason is fairly simple, as mentioned in the previous section, the MVC pattern brings in the flexibility of keeping model data and view separate, making the entire presentation more flexible. Though technically it is possible to keep the entire presentation logic in either Servlet or JSP, it's not at all flexible and maintainable.

In the JEE world, there are many other well-defined design patterns associated with each layer. Front Controller, Session Façade, Data Access Object, Business Delegate, Value Object, Value and List Handler are a few of the JEE design patterns that are used across the layers. Design patterns are the common design solutions to the commonly recurring problems, which are tried and tested best practices. They provide reliable guidelines to the design team to develop an enterprise application. By

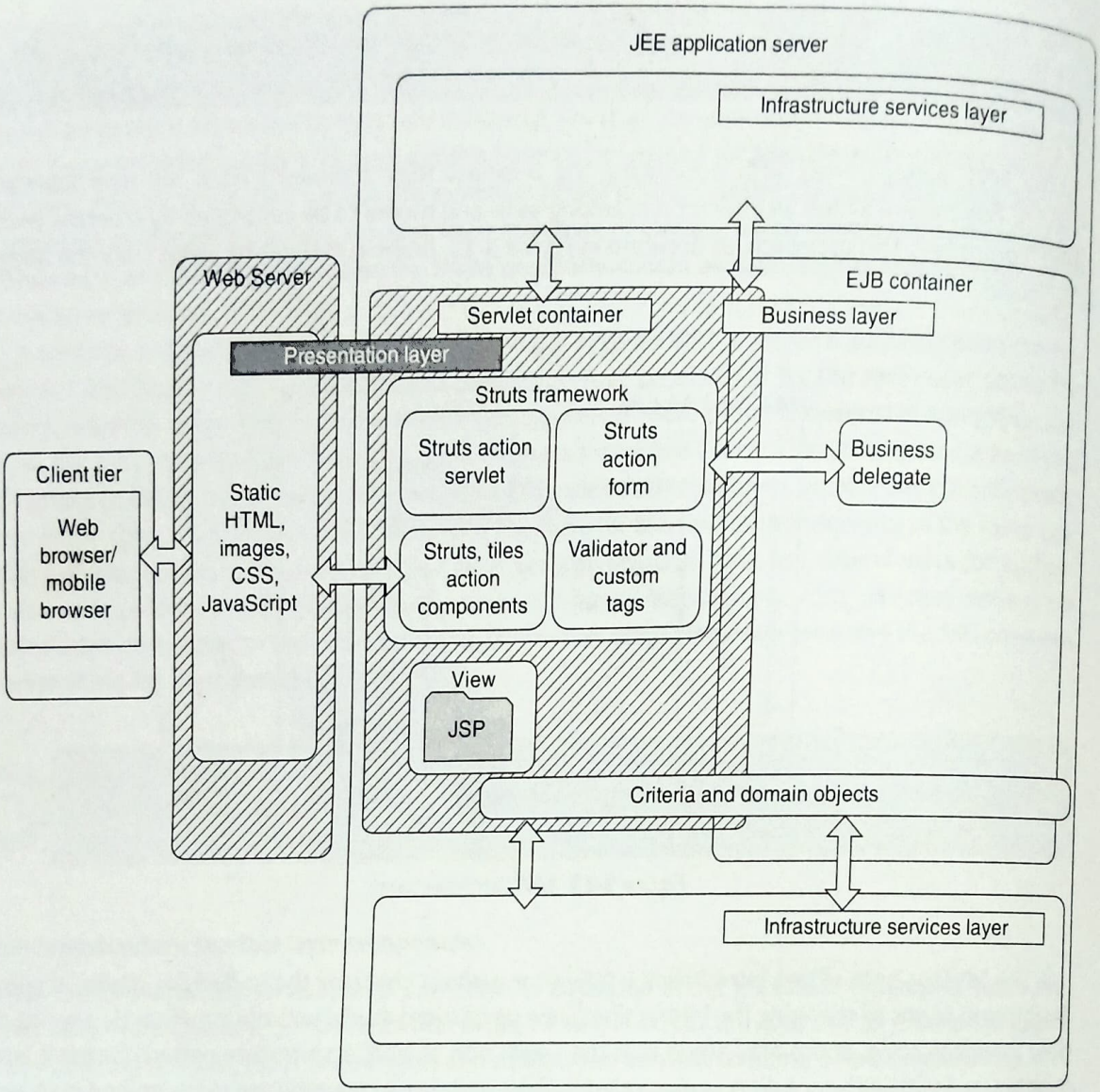


Figure 3-14 Presentation layer.

applying appropriate design patterns a portable, scalable, reliable and maintainable enterprise application can be raised.

A *framework* provides a working implementation of the guidelines and design patterns for the architectural layers of an enterprise application. Building applications from scratch is not a preferred and viable option. It is important for the design team to first evaluate the features of the existing frameworks to make sure if the required functionalities are already available within those frameworks in which case reuse should be the preferred alternative to developing the functionalities from scratch. The JEE platform has various frameworks available for the development of specific layers.

Tt Struts has been the most popular framework for implementing the presentation layer. It is a mature and open source framework which follows MVC design pattern. However, other technologies/frameworks such as JSF and Spring MVC are catching up.

Struts framework ensures compliance of the presentation layer to the MVC pattern. It must be noted, however, that in a real-world enterprise application, several other frameworks may be used in conjunction with other frameworks to implement the overall solution. In general, the selection of a specific framework is based on several factors. Some of the important ones are presented in Table 3-1.

Table 3-1 Selection parameters for a framework

Selection parameters	Explanation
Configurability	The framework should be easily configurable. It means that the framework services and features can be easily modifiable without requiring changes to the code.
Extensibility	The framework should be extensible with little additional effort and without impacting the rest of the application.
Independence	The framework should support generic interfaces and should be vendor agnostic.

Now, let us explore a few of the design best practices in the presentation layer—starting with more of MVC. In the MVC pattern, the controller receives the end user input via an HTTP request initiated through an action on a form. This data is processed as per the business logic and persistence logic behind the scenes by the components of other layers. Depending upon the control flow, the model may directly update the view, or the controller may transfer control to the view after rendering the response page.

Xp

In Struts framework, while the controller servlet can be extended to override its behavior, commonly this is not required, as its default behavior is adequate for most of the presentation layer implementations.

The design team using the Struts framework has to focus usually on developing view and model components of the presentation layer. Let us explore how the entire sequence of activities happens in the presentation layer with the help of the `InitiateLoan` example.

As depicted in the sequence diagram (Figure 3-15), the sequence of actions starts when the actor `Bank Representative` submits the data on `InitiationForm.jsp` through an HTTP request, to save loan details, to the `ActionServlet` (the controller in Struts framework). The `ActionServlet` has to decide the appropriate action to process this input data. `SaveInitiateLoanDetailsAction` is the class which then coordinates the entire sequence of events in the presentation layer. The design approach here follows the front controller pattern. In general, the controller servlet acts as the common point for handling all requests by delegating control to the respective (action) classes for further business processing, and to manage the navigation to appropriate views based on the outcome of the processing. In this particular instance, the `SaveInitiateLoanDetailsAction` to which the request is delegated instantiates the `InitiateLoanForm` bean and updates it with the data received in the request, and invoke the appropriate business delegate client. The subsequent processing that happens beyond the `BusinessDelegateClient` typically comprises of actual business processing

and data access. As a result of subsequent processing, the `InitiationLoanSuccess.jsp` response page is rendered to the end user.

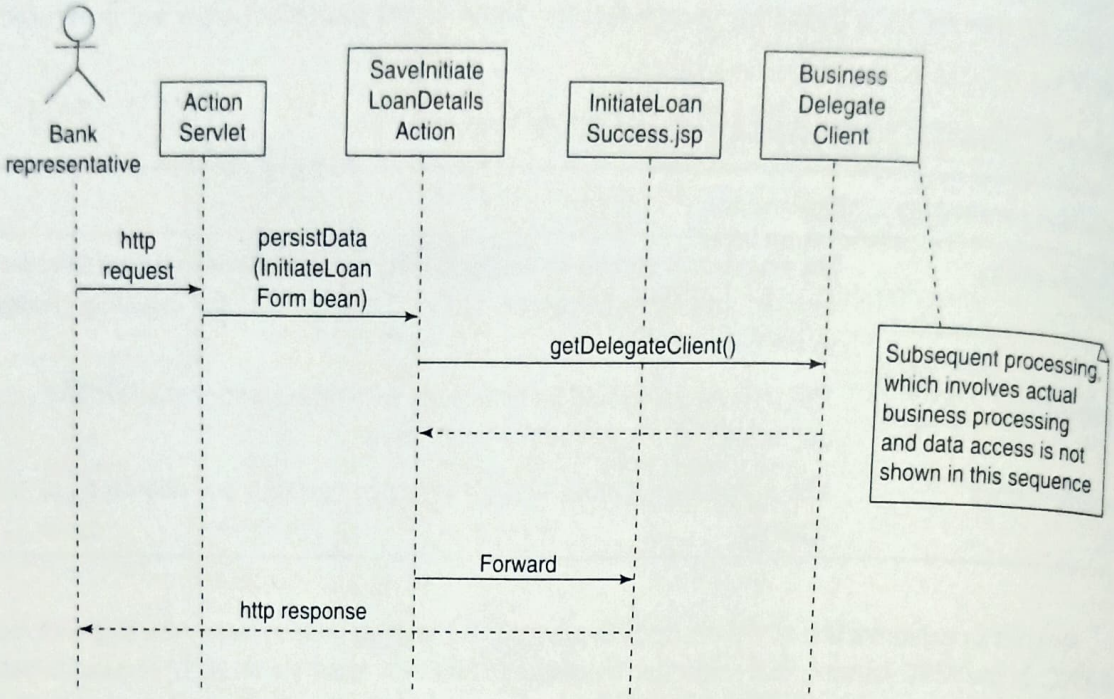


Figure 3-15 Request processing in the presentation layer.

More presentation layer technologies

Apart from the JEE APIs for presentation layer discussed in the previous section, there are several other useful technologies that are usually leveraged for building the presentation layer.

JavaScript is a popular scripting language, which is used across a multitude of Web pages, especially for client side scripting. For example, JavaScript is extensively used for client side interaction such as input data validation, form submission, and menu display and selection.

Asynchronous JavaScript and XML (AJAX) is another promising technology, especially in the arena of Rich Internet Applications (RIA). RIAs are the applications that are more engaging in nature from the end user stand point. There are several uses of AJAX technology—it avoids reloading of the entire Web page as it has the capability to render the Web page partly and supports features such as "auto complete" for implementing a rich UI. This technology creates a desktop feel for the end user whereby updates to the page happen without requiring reload of the page. This is achieved through asynchronous interactions of the page with the server.

Tt Scripting with raw AJAX is tedious and error-prone. There are several frameworks that ease the pain of creating AJAX based views in the presentation layer. DOJO and DWR are a few such frameworks.[7]

[7] For more information on DWR and DOJO, you can refer to http://directwebremoting.org/dwr/ and http://www.dojotoolkit.org/ respectively.

Extensible Stylesheet Language Transformations (XSLT) is used when the presentation layer has to transform the XML data to HTML, or any other format like PDF.[8]

Tt With the advent of Web 2.0 era, new UI technologies are making their way in the software world for designing and implementing the presentation layer, JavaFX™, Adobe Flex and Microsoft® Silverlight™ are examples of such technologies that help implement a desktop look and feel to the UI, commonly referred to as *Rich UI*.

Design aspects of the presentation layer

A good UI enables user productivity, intuitiveness of use, overcoming user disabilities and language barriers. A few aspects that help achieve these objectives and ensure optimum design of the presentation layer are discussed below:

(a) **Usability.** *Usability* of an enterprise application can make or mar the user experience. The design of the presentation layer, from the perspective of usability, is typically dictated by parameters such as proper usage of screen real estate, logical placement of the UI elements, visual appeal, and accessibility features, to name a few. Specialized design techniques exist to address these aspects. Wireframes are a common starting point to design the presentation views and get user feedback on usability requirements.

(b) **Input validation.** *Input validation* is an important design consideration that helps in implementing a robust presentation layer by providing mechanisms to ensure data sanity, data security, and to avoid unexpected errors during processing of the data. Input validation is typically implemented on the client side by using scripting languages like JavaScript. Server side input validation is typically achieved by using application platform and framework services. For example, Struts framework comes with the Validator framework, which supports input validation based on business criteria.

(c) **Internationalization.** It is quite common in an enterprise application to have end users from disparate geographies, which brings in the requirement of enabling the local language support. This requirement is typically realized by using application platform support for internationalization and localization. For example, in a JEE application, these features can be implemented in the presentation layer, either as a single JSP using resource bundles based on locales for different languages, or with different language-specific JSP's. In the latter approach, using servlet filters to pick the right localized page is considered a design best practice.

In Servlet filters intercept the data in the HTTP request or response to make incremental modifications that enable efficient processing or presentation.

(d) **Data transformation.** Data transformation in the presentation layer is different from data processing for business reasons. The purpose here is to transform the processed data in the response to a form suited for presentation. On the same lines, data received as part of a request may be transformed to a form better suited for business processing. For example, XSLT is used extensively to convert data in the XML form to any other required format, especially to HTML or PDF that are more suited for presentation.

[8] For more information on XSL transformations, you can refer to http://www.w3.org/TR/xslt.

(e) **Navigation.** Navigation within the UI, which is part of the interaction model, is an important aspect of the presentation layer. It must be designed with utmost care considering the efficiency and intuitiveness aspects. Navigation can be hard coded in the JSP page but it is considered to be a bad design. Instead, configuration files are commonly used to define navigation, which allows modifying the navigation without requiring changes to the code. For example, in Struts framework (ver. 1.0), the `struts-config.xml` configuration file defines the navigation.

(f) **Session management and cache management.** Session management and cache management are two other important design decisions that have a direct influence on nonfunctional requirements, especially security, performance and availability. For example, not timing out a session after a threshold period of inactivity may lead to a security threat, caching inappropriate data in primary memory may deteriorate the application performance, or improper session replication may be the cause of application unavailability. Design techniques, like appropriate use of cache by requesting minimal amount of data and fewer times may result in improving the performance of the application. Readers have learnt about these two components in the infrastructure services layer section. Their working, along with other presentation layer components, is further discussed in Chapter 4.

(g) **Deployment considerations.** For performance reasons, static and dynamic content of the presentation layer may be partitioned during deployment. In such a case, the Web server is used to host the static content, and the application server is used to host the dynamic content. This minimizes the load on the application server. Refer to Figure 3-14 again that depicts the partition of the presentation layer content. Presentation layer, serving as the gateway to an application, needs to integrate with the security components such as authentication and authorization. The design of the presentation layer should support flexible integration with custom-built or third-party security solutions (like SSO).

(h) **Reusability.** Reuse considerations in designing the presentation layer is motivated by several potential benefits. For example, CSS enables reuse of page style elements to provide a consistent look and feel. Breaking up the page into common parts and variable parts helps in reusing the common parts across all the pages. In addition, there are several other common components of the presentation layer that can be reused like session management functionality, caching management mechanism, internationalization features and generic interaction components.[9]

(i) **Security.** The "attack surface" of enterprise applications is ever increasing and the presentation layer is a major part of it. Attackers often launch attacks on applications and other organizational assets by exploiting weaknesses in the presentation layer design. As learnt before, one of the main reasons behind this is improper input validation, and the design team has to put a robust input validation mechanism in place to counter this.

(j) **Portal integration.** There might be a scenario where the presentation layer needs to span across multiple applications and the requirement is to provide unified user access to these multiple applications. Portal integration is the mechanism that is commonly used to realize such a scenario. It pulls together views of multiple applications in a unified manner to provide a consistent and seamless user experience.

---

[9] You will learn more about the presentation framework and the typical application components that can be reused in presentation layer in Chapter 4.

Consider a business scenario where a user has to interact with multiple applications to complete a single business process. The user can either physically access different applications one after the other to complete the process, or the applications can be integrated in such a manner that the user can access the relevant applications as if in a single window. The former approach relies completely on manual intervention to bridge the applications. The latter approach is ideally suited for portal integration. This approach is depicted in Figure 3-16, where the portal logically unifies the presentation of the multiple backend systems to enable seamless operation spanning across them.

Many service requests from customers require access to a range of customer data. For example, a customer may call to place a new order, check the account balance, check whether a payment has been received or change his or her demographic details. Quite often, this range of data is seldom stored in a single system, but spread across different systems such as order management system, customer relationship management (CRM) system or billing system. Providing a unified view of such a variety of customer data to the customer service representative, through an enterprise portal, is enormously helpful in improving the efficiency and quality of service to the customer. A similar or scaled-down version of the customer representative view could also be presented to enable "self-service" to the customer directly.

In portal integration, it may still be the responsibility of the user to logically sequence the interactions needed to fulfill a business process, since the portal only provides a presentation level integration and not true business process integration.

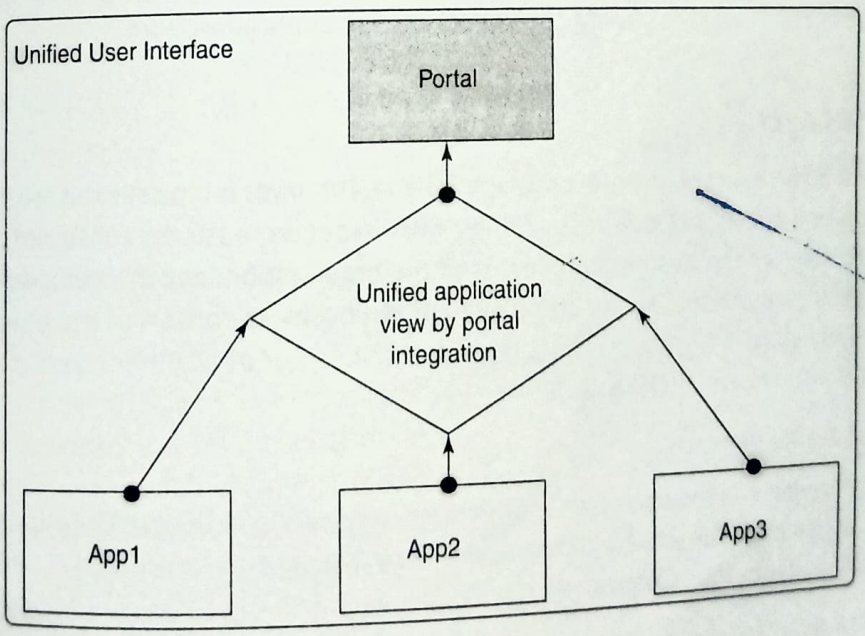


Figure 3-16 Portal integration.

Xp Portals often provide services beyond plain integration such as personalization, SSO, access control and internationalization.

Presentation layer best practices

Some recommended practices in designing the presentation layer of an enterprise application are as follows:

- Keep JSPs lean, as it is not only comparatively difficult to debug JSPs but may also hit the performance.
- Avoid control logic in JSPs.
- Physically, keep JavaScript resources external and embed them logically wherever required by using the include mechanism through the `<%include%>` directive.
- Avoid exposing presentation layer data structures to the business layer to minimize the coupling between them.
- Avoid client-side caching of dynamic pages to avoid issues of stale data.
- Always place shared page elements in a separate file to achieve reuse benefits and consistent look and feel. Typical shared page elements are header, footer, common images, fonts and colors. These elements are merged with the primary content of the page with the help of layout managers.

Tt Layout managers such as the Struts Tiles framework may be used to merge the shared and specific elements on a per page basis.

- Use Java Server Pages Standard Tag Library (JSTL) to handle the internationalization and localization aspects of the enterprise application. Using framework-specific internationalization support, wherever available, is generally a good option.
- Consistent error messages should be used across the application

3.4.5 Business Layer

The business layer is the heart of an enterprise application. This layer is typically the most complex layer of the application and consists of business entities, business rules, business constraints, etc. Typically, the business layer mirrors the business processes of the organization, and the complexity of the business layer is directly proportional to the complexity of the business domain of the enterprise application. As depicted in Figure 3-17, the business layer is at the center of all other layers in a typical *n*-tier application, and is also known as the *middle tier*.

Frameworks and design patterns

The business layer delegates the user events or actions, originating in the presentation layer to its sub components to accomplish the business operations. In a typical enterprise application, the business layer is usually divided into four components:

(a) **Business delegate components.** *Business delegate* components act as a bridge between presentation and business layer by exposing the business processes and services to the presentation layer. The motivation behind using business delegate is to decouple the presentation layer and the business service components, to the maximum extent possible. The business delegate provides encapsulation to the business service component internals such as lookup and access details.

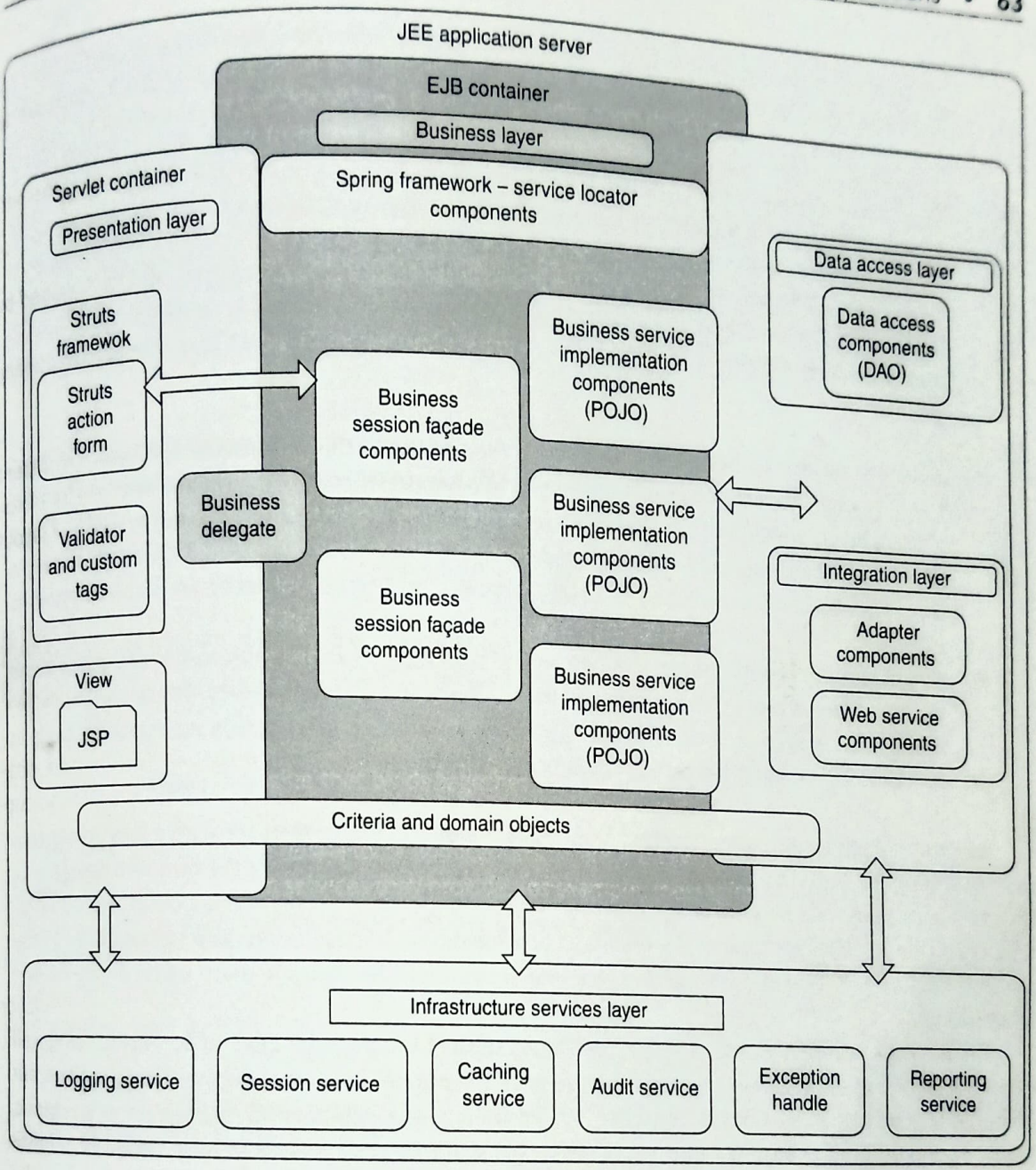


Figure 3-17 Business layer.

(b) **Session façade components.** *Session façade* components play an important role in centralizing the control of underlying fine grained business entities by coordinating them to realize the business functionality of an enterprise. The session façade components help in reducing the network traffic and latency by grouping several fine grained invocations of the business entities. For example, in LoMS, `InitiateLoan` delegate takes care of the coordination, and control the business part of the `initiateloan` use case. The session façade components are typically realized using enterprise session beans, also known as *session EJBs*. The session façade also makes it possible for remote applications to access the business logic via session beans.

Xp

In the description of logical architecture, we have talked about business processes as aggregations of business services which, in turn, are composed from the business components. The technical architecture discussed here could realize such an arrangement through the session facade based on the level of sophistication of its design. A straightforward implementation of the session facade limits itself to the primary objective of optimizing the traffic across presentation and business tiers. A more sophisticated design could achieve composition of processes and services from barebone business components through logical aggregation and orchestration among the components. In most cases, this enhanced role of the session facade is achieved through use of special purpose middleware such as IBM Process Server.

(c) **Business service components**. *Business service* components implement the business functionality of an application. Further, they communicate with the persistence layer or the integration layer, which, in turn, talks to the underlying data stores or external systems. These components are typically implemented using EJBs or POJOs together with other business layer framework components. In LoMS, business service components are implemented as POJOs.

(d) **Business model components**. *Business model* components are the business entity classes that encapsulate the real business data and the methods to manage that data. These can be implemented using EJBs or simple POJOs depending on whether distribution mechanisms such as transaction management or "remoting" are required.

Business delegate components access the business service components through a lookup service, as shown in Figure 3-18. The lookup service is a service locator which connects the business delegate to an instance of a business service, while encapsulating the underlying details of the lookup implementation. The business delegate returns the results of processing by the business service components to the presentation layer. Between business delegate and session façade, typically there exists a one-to-one relationship.

There can be several variations in the implementation of the business layer of an enterprise application. In older implementations, it is common to have substantial amount of business logic in the form of stored procedures as part of the database. While, there was a debate until recently on which was a better option for implementation of the business logic, the consensus seems to have resolved in favor of moving it out of the database. Use of stored procedures should be typically limited to data specific operations rather than business logic.

Several design best practices and patterns are used in the design of the business layer. For example, *service locator* pattern, which is used by the lookup service, is one such design patterns. It decouples the service and its client to avoid a hard binding between them and enables a runtime association between the client and a specific service instance. It also ensures efficiency through sharing, resource pooling and by enabling caching of lookup results. *Business delegate* is the other commonly used design pattern that, in turn, employs the *singleton* pattern to limit the business delegate object to a single instance.[10]

[10] For more information on use of design best practices and patterns in a JEE enterprise application, you may refer to http://java.sun.com/reference/blueprints.

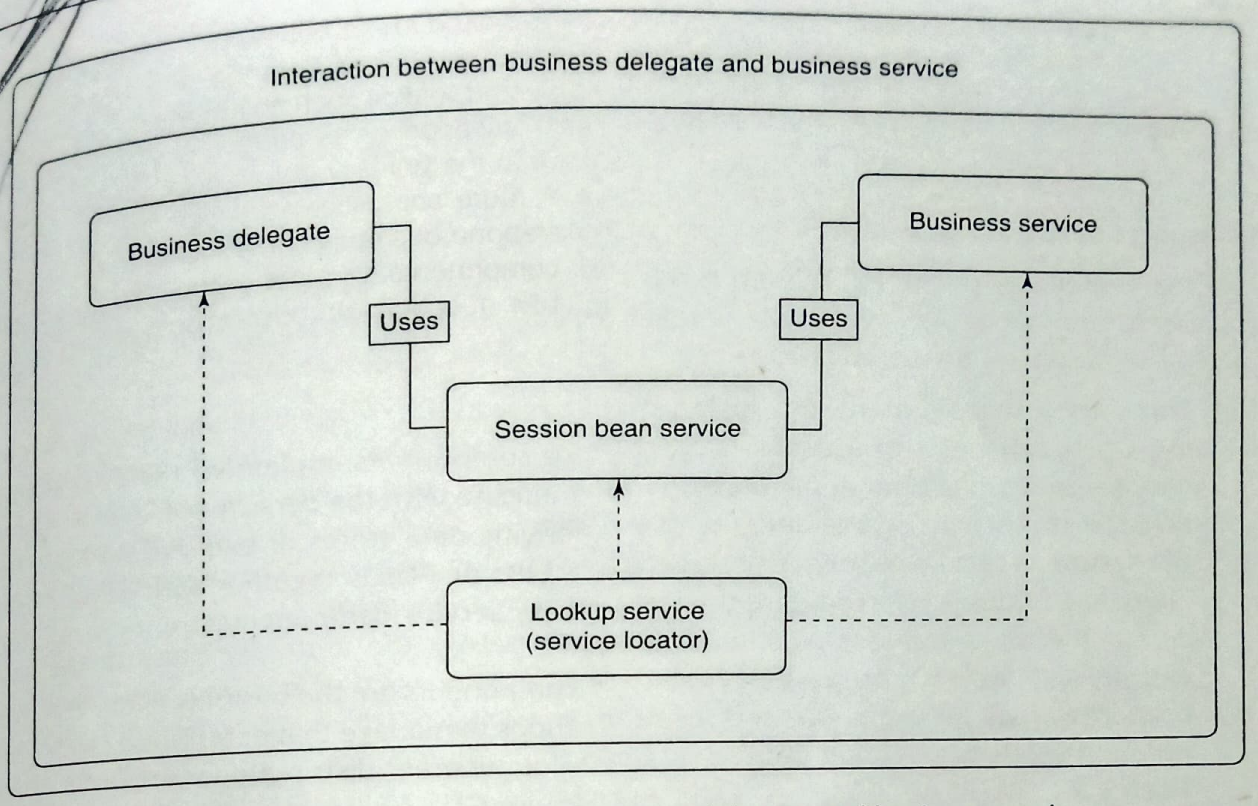


Figure 3-18 Interaction between business delegate and business service.

We now briefly look at how the LoMS business layer components interact with each other. Consider the `initiateloan` use case, as shown in Figure 3-19. The business delegate has a method `getInitiateLoanDelegate` which returns an object of type `IInitiateLoanDelegate` and has a method `saveInitiateLoanDetails`. This method invokes the corresponding service, i.e. `InitiateLoanServiceEJB`. The EJB serves as the session façade in this design and helps avail platform services such as transaction management, connection pooling and life cycle management. The method `saveInitiateLoanDetails` in the service EJB, being only a façade, invokes the actual service implementation, i.e. `InitiateLoanService`. This class, in turn, may talk to the data access layer for further steps of the processing.

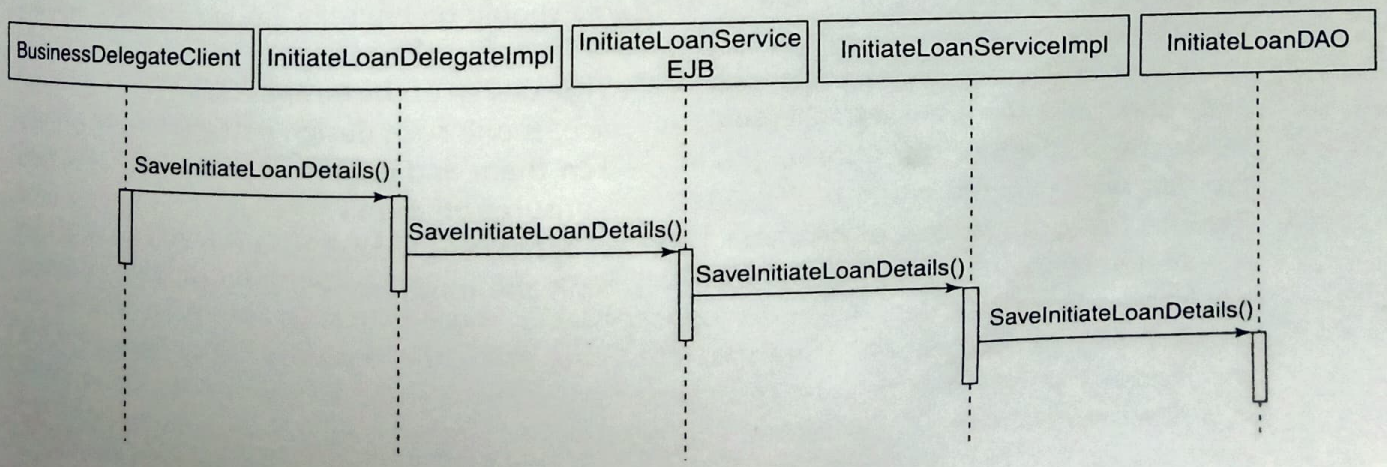


Figure 3-19 Request processing in the business layer.

Tt Besides EJBs, several Java EE frameworks are available for implementing the business layer. Spring is one such framework which supports implementing almost all of the business layer functionality without the need for heavy-lifting of the EJB approach.

Design aspects of the business layer

Most of the complexity of the application resides in the business layer. This necessitates great care in designing it for optimal performance. A few key aspects of the business layer design that require a careful consideration are discussed below:

(a) **Transaction management.** In a distributed and multi-user enterprise application, one of the most critical aspects to consider is *concurrency management*. It refers to managing simultaneous interactions of multiple users with the system, which gives rise to conflicts that could result in compromising the integrity of the state of the system. To ensure the consistency of the system, a certain sequence of activities—a transaction—needs to be handled in an atomic fashion, and this is referred to as the *transaction management*.

In theory, a transaction should rigorously satisfy the atomicity, consistency, isolation and durability (ACID) properties. From a practical perspective, ensuring adherence to ACID properties may be expensive and result in performance issues. To address this, transaction isolation levels are defined depending upon the required level of concurrency. For example, TRANSACTION_READ_COMMITTED is an isolation level where data is visible only after committing the data so that there are no dirty reads that return stale data.

Business transactions are typically different from system level transactions. For example, `InitiateLoan` use case is a business transaction that involves specific stakeholders and a defined process to fulfill the transaction. Fulfilling the business transaction involves certain technology elements like databases or EJBs. These technology components have their own mechanisms for managing transactions, referred to as *system level transactions*. A business transaction may span multiple system level transactions.

Transaction management can be achieved either *declaratively* or *programmatically*. Declarative transaction management is a capability that comes as part of EJBs or an equivalent business layer framework such as Spring. The developer has to just specify the transaction boundaries that may span multiple systems, and the transaction isolation level in a configuration file. The transactions are implemented by the EJB container or the framework without requiring any programming effort by the developer. In programmatic transaction management, the developer codes the transaction management using Java Transaction API (JTA) and establishes the transaction boundaries and isolation levels, possibly across distributed resources involved in the transaction. In programmatic transaction management, the code typically resides in the business delegate.

(b) **Remote (or local) access of business logic.** The mechanism to access business logic is primarily determined by whether the business layer components are physically distributed, or are located in a single server. The former scenario is more typical to enterprise applications and involves remote access of various components of the business logic. Remote access mechanisms suffer from performance penalties, especially if the business logic is accessed in a fine grained manner. Therefore, aggregation of remote calls, where possible, is considered as a good design practice. This is often implemented as a single call to a remote façade,

which is located physically on the server along with the rest of the business logic. The façade decomposes the request based on the distribution of the business services. After processing, the façade aggregates the results, which are returned to the client. This significantly eases the load on the network and the server.

(c) **Use of third-party components.** Usually, several portions of the business layer functionality are independent of the domain of the enterprise application. Rule based processing and workflows are examples of such generic functionality often used in the business layer. One option is to custom develop components for such functionality, but this may involve significant design, coding and testing effort. The alternative which is increasingly adopted is to procure and integrate readily available third-party open source or commercial components implementing this functionality. This approach yields benefits such as reuse and significant reduction in development and testing effort, but may consume some effort in terms of identifying the right components and for their integration.

Rule based processing is typically preferred, when business rules are very complex and change quite often. To provide flexibility and accommodate the changes with minimal impact on the application code, rules engine software is used. Rules engines typically provide generic features such as defining, classifying and managing the business rules, testing the integrity of rules definition, execution of business rules and a set of external APIs to integrate them into an application.

Tt *Drools* is an open-source rules engine, commonly used in JEE applications that involve complex rules processing.[11]

Often applications are developed to cater to one or more business processes. Each business process has its own set of actors and activities, which interact in a predefined fashion. Workflow software is used to automate and streamline the definition of these processes. They typically track the actors and the activities in a process along with the details such as "Who is responsible for the activity?", "When is the activity performed and the state changes in each step of the process?". As discussed before, it is possible to code the logic for a simple workflow in the business layer. However, such an approach could get messy in scenarios involving sophisticated workflows. Such scenarios are best addressed with the use of workflow engines.

(d) **External business function integration.** As discussed in the presentation layer, portals can be used to aggregate the access to multiple applications in a single screen. However, this results in integration that is superficial and limited to presentation views only. But, many business processes require a much deeper level integration of business services, and is referred to as *functional integration*. In practice, several integration technologies can be used to implement this. [12]

Consider a business scenario where a single business service or process stretches beyond the boundaries of an individual enterprise application. Implementing this scenario requires

[11] To know more about Drools, you may refer to http://www.jboss.org/drools/.
[12] You will learn more about integration technologies in the "integration layer" section in this chapter.

creating a layer of software abstraction which orchestrates the services to execute the business process, as shown in Figure 3-20.

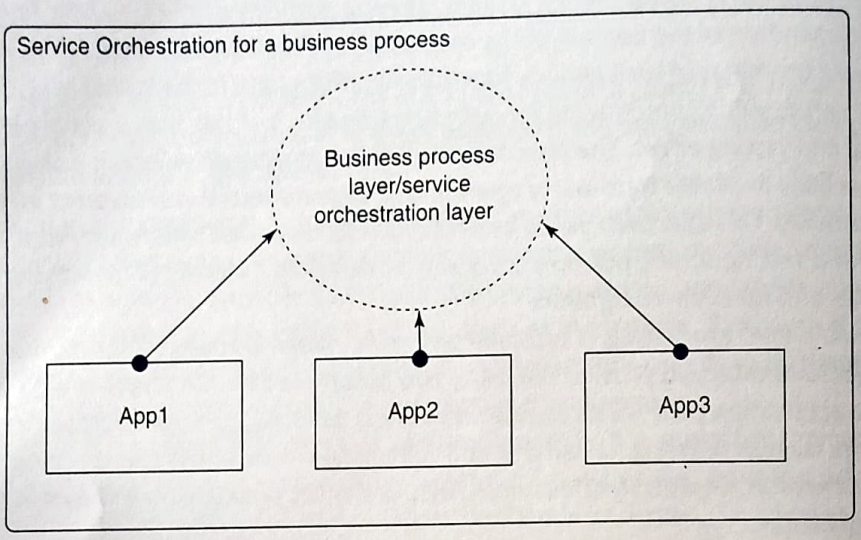
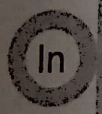


Figure 3-20 Service orchestration for a business process.

For example, in LoMS, loan approval process is composed of several subtasks such as customer validation, credit limit verification, and loan approval. To fulfill the process, several underlying systems such as customer management system, credit rating system and accounting system may need to be integrated. Implementation of this business function in a seamless fashion requires orchestration of the relevant services across the systems.



Functional integration can happen at several levels of granularity. Business services can be integrated together to yield composite services or a process. Similarly, business processes can be composed together to yield a more coarse grained, composite business process.

The business layer depends heavily on the data layer and, in some cases external systems, to realize its functionality. The programming model used in the business layer is often very different from the implementation model of the underlying data sources or external systems. To bridge the differences in the models, we usually introduce layers of abstraction that achieve a logical mapping between them. We will now explore these topics in more detail in the following sections.

3.4.6 Data Layer

Data layer physically stores the enterprise application data. It typically consists of relational databases, but may also have data present in several other data stores such as XML, directory and flat file structures.

Figure 3-21 depicts a typical data layer which has a relational database connected to the business layer through a data access layer. [13]

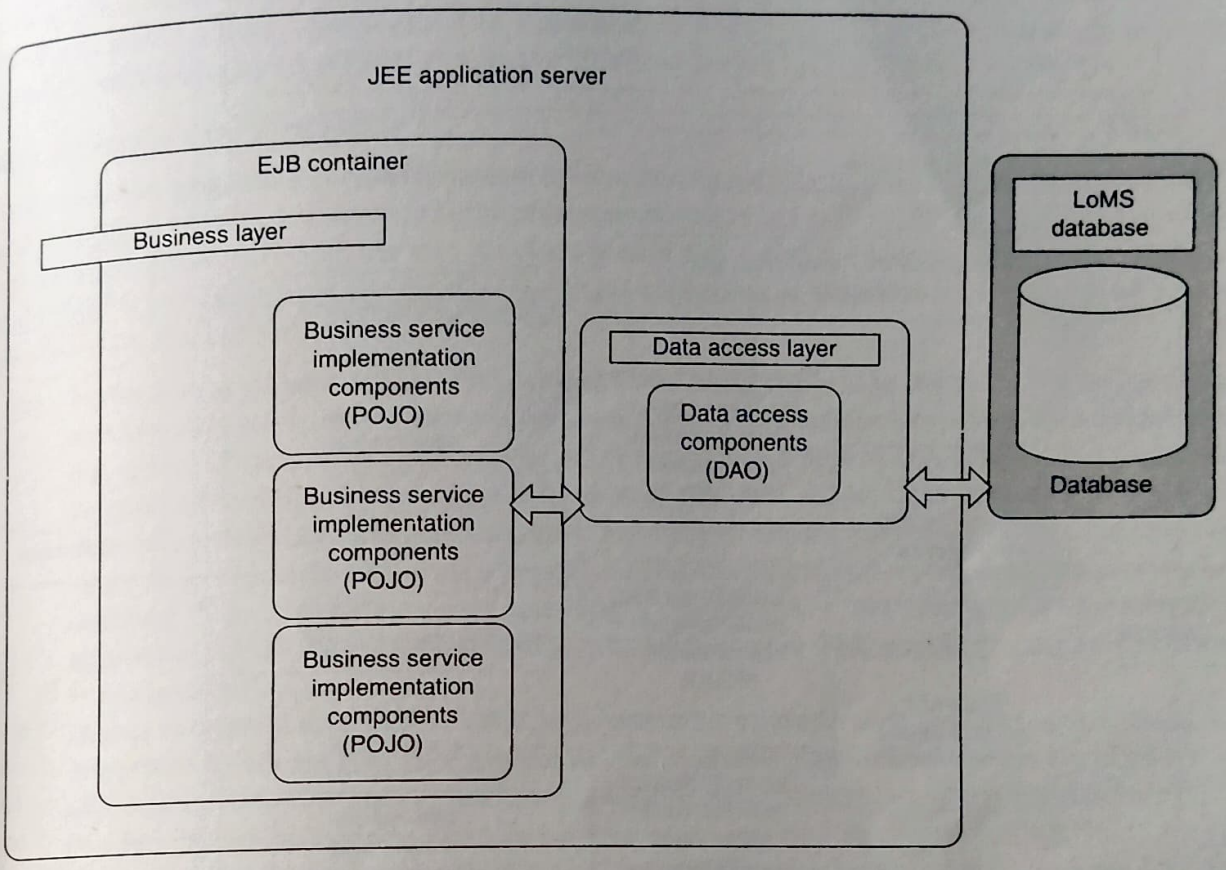


Figure 3-21 Data layer.

3.4.7 Data Access Layer

Data access layer of an enterprise application is responsible for accessing the elements of the underlying data layer which is usually a relational database, on behalf of the business layer. As depicted in diagram 3-22, it acts as an abstraction for the underlying data layer. The sophistication of the data access layer design may vary from one application to another, ranging from simple use of JDBC API to high end Object Relational Mapping (ORM) frameworks such as Hibernate and TopLink.

Frameworks and design patterns

Java applications represent data as objects, and relational databases store data in relational tables. Due to the mismatch between the object and the relational model, there is always a need to bridge these representations. Except in scenarios that involve direct JDBC calls, data access layer implementations typically use some form of meta-data to represent the mapping between objects and relational data.

[13] You will learn more about data layer in the data architecture section in this chapter, especially the types of data stores and their modeling.

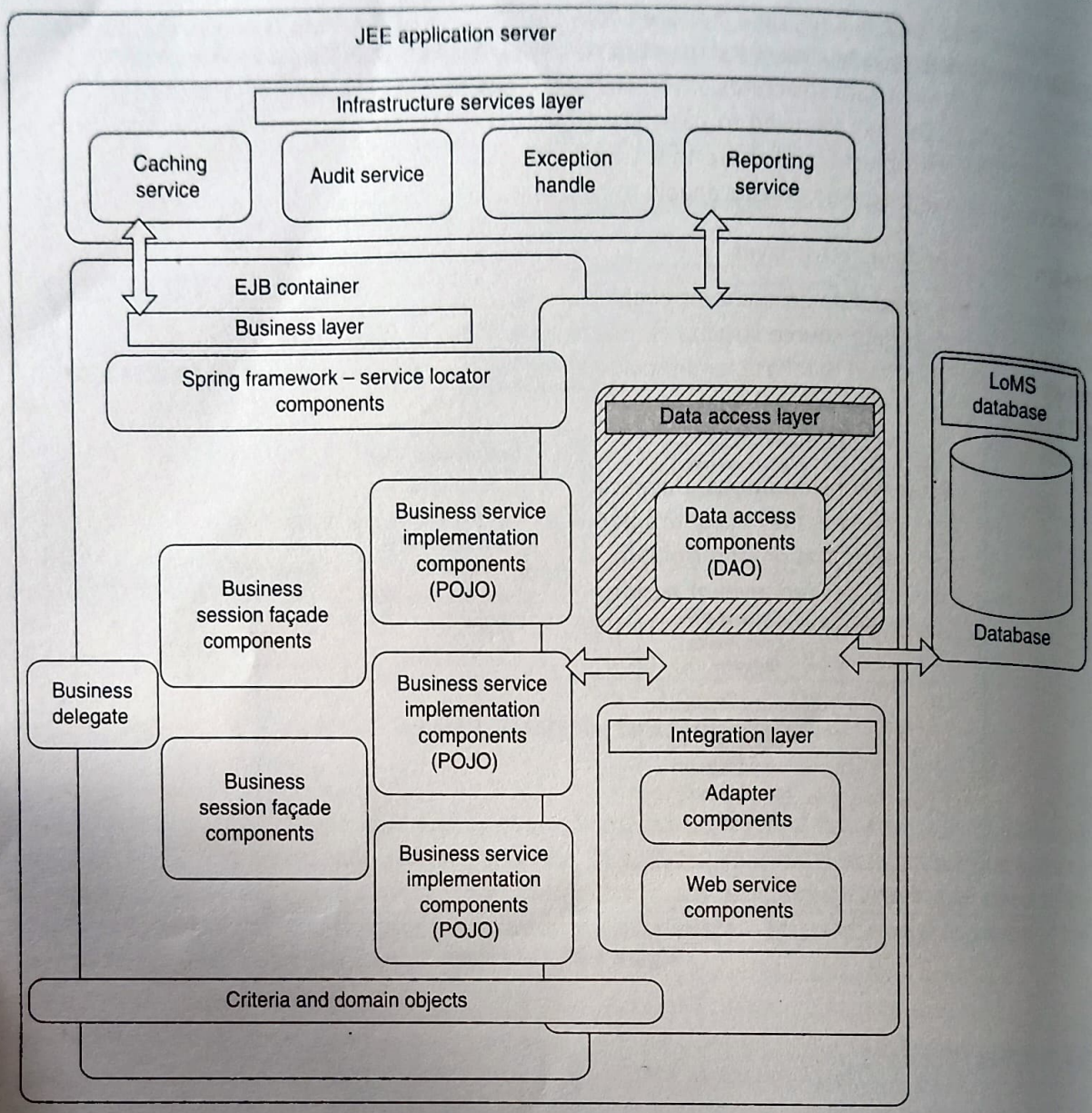
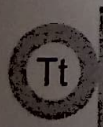


Figure 3-22 Data access layer.



There are several technologies available to design data access layer. Java Data Objects (JDO), entity EJBs until EJB 2.x, Java Persistence API (JPA) and ORM frameworks are some that are widely used.

Data access layer typically uses Data Access Object (DAO) design pattern that encapsulates the details of access mechanism of the underlying data store. It also converts the underlying data formats to an object representation and vice versa. The client of the data access layer, typically the business layer, uses the *Factory* pattern for the lifecycle management tasks, namely creation, pooling and destruction, of the data access object instances.

In a multitier application, data is passed back and forth between the tiers. Data Transfer Objects (DTOs) design pattern is typically used to optimize the data traffic across the tiers and minimize multiple conversions between data structures of the exchanged information.

In LoMS, DTOs that are used to pass data from the presentation layer to the further layers are termed as *criteria* objects, and those that transfer data back from the data access layer as *domain* objects. Criteria and domain objects enable the data transfer across the layers in LoMS.

Design aspects of data access layer

An optimally designed data access layer enables the encapsulation of data access functionality, maps business entities to data source structures, manages connections to data stores, reduces round trips between application and database, and provides secure and scalable interactions with the underlying data source. A few aspects which help achieve these objectives and ensure robust design of the data access layer are discussed below:

(a) **Performance considerations.** Data access layer, being the bridge between the business layer and the data store, may become a bottleneck if it is not designed carefully. Connection pooling, query optimization and isolation level tuning are a few of the widely used techniques to ensure the desired level of performance of the data access layer operations. The size of the connection pool should be managed optimally to ensure ready availability of connections when required while balancing resource utilization. Query optimization techniques such as avoiding '*' in select statements, avoiding computations in the DML statements and using appropriate relational operators and join operations may yield overall performance benefits to the application.

(b) **Object relational mapping.** If ORM tools are used to implement the data access layer, it is important to ensure that they are configured and tuned for efficient mapping between the object and relational models. Framework-specific best practices are often documented by the vendors or available online from respective Web sites and developer forums. For example, Hibernate specific resources can be found at: http://www.hibernate.org/.

(c) **Handling XML data.** XML is very different from relational data and requires special tools/techniques for its manipulation. *XML access object* (XAO) is a special case of DAO, which is specifically designed to map business layer objects to underlying XML data. XAO implementations are realized through the use of XML and XSLT processing libraries.

Tt Java Architecture for XML Binding (JAXB) can be used to create XML access objects or XAOs.

(d) **Security considerations.** The data access layer design should specifically focus on safeguarding the underlying data sources from various security threats. Design considerations, such as avoiding concatenation of strings for preparing a query and validating user input data in dynamic queries, help in minimizing SQL injection attacks.

(e) **Integration of distributed data.** Readers have been earlier introduced to integration scenarios that involve portals and functional integration. However, there may be scenarios that require use of business entities whose attributes may have to be drawn from remote data sources, to fulfill a business need. This requires integration at data level and is commonly referred to as *entity integration*.

Consider a business scenario where an enterprise application needs to access multiple data stores to implement a business function, which requires a unified view of data. This can be achieved by eliminating potential data inconsistencies and creating a common vocabulary of business entities. Integrating various data stores in such a manner to create a comprehensive view of underlying distributed data is depicted in Figure 3-23.

In the enterprise applications landscape, typically there are multiple applications that store the same logical entity, although with variations based on the business processes and rules defined for that entity. For example, in the banking landscape, a customer entity is defined in a customer information management system, account relationship system and payment processing system, among many others. Each enterprise application defines its own view of the customer entity based on the requirements of the processing logic. At times, there may be a requirement to create a unified data view of the customer entity by integrating all of the underlying enterprise applications' data sources, as in the case of a reporting application.

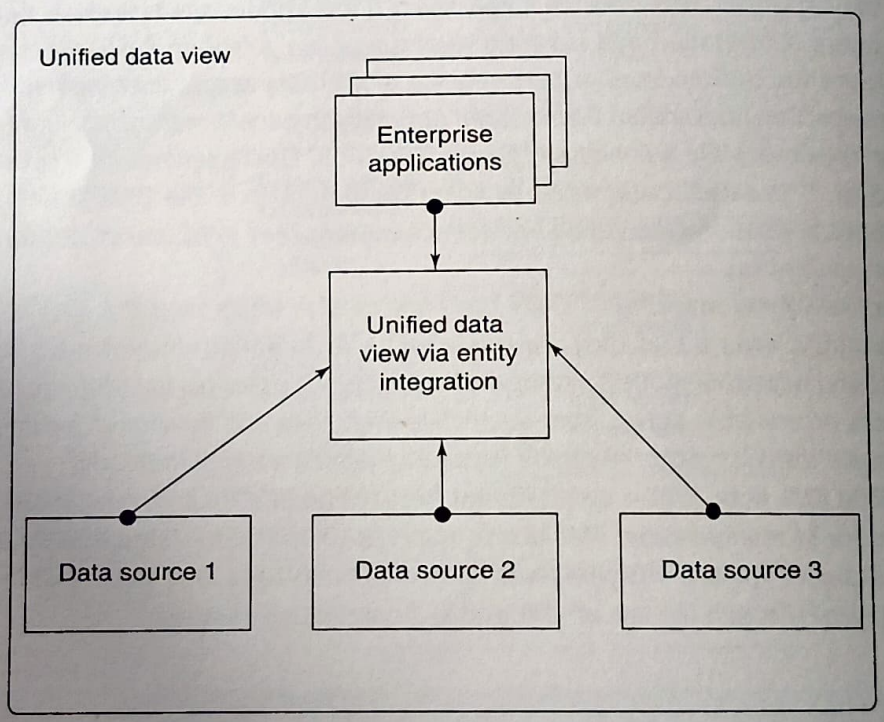


Figure 3-23 Entity integration.

Data access layer best practices

A few of the best practices that are commonly followed while designing the data access layer of an enterprise application are described below:

- In dynamic queries that include user input data, ensure that it is always validated to minimize the possibility of an SQL injection attack.
- Avoid idle database connections for long periods.
- Co-locate the data access layer with the business layer to improve performance.
- Consider caching of frequently accessed data in the data access layer.

3.4.8 External Systems Layer

External systems layer represents the collection of other applications with which the enterprise application under consideration has to interact with. These applications may be hosted internally within the organization, or may be external ones. Usually these applications may be different from each other in multiple ways. They may be legacy systems or modern applications. Modern applications may be based on either JEE or .NET, the two most widely used enterprise application platforms. The applications may follow synchronous or asynchronous, or both, models of communication. They might store data in relational or non-relational formats. Many more such variations are possible.

To design the integration architecture of an enterprise application, the first step is to identify the interfaces through which the applications interact. This activity typically starts in the requirements elicitation phase when business analysts identify the actors interacting with the application. The actors identified as "external systems" are considered as potential integration points.

To illustrate, let us explore the LoMS integration landscape and identify various integration interfaces. Some of the applications are hosted within EM Bank, and the rest are outside the organization's boundary, as shown in Figure 3-24. A partial list of the internally hosted systems is as follows:

- Core product system (Loan) for getting the loan product details
- Customer management system (CMS) for getting the customer demographic details
- Customer relationship system (RL) for customer account relationship details

A partial list of the externally hosted systems is as follows:

- Credit bureau system for getting customer credit rating details
- Insurance system for applying insurance premiums for the loan products

Technology choices for integration with external systems are often made on a case-to-case basis. For example, to connect to a legacy application, a JEE enterprise application may use Java Connector Architecture (JCA) API. For cross-technology integration, Web services might be a good choice. On the other hand, for integration within the JEE platform, there exist native solutions such as RMI-IIOP. We will explore a few of these technologies in the next section.

3.4.9 Integration Layer

Integration layer provides the capability to connect to the external systems through their exposed interfaces. As shown in Figure 3-25, it acts as an abstraction to the underlying systems. An integration layer typically consists of components that enable the enterprise application to either expose a certain capability of the application to the outside world, or similarly to consume the capabilities of an external system to implement some of its functionality. These components may be implemented using various integration types, techniques and technologies based upon the business and technical requirements.

As discussed earlier, enterprise applications implement business processes that span the boundaries of an individual application. This mandates that an architect does not limit his view to a single enterprise application, but considers the whole collection of applications across which the business processes could span to arrive at the integration architecture. Integration architecture gets translated

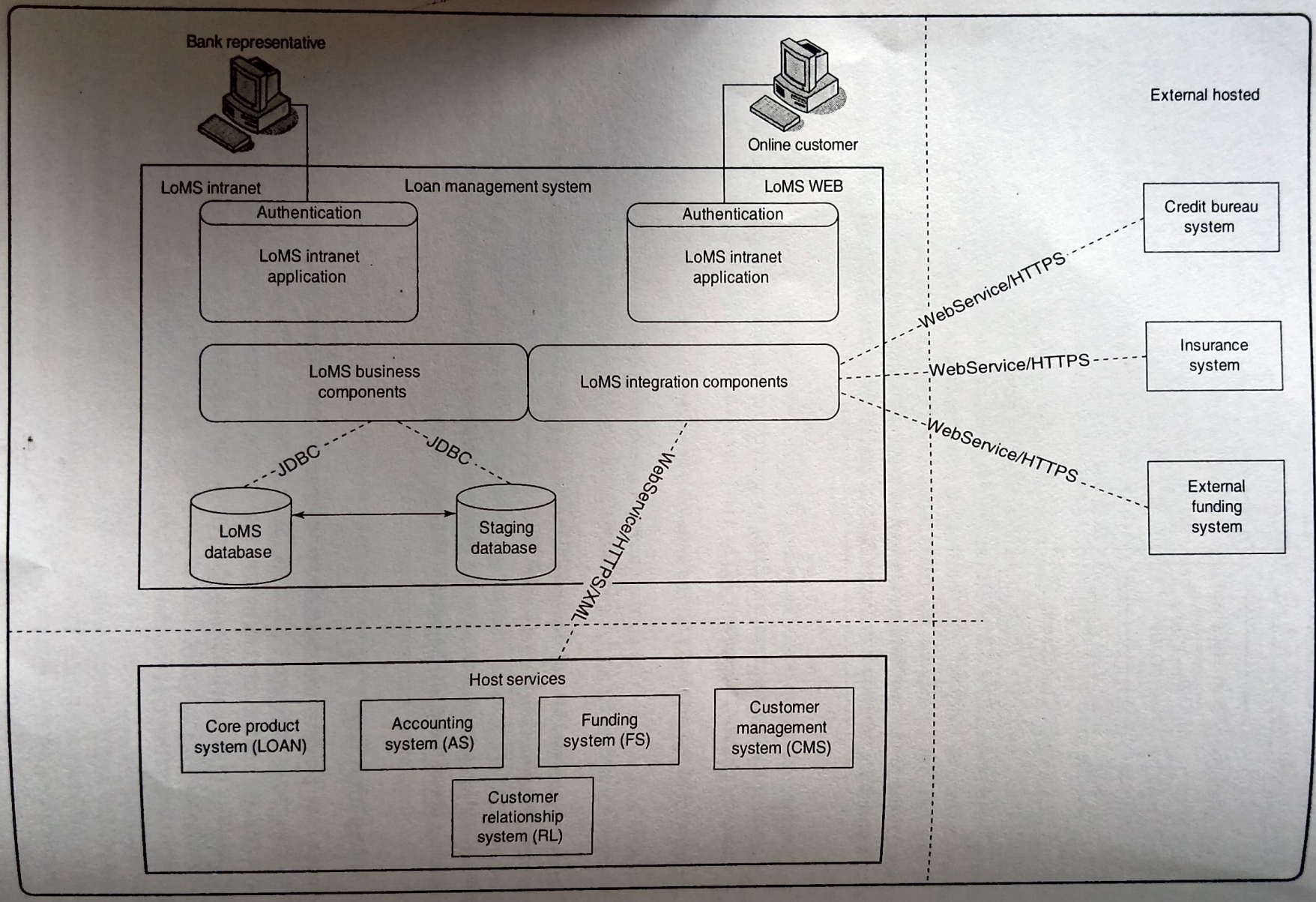


Figure 3-24 LoMS integration landscape.

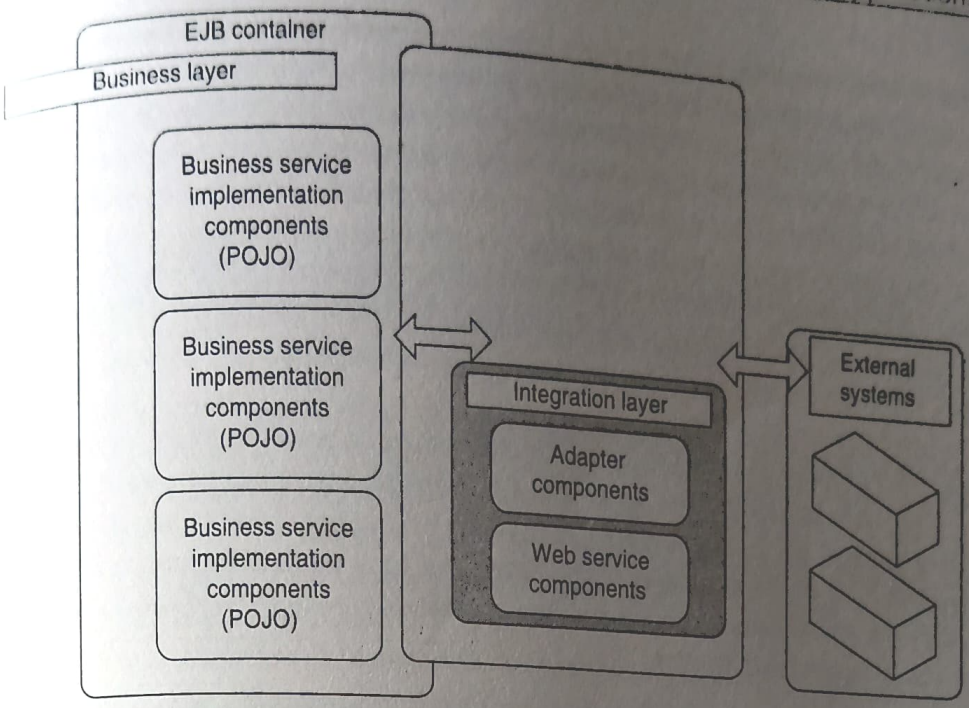


Figure 3-25 Integration layer.

into the implementation of the integration layer. The generic approach to arrive at the implementation of the integration layer is outlined as follows:

- External interfaces with which the enterprise application has to communicate and integrate are identified, as discussed in the previous section.
- Integration types are identified and designed for each external interface as dictated by the integration architecture.
- The design team performs various detailed design activities based upon the technology selected to achieve the requisite functionality of the integration layer.

Integration types

Integration of applications can happen at various levels of the technical architecture stack. These levels of integration are termed as *integration types*.

Xp You will note that an integration layer is a logical entity that can be physically distributed across the layers of the technical architecture stack based upon the chosen integration type.

The three ways in which enterprise applications may integrate with other enterprise applications can be categorized as follows:

1. **UI integration.** UI integration is typically used where some part of the UI or views of an external application has to be part of an enterprise application. This type of integration can be realized through primitive techniques like screen-scraping where the UI stream (usually HTML) is captured, parsed and composed to display the relevant UI elements. The other more sophisticated approach is to use portal integration, as discussed earlier in the presentation layer section.

2. **Functional integration.** Functional integration is used where enterprise applications have to directly access the business logic hosted in the business layer of another application. Business logic might be exposed as a business service or as a business process. Functional integration at a business process level is considered to be at a coarser level in comparison to the functional integration at a business service level, as discussed earlier in the business layer section. Some of the several ways to realize functional integration are as follows:
   (a) Distributed programming paradigms, like Java Remote Method Invocation (RMI), which are used for synchronous interactions.
   (b) Message oriented middleware (MoM), like Java Message Service (JMS), are used for asynchronous interactions.
   (c) Web services like mechanisms, which are based on open standards such as HTTP and XML, are used to connect the systems in a platform and technology independent manner.
3. **Data integration.** Data integration is used where the data layer of an external application is logically part of the data layer of the enterprise application. Data integration can be realized by sharing the database among multiple applications. We discussed data integration earlier in the data access layer section.

In

Enterprise Mashup represents a new generation of integration technology that integrates the information content from independent Web applications. It uses native Web technologies like RSS, JSON and KML. Yahoo!® Pipes™ is one of the tools to create Mashup applications by aggregating, manipulating and mashing up the content around the WWW.

Evolution of integration technologies

The integration technologies have evolved over a period of time to reflect the growing complexity of the integration task. In older Enterprise Application Integration (EAI) solutions, integration mechanisms are largely focused on the application boundaries with intent to somehow marry the applications. As the need for integration became more extensive, a more streamlined and sophisticated approach to integration has emerged. For example, the JEE platform comes bundled with the JMS, and JCA APIs which are specifically for the purpose of integration. In order to cater to the growing heterogeneity of technologies used for implementing enterprise applications, Web services came into existence. Enterprise Service Bus represents the next generation of integration technology with a service orientation in addition to support for Web services. Let us now further explore a few of these integration technologies.

(a) **Third-party EAI solutions.** Before the emergence of Web services, vendor-specific or home-grown solutions were almost exclusively used to achieve integration of applications that had to work together to implement a business solution. Typically, these vendor-specific solutions support multiple kinds of synchronous and asynchronous communication—request/reply, queues and publish/subscribe. Solutions built with them need less development effort and hence foster lower time to market. But, these solutions are proprietary and often come at a relatively much higher cost.

Tt

Some of the proprietary EAI vendors in this space are TIBCO and WebMethods, They provide indigenous/proprietary ways to realize federation, mediation, message brokering, message bus, APIs, etc., to implement an EAI solution.

Java based application integration. Java based application integration is preferred when one or more of the applications that need to integrate are built using the Java platform. In such a scenario, Java based application integration is the best way to go forward.

Java Message Services (JMS) and Java Connector Architecture (JCA) are the two APIs in JEE suite that support application integration.

JMS API provides the interface to messaging middleware with support for both synchronous and asynchronous communication. It can deliver messages in XML or serialized Java object format. It supports both point-to-point as well as publish/subscribe models of messaging. Point-to-point messaging is used where only two systems are involved—sender and receiver. Publish/subscribe messaging is used where one or more receivers subscribe to messages from one or more publishers.

Figure 3-26 illustrates an instance of integration design with JMS. The loan orders placed in LoMS need to be delivered to the funding system for further processing in an asynchronous manner. To realize this scenario, LoMS uses JMS API to deliver the loan orders. In the process, the `loan orders` data is converted from the Java object format to a technology neutral format used by the underlying messaging service provider. The funding system which is listening for these `loan orders` messages, uses either JMS or a native message service API to retrieve the messages from the messaging service provider for further processing. A similar sequence of message passing, but in the reverse direction is initiated after the funding system has processed the `loan orders`.

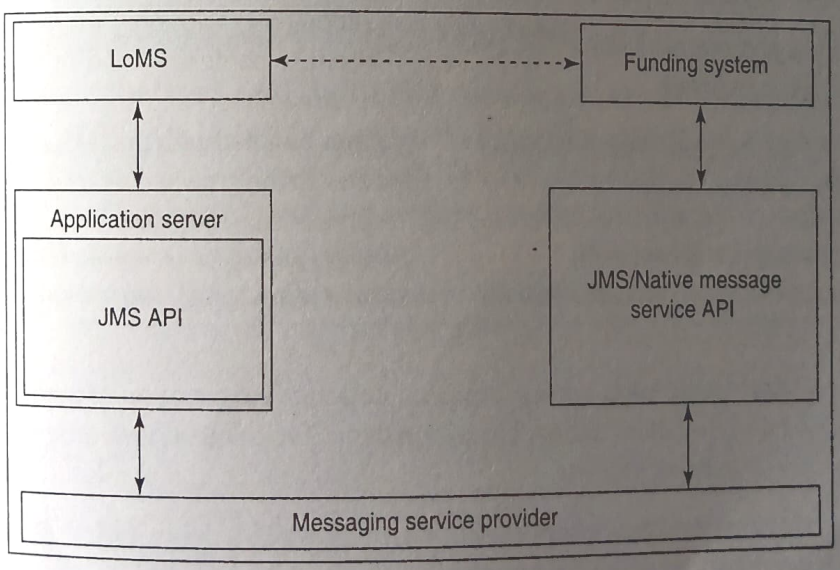


Figure 3-26 JMS based integration.

JCA API is used to provide the integration between JEE applications and a variety of enterprise information systems such as enterprise resource planning (ERP) system, a mainframe customer information control system (CICS) or any other legacy system. Typically, the vendors

provide JCA adapters for their systems to enable their integration with JEE applications in an easy and efficient manner.

(c) **Web services.** Over a period of time since the Internet was launched, the Web has seen a tremendous growth and organizations have realized the potential of the Web in businesses. Traditionally, businesses have offered their services in a human accessible form via the browser. However, as businesses grew they realized the opportunity for much higher growth by exposing these services for direct consumption by other applications. The consolidation of open Web technologies standards made this proposition practical leading to the creation of Web services. This has opened altogether a new arena for business-to-business (B2B) and business-to-consumer (B2C) applications.

W3C defines Web service as "a software system designed to support interoperable machine-to-machine interaction over a network". Web services facilitate distributed computing based on open standards to provide access to business functionality exposed over the network. Consumer applications can access them using the standards based APIs.

There are two types of Web services—SOAP based and REST Web services. SOAP stands for Simple Object Access Protocol, while REST stands for Representational State Transfer. Table 3-2 highlights a few differences between them.

Table 3-2 SOAP vs. REST based Web services

SOAP based Web services	REST based Web services
(a) Used in enterprise applications scenarios where complex stateful and transaction oriented processing is involved	Used in simpler scenarios requiring high performance
(b) Multiple transport protocols like HTTP, SMTP and SNMP are supported	Uses HTTP transport protocol exclusively
(c) Message formats and processing involved are considered heavy weight	Considered as relatively light weight
(d) Supporting standards for security, messaging, transactions, etc., are available	Uses technology or language specific features for security, messaging, transactions, etc.

The SOAP based Web service protocol suite is a collection of technologies/standards/protocols that are used to define, discover, bind and implement Web services. It contains the following four building blocks:

1. **Service description.** It provides the description of the public interfaces for invocation of Web services by using Web Services Description Language (WSDL) standard.
2. **Service messaging framework.** It provides a common messaging framework for service providers and consumers, by using a common XML format to encode the messages. This is typically realized using standards like XML-Remote Procedure Call (XML-RPC) and Simple Object Access Protocol (SOAP).

3. **Service transport.** It specifies the common transport mechanisms among interfacing enterprise applications. This is typically realized using standards like HTTP or SMTP. Unlike REST based Web services, SOAP based Web services are agnostic to transport protocols.
4. **Service discovery.** It is similar to the Yellow Pages service and provides a mechanism to publish and find Web services. Universal Description Discovery and Integration (UDDI) is the standard protocol used for service discovery.

Web services are based on a client server model, where the client and server interact with each other typically by using HTTP, XML and other W3C standards like SOAP and WSDL. In a Web service, typically XML messages, packaged as per the SOAP specification, are exchanged between clients (consumers) and servers (providers). WSDL is a list of operations published by the service provider in an XML format. Web services work in the "publish-find-bind" manner wherein the service provider publishes the services in a common UDDI repository. The consumer finds the service as per the need, and finally binds with the provider to consume the service. The consumer uses the list of operations provided in WSDL to interact with the provider. These interactions (request/response) among client and server happen, using SOAP messages.

In a typical integration scenario using Web services, a Java enterprise application can be integrated with another application as explained below:

1. **Using POJOs**. Java API for XML based Web Services (JAX-WS) API is used to expose interface of a POJO as a Web service. JAX-WS is the successor of Java API for XML based RPC (JAX-RPC), which is an older API to implement the Web services. Both the APIs are supported in JEE 5.0.
2. **Using EJBs**. Tools provided as part of the application server can be used to create a Web service endpoint to expose EJBs as a Web service.

On the other hand, REST based Web services consider all Web elements as resources, and uses HTTP verbs to perform operations like create, delete and modify these resources. All these operations are modeled using HTTP's GET, PUT, POST and DELETE methods. Java API for XML (JAX-RS) based RESTful Web services provides full support for implementing RESTful Web services.

(d) **Enterprise service bus—service-oriented architecture.** The early enterprise applications were typically integrated in a tightly coupled fashion. The phenomenal growth of the Internet has provided an opportunity, as well as a need, for businesses to come together. This has necessitated architecting applications in a loosely coupled manner to minimize the impact of changes to any of the applications. This has led to the inception of Service Oriented Architecture (SOA). SOA deals with services as components from which more complex services or applications can be composed. It enables legacy systems to interoperate with the new generation of Web applications and participate seamlessly in the implementation of business processes.

An enterprise service bus (ESB) provides a mechanism for service-oriented integration through which business services can be exposed and consumed in a protocol and location independent manner, as shown in Figure 3-27.

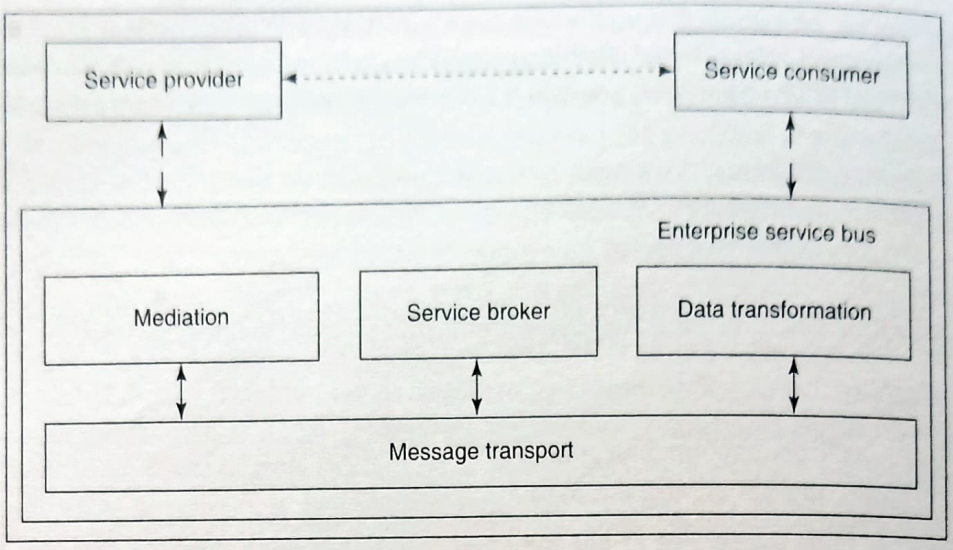


Figure 3-27 Enterprise service bus.

Currently, no universally accepted standard exists for the definition of an ESB. However, Java Business Integration (JBI) is a step in the direction to standardize the specification for an ESB. A typical ESB implementation provides services such as mediation, message routing, service brokering, and data transformation, in addition to providing a message transport mechanism.

Tt BEA Aqualogic Service Bus and IBM Websphere ESB products are a few of the tools used to realize ESB based Service Oriented Architecture. Tools like Apache Service Mix are also available as an open source alternative.

Design aspects of the integration layer

The integration layer, being the gateway to the external systems, should be designed from a holistic perspective that caters to multiple technology and protocol standards. Typically, external interfaces are defined based on consensus among owners of participating systems for streamlined integration. Several key aspects need to be considered to arrive at the design of integration layer such as trigger points for system interactions, communication mode, data format/structure for communication, integration topology and error handling. A few aspects, which help achieve these objectives and ensure reliable design of the integration layer, are discussed below:

(a) **Communication triggers.** The communication between an enterprise application and an external system is driven by external triggers. These triggers are either time based or event based. An example of a time based trigger is a notification reminding the user of the payment due date. An event based trigger originates due to user actions or from external devices.

(b) **Communication methods.** Applications communicate with each other in different ways driven by business requirements. An application may expect an immediate response to a request in which case the communication is referred to as *synchronous*. Alternatively, it may choose not to wait for a response and resume further processing, which is referred to as *asynchronous communication*. In asynchronous communication, the application may be notified when a

response is available, or the application may poll for a response periodically. For example, in a bank, a request for payment transfer is executed synchronously if it is internal (across accounts within the bank), or asynchronously if it is from/to an account with an external bank.

Synchronous communication assumes the availability of the receiving application for communication to succeed. The sender application receives an error notification almost immediately, if the receiving application is not up and running. Synchronous communication is based on the request–reply model and typically uses the remote procedure call (RPC) mechanism. Java provides Remote Method Invocation (RMI) for synchronous communication. Due to the fact that the sender waits for a response before it resumes further processing and the latency of network communication, there may be deterioration in the performance of the application.

In asynchronous communication, application resumes processing irrespective of the state of receiving application and the availability of response. In a JEE application, asynchronous communication is normally achieved using JMS together with messaging infrastructure such as IBM MQ series. A design that uses asynchronous communication is considered to be more fault tolerant than a design using synchronous communication.

(c) **Interface contract.** *Interface contracts* help establish a stable view of services that are exposed by an application and how they can be accessed, while hiding the details of service implementations. This information typically includes various operations exposed by the services, the parameters to invoke the service, and the results returned by the operations. The interface contract may also consist of exceptions that could result during servicing the requests. For example, Web services use WSDL to specify their contracts.

(d) **Error handling.** An enterprise application requires a robust error handling mechanism to ensure that each point of integration is fool-proof. If all potential errors are not well thought of during design, they may prove to be a cause of various problems including exponential maintenance cost. Logging of key information in requests and responses helps quick and efficient diagnosis of problems. Capturing adequate information about the context of errors also helps diagnose the problems easily and efficiently. Recovery mechanisms should be considered and implemented when feasible.

(e) **Integration topologies.** Depending on business needs and technical considerations, various topologies may be used to integrate the applications, commonly referred to as *integration topologies*. Some of the common integration topologies are as follows:

1. **Point-to-point topology.** A point-to-point connection is a direct connection between two systems. The communication between systems is in pairs, and the message and data formats may require conversion if the same formats are not used in both the systems. This topology is feasible only when a small number of systems are involved.
2. **Message broker topology.** *Message broker topology* is a mechanism that decouples the participating systems. All the systems in this topology are connected to a central broker, which takes care of routing and transformation of messages. The message broker topology is also known as a hub-and-spoke topology. There is no limitation on the number of systems that an enterprise application can communicate with, using this topology.
3. **Message bus topology.** *Message bus topology* is a variation of the message broker, where standards based data formats and protocols for communication are used by the participating systems.

3.4.10 Technical Solution Ecosystem

We have discussed the architecture, design patterns, tools, technologies and best practices to design the technical layers of an *n*-tier enterprise application. We now bring all these elements together to provide a holistic view of the complete solution and the flow of requests and responses through them, as shown in Figure 3-28:

1. Client/browser issues an HTTP request to Web server.
2. Struts Action Servlet (controller) receives the request.
3. The controller reads the Struts configuration file to get the action mapping.
4. The relevant action component gets executed.
5. Criteria objects are populated with Action Form values.
6. The action component calls the Business Delegate, which in turn, lookup (6a) the required session façade based on the EJB service ID using Spring Framework Service Locator.
7. Business delegate calls the respective session façade using (7a) Spring Framework Service Locator.
8. The session façade invokes the POJO Business Service using (8a) Spring Framework Service locator.
9. The Business Service calls data access components, which in turn, performs the required database operation using JDBC. It may also call adaptor components or Web services, in order to interact with external systems.
10. Domain objects are populated with the data returned by data access components.
11. The controller invokes Tiles Action Components for the view purpose.
12. Action Form for the view is populated with domain objects.
13. The Tiles Action Components invokes the respective JSP.
14. The JSP uses the Action Forms and custom tags for creating the view.
15. The view is returned as an HTTP response back to the client/browser.

3.5 Data Architecture and Design

Managing and controlling data is one of the herculean tasks for any enterprise. Organizations, across different industries, are focused on standardizing the business entities to achieve harmonization of data to make it more manageable, reusable, interoperable and controllable. In today's world, where IT and businesses are seamlessly intertwined, it is not unusual to say that a business resides in data. Data architecture is one of the domains of enterprise architecture which helps in defining data models for businesses.

In Telecom industry has managed to standardize the telecom industry data using one of the subframeworks of NGOSS—Shared Information and Data (SID).[14]

Enterprise data is present in a myriad formats, ranging from relational databases to data warehouses, from flat files to directory servers, from XML to other prevalent nonrelational data stores.

[14] More information about SID can be found at http://www.tmforum.org/InformationFramework/1684/home.html.

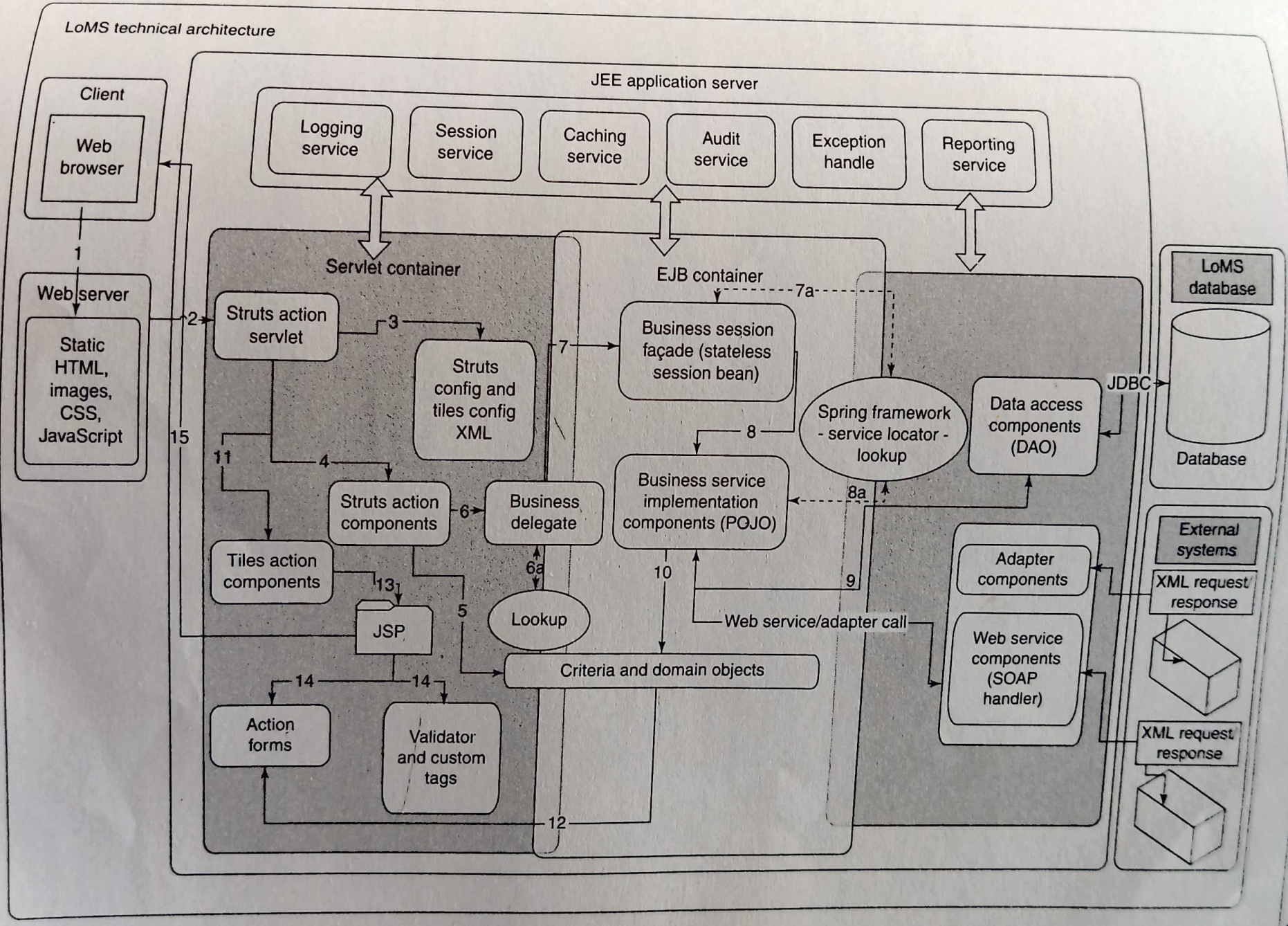


Figure 3-28 Technical solution ecosystem.

Identification of business entities/data, data modeling, life cycle management of data, and data security are a few of the primary building blocks of data architecture. Data architecture also deals with how these building blocks are interrelated and their interactions with each other in an enterprise context.

Data modeling is one of the important building blocks of data architecture. Different kinds of data have specific modeling needs. Let us understand a few of the techniques to model the enterprise data such as relational data modeling, file storage modeling, XML modeling, and other nonrelational data models like directory servers.

3.5.1 Relational Data Modeling

Often enterprise applications store and process the data to fulfill business operations. Significant amounts of data may also be generated in the course of executing the business processes. Usually, relational databases are used to implement the data layer of an application. The design of this layer is realized using a data model which provides the definitions, formats and relationships of the data elements required by the application. The technique of relational data modeling uses a three-step approach to arrive at an optimal design as follows:

(a) **Conceptual modeling**. The first step is to model the real-world business entities, which are in the scope of the application, and the relationships among them. This model is commonly referred to as *conceptual data model*. This activity is typically performed in the requirements engineering phase.

(b) **Logical modeling**. The next step transforms the conceptual data model into a logical implementation, and results in what is commonly referred to as the *logical data model*. It typically consists of relations (tables), attributes and relationships among entities. This activity is typically performed in the application design phase.

Tt

Entity Relationship Diagram (ERD) can be used to represent both a conceptual data model and a logical data model.

(c) **Physical modeling**. The last step is to transform the logical model into a *physical data model*, which involves the physical creation of the data structures such as tables, views and indexes, collectively referred to as *schema*. The creation of physical data model typically happens in the construction phase.

Xp

While there are arguments for and against creating object model prior to data model, or vice versa, in real life, these two models are created in tandem in a parallel and iterative manner.

The business entities, in the scope of an application, can be identified from several sources such as use cases and business requirement specifications. Relational data modeling is performed on similar lines as object modeling, which was discussed earlier in this chapter. Let us now explore a few specific data modeling basics.

Typically, a noun can be a potential entity if it occurs repeatedly and has relevance to the business. Entities are similar to the concept of "class" in object modeling. Customer, loan product and loan account are some examples of entities in the LoMS context. Entities have attributes. Customer name, gender and date of birth are a few attributes of the customer entity. Just as objects are instances of classes, the rows in a relation (table) are entity instances. Every entity should have a primary key, which is an attribute (or a combination of attributes) to uniquely identify the entity instance. For example, in LoMS, `SSN` (social security number) is the primary key of the `Customer` entity. Since there may exist multiple unique keys for an entity, one should ask the question, "Which of the unique keys is usually used to retrieve the entity instances in most of the scenarios", to choose one as the primary key among them.

No entity exists in isolation, and is associated with one or more entities. Such association is commonly referred to as a *relationship*. As in OO modeling, *cardinality* (*multiplicity*) of a relationship can be one-to-one, one-to-many or many-to-many, depending upon the number of entity instances participating in the relationship.

As mentioned earlier, the relationships between entities are depicted using an ERD. An ERD represents the semantics of the underlying business data by depicting the associations among entities in the logical model.

Tt Tools such as Together Architect, Erwin and Visio are typically used to create ERDs.

The ERD shown in Figure 3-29 is the logical data model representing the `Customer` and `LoanDetails` entities. These entities are associated through a one-to-many relationship, indicating that a customer may have many loans, and each loan is associated with one and only one customer.

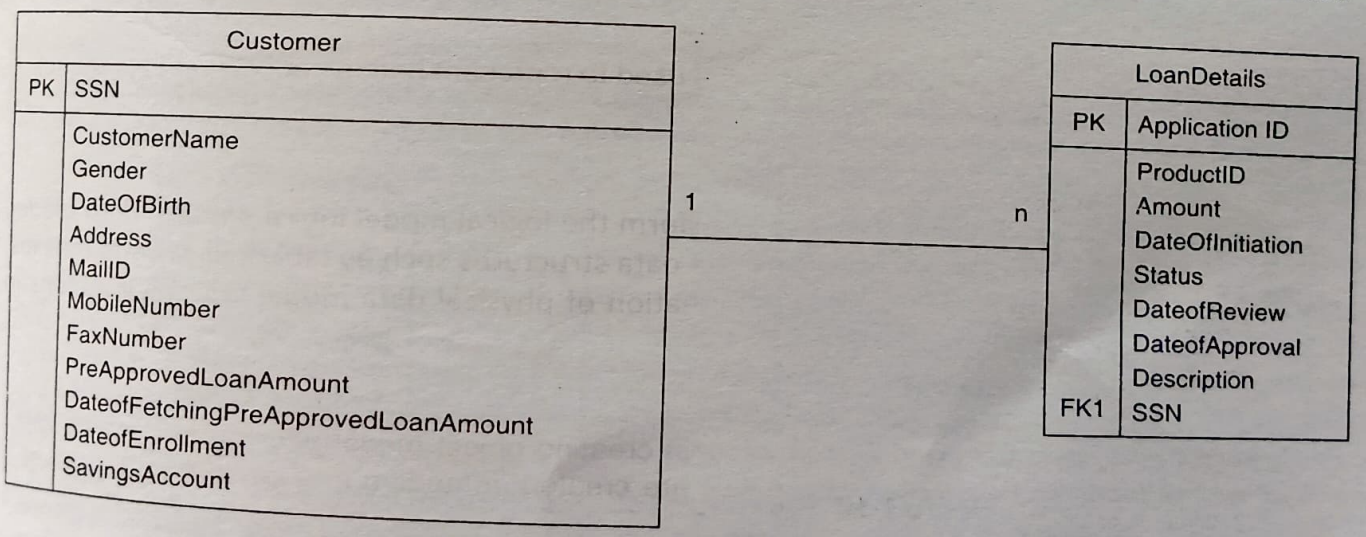


Figure 3-29 Entities and relationships.

A *foreign key* serves as the mechanism to realize a relationship between two entities by providing a reference from one entity to the other. It is implemented as an attribute (or set of attributes) of the referencing entity, whose value is the primary key of the referenced entity in the relationship. For example, as in Figure 3-29, SSN is a foreign key in the `LoanDetails` (referencing) entity which is the

primary key of the `Customer` (referenced) entity. Different variations of relationships that may exist among entities are as follows:

(a) **Recursive relationship.** Relationship among entity instances (rows) of the same table (relation) is commonly referred to as *recursive relationship*. For example, an employee can be a bank representative or a manager. An employee can be a manager of zero or more bank representatives, and every bank representative has one manager. This relationship is realized in the `Employee` table (entity) by declaring `ManagerID` as the foreign key and assigning the `EmployeeID` (primary key of the employee table) as its value.

(b) **Supertype-subtype relationship.** It is easier to understand the *supertype-subtype* relationship through the analogy of generalization relationship used in OO modeling. For example, a `Customer` entity can be either a prospect customer or a loan customer. The relationship between these three entities can be modeled as supertype-subtype relationship, as shown in Figure 3-30. In this example, `SSN` attribute is a primary key for all the three tables. In addition, it also serves as a foreign key to the `Customer` table in the `Prospect` and the `LoanCustomer` tables.

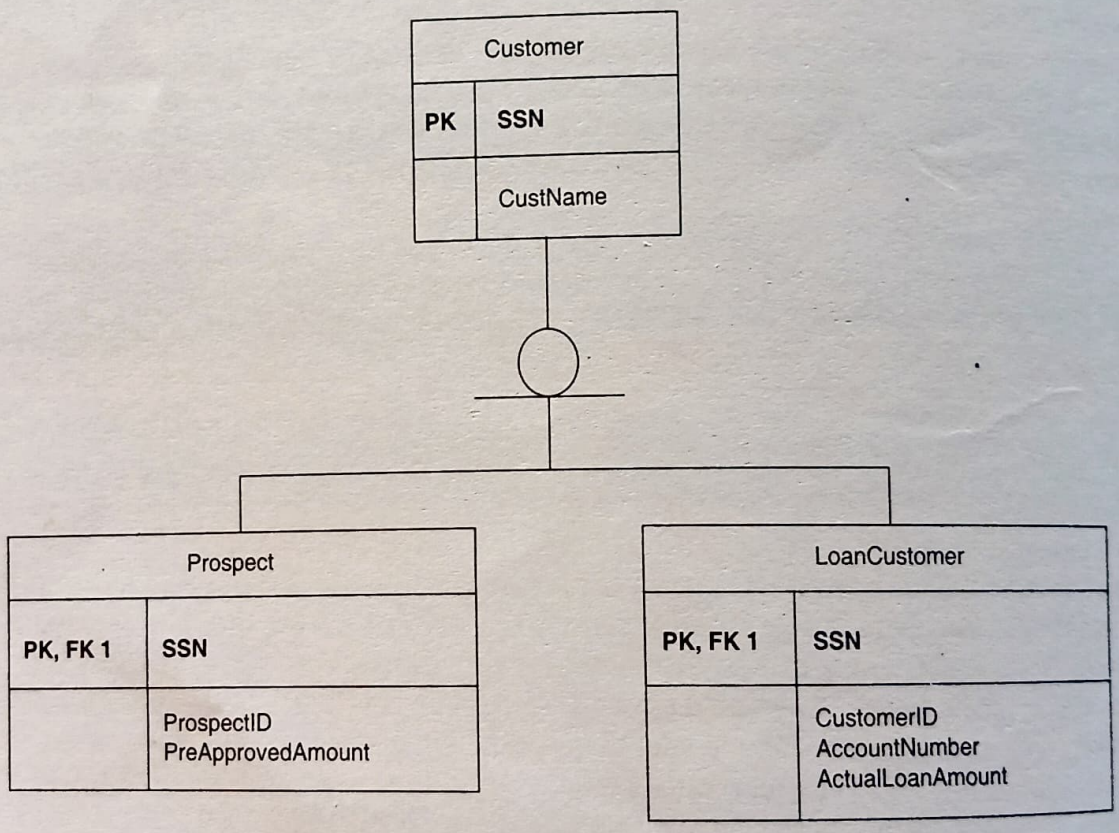


Figure 3-30 Supertype-subtype relationship.

An optimal logical data model can be achieved by following some best practices that include minimizing the data redundancy and usage of atomic attributes. *Data normalization* encapsulates these best practices, and design teams use it to arrive at the desired logical data model. This is a step-by-step procedure to eliminate data redundancy, and results in a series of normal forms, which are successively more compact representations of the data. In practice, the normalization process stops at third normal form (3NF) as a reasonable balance between efficiency of storage and performance.

At times, due to technical considerations such as achieving higher performance, entities may have to be de-normalized. For example, a business reporting application requires data spread across multiple tables, and designing the database to a higher normal form may prove expensive, as it necessitates multiple join operations to fetch the required data.[15]

3.5.2 XML Modeling

XML is an extensible and structured markup language used to describe data in a plain-text format. Use of XML is deep rooted in modern enterprise applications, as it provides a language and technology independent open standards-based mechanism for interactions among heterogeneous platforms. XML is useful for both data and metadata (data about data) representation. You are already familiar with some of the practical uses of XML such as WSDL and SOAP in Web services.

XML represents information in a hierarchical fashion, using a document format. Due to the hierarchical nature, the modeling starts by specifying a root element. While it is not always the case, elements in an XML model can be compared to entities in the relational data model. Likewise, the attributes of XML elements are similar to attributes of entities. Entities (elements) and their relationships in an XML model, besides being dictated by business semantics, imply a hierarchy—parent, child, sibling, ancestor, descendent, etc.—due to the inherent nature of the model. On the other hand, the entity relationships in a relational data model are purely governed by business semantics.

To enforce integrity of the data stored in an XML document, an *XML schema definition* (XSD) is used, which is also an XML document. The XSD specifies a grammar for the data stored in the XML document. Hence, an XML document should conform to the related XSD in order to be considered a valid XML document.

Tt Several tools exist to help manual generation of valid XML documents. XMLSpy is one such tool that supports XML modeling.

XML modeling tools typically have the capability to graphically represent the XML documents and their schema definitions. They also can convert the model between the text and graphical formats.

Developers use XML parser APIs such as Simple API for XML (SAX), Document Object Model (DOM) and Streaming API for XML (StAX) to parse and read/write data from/to the XML documents. A parser reads the text stream from an XML document and generates the hierarchy of data elements. In the process, it also validates the document against an XSD, if one is provided. A SAX parser produces a linear stream of data elements with no representation of the hierarchical relationships among them. It is entirely up to the application logic to infer the relationships based on events generated by the parser. On the other hand, a DOM parser creates a tree representation of all the elements in the document in the memory to allow the application to manipulate the complete model in any way it chooses to. StAX API also works on the principle of stream processing of XML, but unlike SAX, it works in a "pull" mode, where a client program asks parser for the next set of data rather than the parser intimating the client program about it. SAX parsing is appropriate in scenarios, where huge XML documents are being processed and there exists constraint of memory. DOM parsing is appropriate when a representation of the

[15] For a deeper discussion about normalization and normal forms, you can refer to *An Introduction to Database Systems* by C.J. Date.

complete XML document is required in memory for processing. Like SAX, StAX also has the capability to read large documents. However, in StAX, an application is in control rather than the parser, which provides a more flexible alternative to process XML streams.

Tt Java provides the Java API for XML processing (JAXP) and the Java architecture for XML binding (JAXB) to deal with XML documents.

JAXP API is not an XML parser, but it is used to abstract SAX, DOM or StAX parsers. These parser objects can be created and invoked using JAXP. JAXP also supports XSLT transformations. JAXB API provides a simpler way to access and manipulate the XML documents, in comparison to JAXP, when the application requirement is to have an object representation of XML data. JAXP API is useful in those scenarios, where there is a requirement to initiate some processing based on the elements and their values in the XML document, for example, processing configuration files or transform an XML document from one format to another.

Tt Most of the XML data is still stored in file systems. XML can also be stored as a binary object in relational or object-oriented databases.

3.5.3 Other Structured Data Representations

By now, you have understood that data can be stored in structured representations such as relational databases or XML. In addition, there are other representations such as directories and object-oriented databases that are used in special cases.

The data that resides in hierarchical data stores in directory servers is referred to as *directory data*. This data is usually for enterprise wide use, and may include information such as user credentials, network policies, group policies and references to other shared network resources. The primary characteristic of such data is that it is read very often, but changed rarely. There are several commercial and open source directory server products available such as the Sun® Java Directory Server and the Microsoft® Active Directory Server (ADS). Java enterprise applications use JNDI API, which abstracts the Lightweight Directory Access Protocol (LDAP), to access directory data in a vendor neutral manner. LDAP is a protocol, similar to SQL that is used to access data in a relational database in a directory server.

Data may also be stored in *object-oriented databases*, where the data is inherently object oriented in nature. Such databases support key object-oriented features such as objects, classes, polymorphism, inheritance, abstraction and encapsulation. Databases from some vendors support object-oriented data storage. For example, Oracle database uses object types to implement this representation.

3.5.4 Unstructured Data Representations

Many a time, data does not fit into any structured representation. Such data is usually represented as a stream of binary or text data (flat files), and is commonly referred to as documents. The structure of the data in such documents may not follow a standard representation, and has to be known to the

application that processes such data. As such, some types of data may have specialized readers or applications specifically designed for them. Data stored in the flat files, text documents, PDFs, images, sounds and videos are some examples of unstructured data.

Many application scenarios involve use of a combination of structured and unstructured data. To support such scenarios, mechanisms have evolved to embed unstructured representations into structured data, or vice versa. For example, multimedia formats, such as video and audio, can be embedded in an XML document as multipurpose Internet mail extensions (MIME) types. The underlying tools may provide the feature to either directly include the unstructured data or store a reference to it. For example, Oracle provides binary large object (BLOB) type to store binary data, or binary file (BFILE) type to only store the reference to external binary data, in relational tables. Design teams need to appropriately use these features to efficiently manage unstructured data. The choice is based on considerations such as whether the data storage should be part of database transaction and backups, and security access levels.

3.6 Infrastructure Architecture and Design

By now you have learnt how business requirements are transformed into application's technical and data architecture. These are transformed into an implementation which needs infrastructure to get deployed. The infrastructure includes elements such as middleware components used to glue the applications, IT infrastructure hardware and software, and networking and communication protocols. There are several expectations about the quality of service of the application—consistent user experience, enterprise data security, ability to handle vast online user base, 24 × 7 availability and disaster recovery to ensure business continuity—which significantly depends on the ability of the infrastructure to support these requirements. *Infrastructure architecture* ensures that these capabilities are planned for, and built into the system.

Architecting and designing infrastructure is a specialized activity handled by a skilled infrastructure team, and is typically not a direct responsibility of application development team. However, the application development team should possess knowledge of the key building blocks and their influence on the application, in order to carry out the technical design of the solution, and to ensure smooth application deployment.

3.6.1 Infrastructure Architecture Building Blocks

Infrastructure architecture is composed of several elements, as shown in Figure 3-31, which can be grouped into four key building blocks:

(a) Networking, internetworking and communication protocols
(b) IT hardware and software
(c) Middleware
(d) Policies for infrastructure management

These building blocks significantly influence the design and implementation of an enterprise application, especially from the perspective of adherence to nonfunctional requirements such as security, scalability and fault tolerance. The following sections provide an overview of these building blocks.

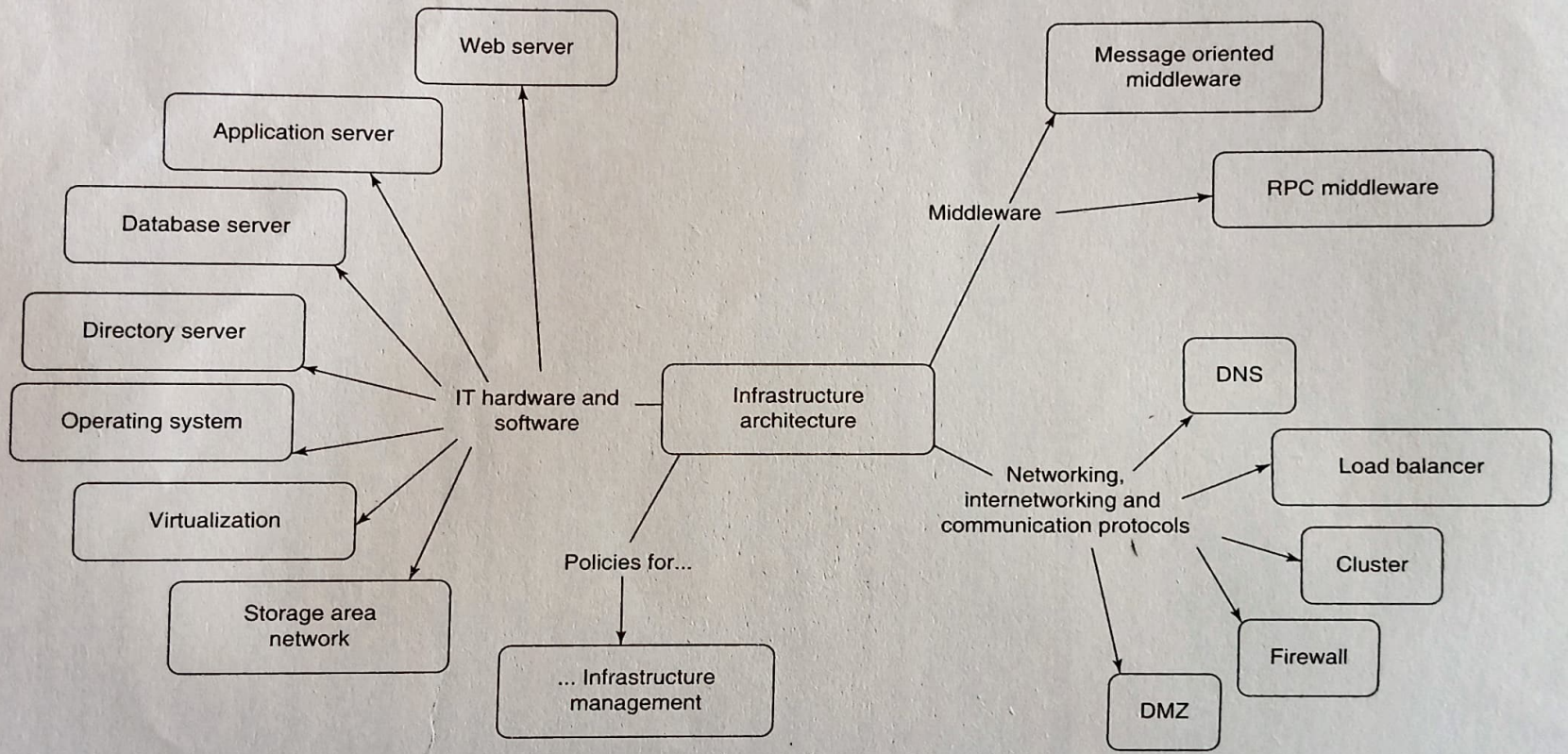


Figure 3-31 Infrastructure architecture building blocks.

3.6.2 Networking, Internetworking and Communication Protocols

An enterprise may have its operations being carried out at multiple business locations. Connectivity between these locations, availability of resources in these locations and secure and reliable access to these resources are of significant importance to an enterprise. Further, each location may have multiple servers hosting different parts of an enterprise application. These servers are accessed by users from variety of devices ranging from desktops to hand-held devices. The connectivity across all the locations and devices is supported by the networking, internetworking and communication protocols building block of the infrastructure architecture.

Let us now understand a few of the key elements of this building block:

DNS

Each resource on the network, such as an application server, Web server, database server or an LDAP server, requires a unique identity to be accessed over a network. This unique identity is provided by an Internet Protocol (IP) address. It has two parts—the network address and the address of individual network resource. For example, in the sample IP address—10.1.1.1—the network address may be 10, and the individual resource address, within this network is 1.1.1. The IP addresses currently used are assigned from a 32-bit IPv4 address space.

In The rapid proliferation of computers connected on the Internet has resulted in IPv4 address space being nearly exhausted. This has necessitated a much larger address space that can provide room for anticipated exponential growth of resources accessible on the Internet. The new IPv6 addressing scheme uses 128 bits, thereby providing a virtually inexhaustible address space that can support about 340 trillion, trillion, trillion addresses.

An Internet, or intranet, enterprise application is accessed via a Uniform Resource Locator (URL), which is mapped to a specific IP address. This mapping is provided by a Domain Name Server (DNS). For example, the URL www.EMBank.com/LoMS (the hypothetical URL of example application) can be used to access the LoMS application of EM Bank. A user request to access LoMS is transparently directed to a DNS server for conversion to an IP address. The returned IP address is used to setup connection with the server hosting the application.

Tt DNS servers are typically hosted as services/processes running in servers such as Windows, Linux and Unix.

Load balancer

Sites that receive high volume of request traffic require dividing the traffic, and diverting the requests to multiple identical servers, to serve the requests in an efficient and scalable manner. A *load balancer,* which is available both as hardware and software, helps in splitting the traffic to ensure the scalability of an enterprise application. It also may have the intelligence to manage application failover, whereby processing is switched from a failed server to a working server. The URL used by the end user is mapped to a load balancer, rather than a specific server hosting the application, which is achieved by the load balancer in a transparent manner.

Cluster

A *cluster* is a mechanism that helps ensure the availability and performance of an enterprise application. It is a collection of servers that appear to be a single virtual server to the user making the request. An application is hosted on each of these servers, and any server can process a request to the application.

In the deployment of an enterprise applicaiton, a load balancer and cluster are used in conjuction to achieve scalability (through division of request traffic) and failover (by diverting requests to a working node of the cluster).

Firewall

Security of an enterprise application is of paramount importance, and is implemented at several levels such as network security, platform security and application security. From an infrastructure perspective, a *firewall* is used to achieve the security at the network level. Policies are defined in firewall to allow/deny traffic to/from particular servers, or based on protocols used. This security mechanism is adopted by organizations to protect resources on their private network, against unauthorized access through Internet. Firewalls are typically configured to allow HTTP/HTTPS traffic to pass through them.

Cisco Pix and Checkpoint are two popular firewalls. Cisco Pix is a hardware firewall, while Checkpoint is a software implementation.

DMZ

Demilitarized Zone (DMZ), also known as *perimeter network*, is a sub network of an organization that provides access to the external facing services of an enterprise to the outside world. Servers providing access to such services are configured to be part of the DMZ, and are placed between an external and internal firewall. This adds an extra level of security to a private corporate network.

In a typical infrastructure setup of an organization, DNS and Web servers are kept in DMZ. Application servers, database servers and other servers are kept behind DMZ.

3.6.3 IT Hardware and Software

IT Hardware and Software is the core building block of infrastructure architecture, which comprises of several elements such as operating systems, servers, storage mechanisms, communication mechanisms and application platforms. These elements play a crucial role in the operation of an enterprise application. Let us now explore a few key elements of this building block.

Operating system

Various types of servers are hosted in an operating system. Basic knowledge of the operating system, which hosts these servers, is important at all stages of the enterprise application development, deployment and operation. The operating system provides a complete set of facilities to manage the hardware and software resources of these servers. These include activities such as monitoring running

applications, starting or stopping the servers, managing configuration files and searching for a particular resource, to name a few. Knowledge of administration wizards, commands, shell scripting, etc., help in efficient performance of these activities and improve productivity.

Database server

A *database server* provides data related services such as efficient storage, search and retrieval, data integrity, security and transaction support. These are important for online transaction processing (OLTP) type of applications to which most enterprise applications belong. Depending upon the design of an enterprise application, a database server may also keep some of the business logic in the form of stored procedures, functions and triggers. As mentioned earlier, relational databases are the most commonly used form of database management systems.

Tt Oracle, Microsoft SQL server and IBM DB2 are some of the most popular database servers. Open source alternatives like MySQL are also available.

Many present-day database servers provide high availability support in the form of clustering, replication, mirroring, etc. For example, Oracle provides the Real Application Cluster (RAC) feature to implement clustering.

Web server

In an *n*-tier enterprise application, the server which accepts HTTP/HTTPS requests from a browser, and services them by interacting with other tiers, is called a *Web server*. It acts as an entry point to access an enterprise application, and typically supports load balancing features in order to improve overall application performance, scalability and reliability. Web servers are usually placed behind a firewall and a hardware load balancer in a typical IT infrastructure setup.

Tt Examples of Web servers are Apache Web server and Microsoft Internet Information Server (IIS).

Application server

The core business logic of an application is hosted on an *application server*. The application server provides several system-level services that simplify the implementation of NFRs. These services typically include object/component life cycle management, concurrency management, transaction management, resource monitoring, application security, load balancing and failover capabilities. An application server also provides a centralized mechanism to administer the resource configurations for an enterprise application such as setting up a message queue, a JDBC data source or a JNDI namespace.

Tt There are several products in the application server space such as Oracle® Weblogic server, IBM Websphere application server, Adobe JRun and Sun Java System Application Server. There are also open source alternatives such as Red Hat JBoss and Sun Glassfish servers.

Directory server

As discussed earlier in the data architecture section, a *directory server* is a kind of a data store. Typically, organization wide data, which is almost read only and is required by various enterprise applications for purposes such as user authentication and shared resources on the network, is stored in directory servers. This includes data such as user credentials, user profiles, network printers and shared network locations.

LDAP is the most commonly used protocol for directory servers. Many commercial application servers come bundled with an LDAP server, which can be used by the enterprise application development team to test LDAP connectivity and configurations at development time.

Tt Examples of LDAP servers include the IBM Tivoli Directory Server and Microsoft Active Directory Service (which comes bundled with Windows Server 2000 onwards). Open source alternatives such as OpenDS and OpenLDAP are also available.

Virtualization

Virtualization, as represented in the block diagram (Figure 3-32), is a mechanism to abstract IT infrastructure. It can be viewed as a design pattern in the IT infrastructure landscape and is implemented at different levels like platform or system resource level. This helps achieve consolidation of platforms and resources through sharing, so that a physical processor or a disk can be assigned transparently on a need basis, rather than being dedicated, to an enterprise application running on such a virtualized infrastructure.

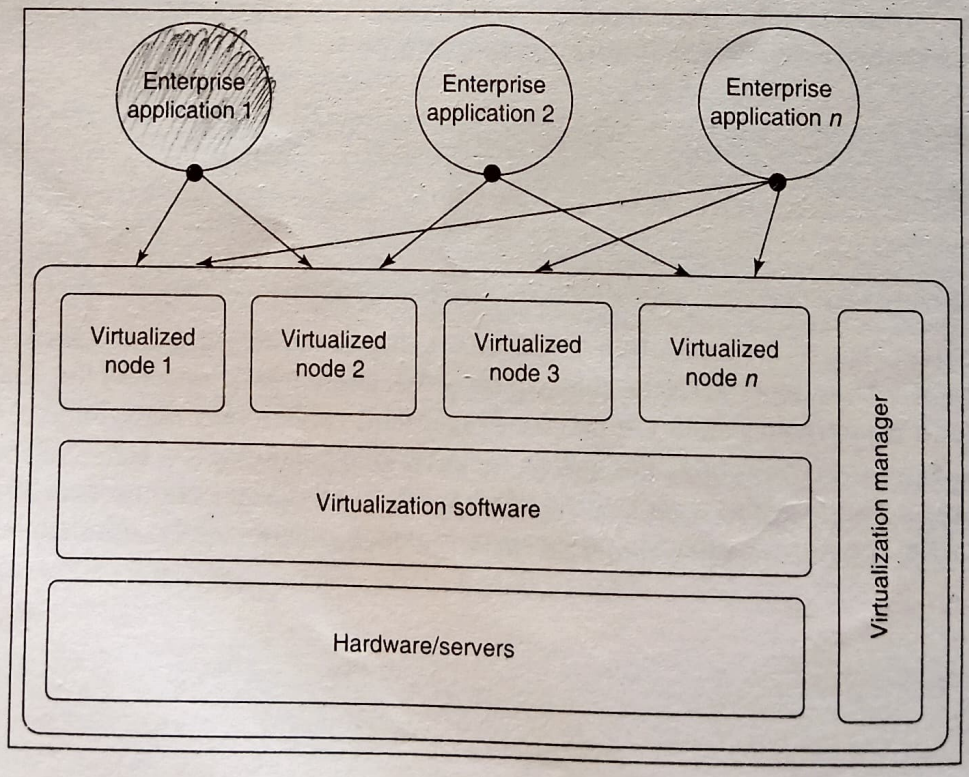


Figure 3-32 Block diagram of virtualization.

Many development organizations use virtualized environments for development and production testing. However, acceptance and production testing are best done in a real environment due to capacity and performance considerations. Enterprise application development and deployment teams need to be aware of this fact from the performance perspective of their applications.

Tt VMware is one of the earliest and most widely used virtualization platforms.

Large enterprises have big IT infrastructure spread across multiple geographic locations in their data centers. Enterprises use virtualization as a means to achieve energy efficiency and better infrastructure management.

Storage area network (SAN)

In order to simplify the administration and maintenance of storage and ensure optimal utilization, dedicated servers are hooked up with storage devices of large capacity and shared across applications. These appear as if they were local storage, and the resulting configuration is called a *storage area network* (SAN). It provides flexibility in terms of administration and dynamically allocating storage without having to physically move the servers.

3.6.4 Middleware

Middleware is the software glue that binds together the software pieces of a distributed application, and enables integration of discrete enterprise applications and their components. It is used to enable interoperability and enhance reliability of applications. There are several types of middleware catering to the various kinds of system services they provide, for example, messaging, transaction management, security, etc.

Tt TIBCO and Web Methods provide messaging middleware. Application servers are also a form of middleware which combine multiple services.

Middleware can be categorized into different types:

MOM

Message Oriented Middleware (MOM) is based on a client/server architecture used for reliable transport of messages across heterogeneous, geographically distributed systems. MOM persists the messages that are in transit to ensure their successful delivery despite any kind of network or system failures. MOM may also provide additional services such as message translation and transformation, and routing the message to the appropriate destination through unicasting, multicasting or broadcasting mechanisms. MOM is typically used for exchange of messages in an asynchronous fashion, which decouples the sender and the receiver.

RPC

Remote Procedure Call (RPC) is also based on the client/server architecture, and provides a mechanism for invocation of procedures on a remote server in a way identical to invocation of local procedures, by completely hiding the details of the underlying network.

Tt Remote Method Invocation (RMI) in Java and .NET Remoting are examples of RPC middleware. COM and CORBA are also older RPC based technologies.

3.6.5 Policies for Infrastructure Management

IT infrastructure of an organization is governed by policies for infrastructure management. These policies are driven by infrastructure management best practices that are codified as industry standard frameworks such as Control Objectives for Information and related Technology (COBIT) and Information Technology Infrastructure Library (ITIL). COBIT provides an IT governance framework and supporting toolsets.[16] ITIL is a set of policies and best practices to manage IT infrastructure, development and operations.[17]

From the limited perspective and scope of this book, we will only briefly touch upon one of the most crucial, and the one that consumes maximum attention and resources in any organization, which is the *data governance policy.*

Data is a crucial asset of any organization required to run its operations. In addition, it may be essential to ensure that the usage, storage, transmission and sharing of data comply with various regulatory and administrative requirements, which may change from time to time. Further, the vast amount of data generated/acquired, stored, processed and moved from one place to another need to be managed in a cost effective fashion, while also ensuring that it is readily available when needed. A data governance policy addresses all of these aspects and more, by defining procedures, guidelines and processes for classification, version management and backup and restoration of data, among others.

There are several other polices that play a vital role in the development, deployment, maintenance and operation of enterprise applications. These include policies for technology acquisition and deployment, security management and user management to name a few.

3.6.6 Deployment Strategy

Deployment strategy is formulated to ensure the fulfillment of several qualities that an application in operation is expected to exhibit such as scalability, performance, security and availability. Decisions such as where to keep application configuration data (in database or in properties files), when to load data from property files, number of application instances required and number of servers on which to deploy them, where to keep the static content of an enterprise application, how to define the security perimeter, and many more need to be addressed as part of a deployment strategy.

Figure 3-33 represents the deployment view of the LoMS enterprise application. This may hold true, in general, for any *n*-tier enterprise application. However, there may exist several variations depending on factors such as business criticality of the application, technology considerations, and whether the application is deployed for access over Internet or intranet only.

The most basic kind of deployment may comprise of all the layers of an enterprise application deployed on a single application server. This kind of deployment is used when the expected load on the application is not very high, and the application is not very business critical.

---

[16] To know more about COBIT, you can refer to http://www.isaca.org/cobit/.

[17] To know more about ITIL, you can refer to http://www.itil-officialsite.com/home/home.asp and http://www.itil.org/en/.

Figure 3-33 represents a multisite and *n*-tier deployment with support for load balancing and failover in all the tiers. To ensure the performance and availability of the application, a hardware load balancer is used in active/passive configuration, which ensures the availability of a backup (passive) in the event of failure of the primary (active) load balancer. Web servers can also incorporate load balancing to route a request to an appropriate application server to service the request. This load balancingcapability is implemented in software, and enabled, using configuration parameters of the Web server. For example, in the Apache Web server, one has to modify the `httpd.conf` file to enable the load balancing feature. The application servers and database servers are also deployed in a clustered configuration to ensure adequate performance and application availability.

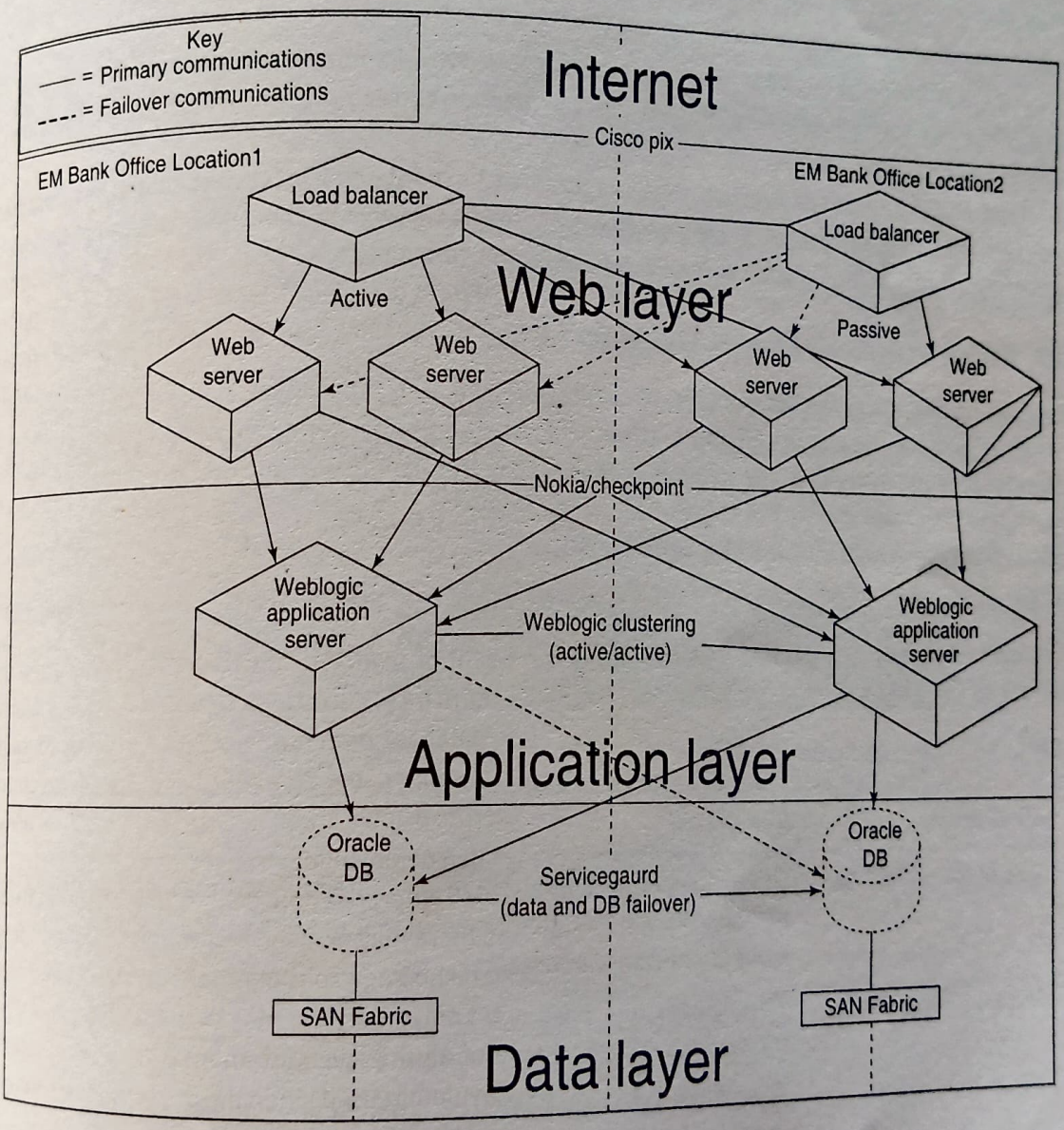


Figure 3-33 Deployment view of LoMS application.

To ensure the security of an enterprise application, firewalls and intrusion detection systems are used. As shown in Figure 3-33, Cisco Pix firewall is facing the Internet, while Checkpoint firewall is facing the application servers. The hardware load balancers and the Web servers are placed in the DMZ

between them. As mentioned earlier, this provides an extra layer of security, in case the first firewall is compromised.

To ensure scalability, the infrastructure team may use *scale up* or *scale out* options. Scale up or scaling vertically means adding more muscle power (resources such as more powerful CPU or more memory) to the same server. Scale out or scaling horizontally means adding more servers to a system.

Software as a service

Software as a Service (SaaS) is an emerging model for deploying applications without users, having to incur big upfront investments in hardware and software. A SaaS provider sets up applications required by users along with the necessary infrastructure, and offers the use of the application as a service. Users access the service based on need, and pay for it based on the level and duration of usage.

Tt Salesforce.com is a popular example of SaaS based application. It is a CRM system that is used across the industry verticals.

From a SaaS provider perspective, the benefits are efficient utilization of resources, and centralized management of the application. There are currently certain issues that are hindering widespread adoption of this model such as quality of service assurance and data security.

3.7 Architecture and Design Documentation

This section introduces the key elements of architecture and design documentation, which records the results of architecture and design phase. These artifacts are essential during the construction phase to carry forward the integrity of the solution as envisaged in the architecture and design phase.

3.7.1 System Architecture Documentation

The architecture of an enterprise application is documented in the *system architecture document* (SAD). The term *high-level design* is also used interchangeably with SAD. It acts as a blueprint for the detailed design process. Table 3-3 describes some key elements of a typical SAD.

Table 3-3 Key elements of SAD

SAD element	Description
Goals of the architecture	This includes the standard principles of architecture as well as organization-specific architecture principles that need to be followed for development of enterprise applications.
Solution constraints	This includes the constraints that have to be met in the architecture for the solution to be acceptable/useful to the customer.
Logical architecture	This includes the logical representation of presentation services, business services, infrastructure services, integration services, data access services, data layer, etc., and their interrelationships as part of the overall solution.
Quality of services	This captures the required "ilities" or NFRs of the system.

Table 3-3 *(Continued)*

Interface/integration requirements	This captures the required information about external systems that need to integrate with the application, especially with relation to the inward and outward data movement at their interfaces, along with references to external system interface design documentations.
Technology selection	This is one of the very important elements, which focuses on the evaluation and selection of the right set of tools, technologies, frameworks and/or commercial off-the-shelf (COTS) products that will be used in the solution implementation.
Technical architecture	This includes the description of how the logical architecture is realized as a technical solution in terms of layers, design patterns and framework components.
Data architecture	This captures the business entities and their relationships. It also includes information such as data life cycle, data security, data storage and data integrity.
Infrastructure architecture	This includes the elements of IT infrastructure including hardware, software, middleware, network and related policies, required for deployment of the application.

3.7.2 Design Documentation

The detailed design of an enterprise application can be documented in several ways—technical layer wise, module/use case wise, component wise, or typically a mix of them. The overall objective of the design documentation is an effective communication among application teams that are involved in design, development and deployment activities. The design documentation should include, for each use case, the presentation, business and data access components. In addition, integration and database designs may be documented separately. Table 3-4 depicts the key elements that need to be captured in the presentation layer design document.

Table 3-4 Key elements of presentation layer design document

Design document elements (for presentation layer)	Description
Screen mockups	This includes wireframes, UI layout, screen templates and event handling. Relevant class diagrams and sequence diagrams may also be documented.
Error handling	This includes recovery actions in the event of errors, and user-friendly diagnostic messages to be displayed.
Input validation	This includes rules for validating user input data on both client side and server side.

Table 3-5 provides a list of key elements captured in a business and data access layer design document.

Table 3-5 Key elements of business and data access layer design document

Design document elements (for business and data access layer)	Description
Business delegate	This captures the set of methods that need to be invoked and underlying access details on behalf of presentation layer request for a specific business service.
Session façade	This captures the list of services that need to be grouped under a facade.
Business service	This captures the business interfaces and their implementation details.
Business model	This captures the business entities and their interrelationships.
Data access layer components	This includes the design details of data access components, object relational mappings, etc.

These documents should be reviewed and baselined as configuration objects before start of the next phase—construction. This brings us to the end of architecture and design phase. We will take a deep dive into the construction phase of an enterprise application in the next chapter.

Summary

This chapter has introduced the concept of enterprise architecture, architecture frameworks and various viewpoints to view an enterprise application. Four viewpoints—logical architecture, technical architecture, data architecture and infrastructure architecture—are explained in this chapter with a focus on their building blocks and interaction with each other.

This chapter has also introduced the common layers in logical architecture and how they get mapped to the technical architecture of an application. In this perspective, several frameworks, design patterns, tools, standards and technologies, which are used to design enterprise applications, are discussed.

In this chapter, we have introduced data architecture and different kind of data representations in an enterprise application. The modeling required designing different types of schemas such as relational data modeling and XML modeling is also discussed.

This chapter has also introduced the building blocks of infrastructure architecture and various key elements that ensure and assure several "quality of services" of an enterprise application such as scalability, security, reliability, availability and performance. The deployment strategy and various factors that need to be considered for devising a deployment view are discussed. The chapter is concluded with an overview of architecture and design documentation, which act as a blueprint to start the construction activity.

Review Questions

1. Define the four domains of enterprise architecture.
2. Explain different kinds of relationships that may exist among classes.
3. What are the typical components of an infrastructure layer of enterprise application?
4. Define UML sequence diagram.
5. What are the different approaches to design session management?
6. Explore DWR framework.
7. Define XSLT.
8. What is a transaction isolation level? Explore different types of transaction isolation levels.
9. What is a rules engine? Explore different kinds of rules engine.
10. Define service oriented architecture. How is enterprise service bus associated with it?
11. Define different types of integration topologies.
12. Explain the application integration options provided by JEE.
13. Explain entity relationship diagrams.
14. Define XSD.
15. Explain SAX, DOM and StAX parsers.
16. Explore the difference between JAXP and JAXB APIs.
17. Explore IPv6.
18. Define DMZ.
19. What are the advantages of virtualization?
20. What is a middleware?
21. What are the disadvantages of SaaS?

Further Readings

Architecture, architecture description, views and view points: http://www.iso-architecture.org/ieee-1471/index.html
Control Objectives for Information and Related Technology (COBIT): http://www.isaca.org/cobit/
Design Patterns with Java Blueprints: http://java.sun.com/reference/blueprints
DWR framework: http://directwebremoting.org/dwr/
EA resources by Gartner: http://www.gartner.com/it/products/research/asset_129493_2395.jsp
EA resources by Microsoft: http://www.msdn.microsoft.com/architecture
Enterprise Architect: http://www.sparxsystems.com/
Information Technology Infrastructure Library (ITIL): http://www.itil-officialsite.com/home/home.asp and http://www.itil.org/en/
Mashups with Yahoo! Pipes: http://pipes.yahoo.com/pipes/
ORM framework: Hibernate: http://www.hibernate.org/
Rules Engine by JBoss: http://www.jboss.org/drools/
Shared Information Data model: http://www.tmforum.org/InformationFramework/1684/home.html
Software as a Service (SaaS) and Salesforce.com: http://www.salesforce.com/
The Open Group Architecture Framework: http://www.opengroup.org/togaf/
Web application security: http://www.owasp.org
XSLT: http://www.w3.org/TR/xslt
Zachman Enterprise Architecture Framework: http://zifa.com

Constructing Enterprise Applications 4

Learning Goals

After completing the chapter, you will be able to:

- Outline construction readiness.
- Create software construction map.
- Develop application framework components.
- Develop application components for corresponding layers.
- Perform code review and static code analysis.
- Formulate build process.
- Perform unit testing.
- Perform dynamic code analysis.

The construction of an enterprise application involves translating the design into code components to build an application, and fulfill the business perspective by leveraging architecture, development techniques, modeling, frameworks, tools and technologies. The translation of design to code typically involves the following activities:

- Achieving construction readiness
- Constructing solution layers—application framework and application components
- Code review and static code analysis
- Compilation and creating deployable packages
- Unit testing
- Code profiling and code coverage analysis

In general, the bigger picture for constructing an enterprise application involves sharing the common practices, procedures, processes and guidelines, irrespective of the technologies and tools used. In this chapter, however, we will be primarily using Java Enterprise Edition (Java EE), and related frameworks, tools and technologies to illustrate the principles, practices and paradigms of construction. Java EE provides the platform to build distributed and multi-tier enterprise applications.

We have explored technical architecture layers and its design components in the previous chapter. Let us now look at the key activities required to start the construction phase, followed by creation, review, verification and analysis of code components, as envisaged by the architecture and design.

4.1 Construction Readiness

The development blueprint of an enterprise application is comprehensive after completion of requirements engineering and design phase in a particular iteration. A few of the activities that are necessary before getting into the construction activities are as follows.

4.1.1 Defining a Construction Plan

A construction plan will consist of the sequence and schedule of tasks required to be carried out in the construction phase, their interdependencies, technical risks, and resources required to implement the task. The primary objective of the plan is to complete the construction phase in the shortest time possible, while not compromising on functionality and quality requirements. To achieve this, tasks are parallelized as much as possible. The ability to parallelize is limited due to interdependencies among tasks, in which case they need to be sequenced appropriately. Further, resources and schedule conflicts also need to be taken into account. There are a lot of activities that can be done in parallel to coding such as test case preparation, setting up the test environments, planning review activities, planning version control, release planning, etc. A setup for continuous integration is also planned to facilitate ongoing integration of code as and when it is developed.

4.1.2 Defining a Package Structure

The code units that are created during construction have to be structured in a meaningful and logical way. This can be based on considerations such as loose coupling, tight cohesion, reusability, use of third-party libraries, etc. The package structure reflects the physical organization of the units of code. It provides a clear picture to the developer about the location of classes they develop relative to rest of the code. The technology and tools used to raise an enterprise application may also influence the package structure.

Xp

Package structures should follow mutually exclusive and collectively exhaustive (MECE) principle, while grouping different application components. Each node of the package structure should have a clear and meaningful name to help in better organization and management of the components.

The package structure followed in LoMS is shown in Figure 4-1. We will further expand the package structures corresponding to the layers in the following sections.

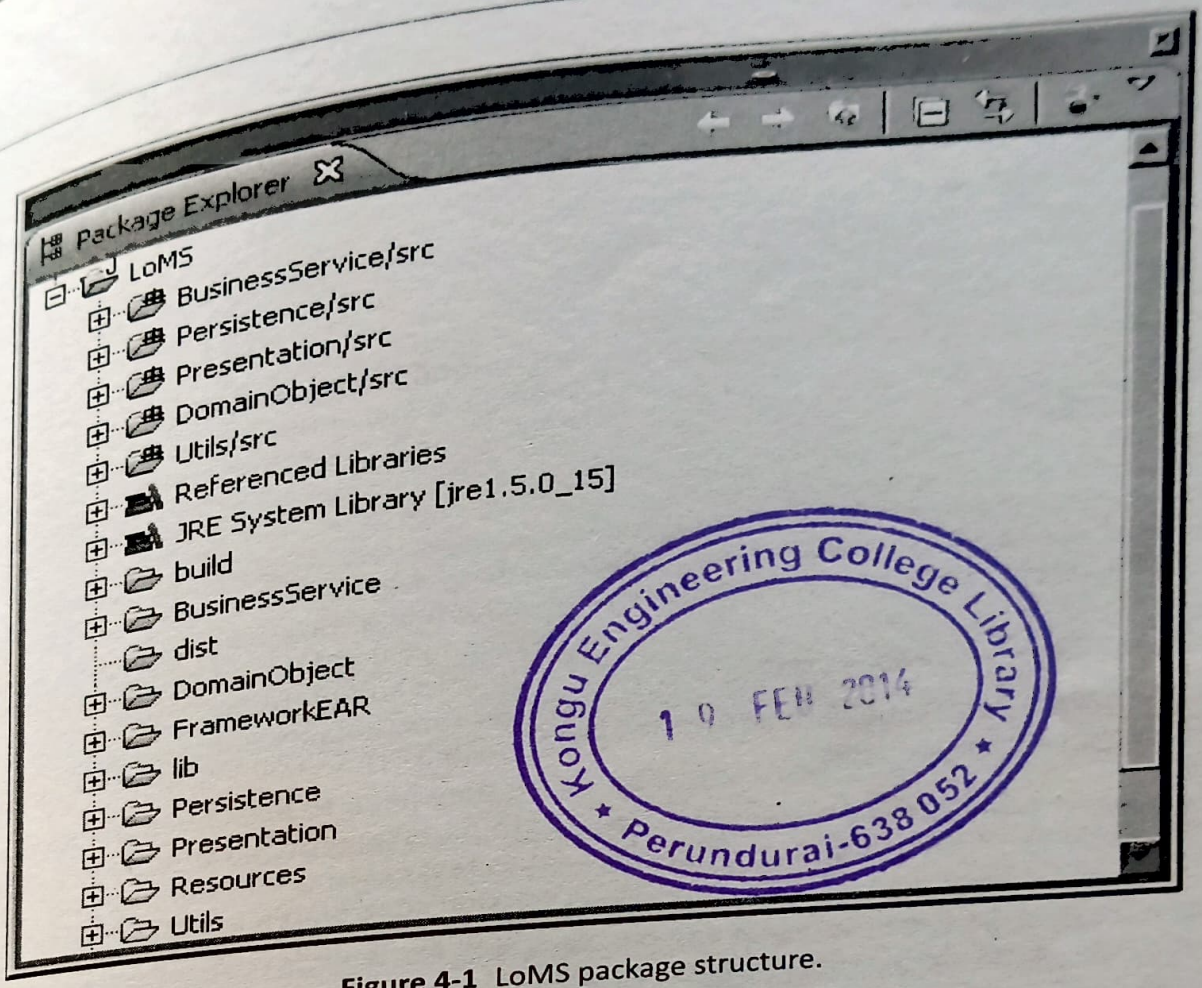


Figure 4-1 LoMS package structure.

4.1.3 Setting Up a Configuration Management Plan

The evolution of artifacts through several versions has to be tracked in order to ensure the integrity and consistency of the artifacts. This process is referred to as *configuration management*, and the artifacts that are managed are referred to as *configuration objects*. This is also required for controlled change management. For example, changes may occur due to a requirement change, or an artifact may have to be rolled back to a previous version. The configuration management plan provides the basis for managing the configuration objects in a consistent and meaningful manner by specifying the tool to be used for the purpose, defining a repository structure, which is aligned to the package structure defined for the project, and specifying appropriate access rights to parts of the repository for the different users.[1]

4.1.4 Setting Up a Development Environment

A development environment is the construction ecosystem that facilitates developers to develop, build, review, verify and analyze code components of an application under development. This environment typically comprises of infrastructure elements (hardware, operating systems and servers), processes

[1] For more information on configuration management plan, you may refer to IEEE Std. 828-1990.

and construction tools, which need to be installed, initialized and configured in order to start the construction. The following is the list of the typical elements that are set up as part of a development environment.

1. Hardware including operating systems
2. Servers such as application, Web, database, directory and portal servers
3. Integrated development environment
4. Third-party libraries
5. Build tools
6. Unit testing tool
7. Profiling tool
8. Static code analysis tool
9. Licenses for all the required software

Assuming that required hardware and operating systems are in place, the following key activities are followed to set up a development environment.

Installation and configuration of servers

Servers that are typically required to be installed in a development environment are file servers, database servers and application servers:

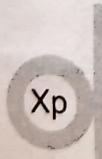
- *File server* typically contains configuration objects under source control.
- *Database server* contains the database with required schema.
- *Application server* contains the various application components. It is typically configured with the required JDBC, JNDI and JMS objects, along with the generic configurations such as object pool size, ports and security settings.

Setting up the integration development environment (IDE)

An IDE is configured with relevant libraries, plug-ins, database and server configurations. A project is set up in the IDE as per the package structure that will be used by the construction team. Members of the development team replicate this IDE configuration in their desktop, and may customize it further based on their role and preferences.

Setting up frameworks

An enterprise application is typically built on top of an application framework which, in turn, may consist of several other layer-specific frameworks. These frameworks are typically extended, configured and instantiated to build the application components. Setting up the framework typically involves importing and configuring the libraries of the framework as part of the project in the IDE.



In addition to all of the above, it should be kept in mind that the construction phase is most intensive in terms of number of people involved and the simultaneous activities in progress. This requires a lot of coordination and communication within the team, besides careful planning and meticulous implementation on the part of the project leadership.

4.2 Introduction to Software Construction Map

- A developer gets a designer perspective of the software solution from several design output such as relational models, UML sequence diagrams and class diagrams. To understand the design and translate it into code, a developer may often need some more information on the design.
- The common dilemma of a developer while coding may include the following:
  - How does the overall picture of components look like?
  - What is the relationship between the solution components?
  - How do the components interact and interface within and outside the layer?
  - How to visualize the flow of functionality across various technical architecture layers?
- Based on our experience, we felt a need to represent the design elements in a different form for better understanding of components, their interactions, and to help visualize the overall solution from a developer perspective. This brought out the new form of representation of various components which we christen as *software construction map* (SCM).
- A SCM coupled with existing design models, brings out several benefits to the develoment team such as visualizing the big picture of solution, inter and intra relationships among various components across the layers, and multiple hand offs between the development team.
- Readers will be introduced to the SCM in the following section.

Let us now explore the construction of code components which corresponds to the technical architecture layers.

4.3 Constructing the Solution Layers

Construction of an enterprise application typically does not start from scratch, even for a greenfield development project. Every organization usually has a reusable assets repository and frameworks to ensure that the wheel is not reinvented every time. Reusable assets repository may have different components as follows:

- An application framework is one of the most important reusable components. It provides the plumbing code across all the layers of an enterprise application.
- A repository of technical components typically has components such as workflow component, rules engine, infrastructure services components and generic reporting components.
- A repository of domain specific components typically has components related to business such as *check credit rating* component and *quote generation* component in the banking domain.

Use cases captured in requirements engineering phase are implemented using these reusable components by either directly using or extending them. These components provide a standard way for the entire development team to construct an enterprise application. This section will take the readers through the construction of technical architecture layers of an enterprise application.

4.3.1 Infrastructure Services Layer Components

As described in Chapter 3, infrastructure services layer of an enterprise application comprises of general purpose components such as logging, session management, security, exception handling, auditing, caching, notification and reporting, which are typically used across all the layers. These components are usually designed and constructed as part of the application framework. A common scenario is that such components are readily available off the shelf, and are plugged into the framework with enhancements and proper configuration as required.

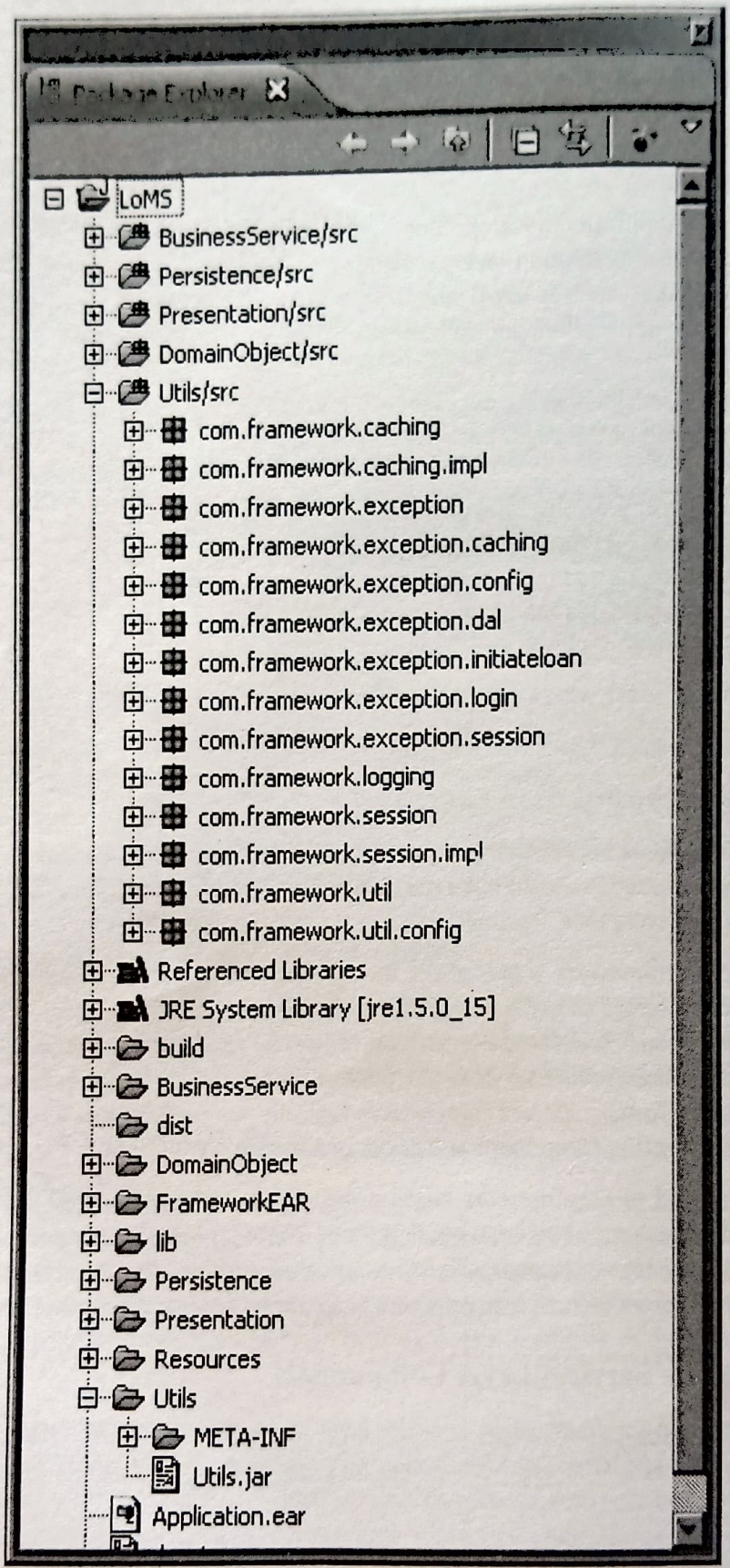


Figure 4-2 Infrastructure services layer components—package structure.

The elements of infrastructure services layer in LoMS are grouped in the *Utils* folder, as depicted in the package structure diagram (Figure 4-2).

Let us now explore the construction of some key components of infrastructure services layer such as logging, session management, exception handling and caching.

Logging

For a large distributed application development, logging is an important element of infrastructure services used by almost all the solution layers, as explained in the previous chapter. As you may have noted, third party packages, such as Log4J, are generally used to implement the logging functionality in a typical Java enterprise application, and calls to the package APIs are inserted in the application code. As the best practice to improve the maintainability and flexibility of code, it is important to insulate the application from a specific logging package. For example, in LoMS, the `LoggingManager` class provides an encapsulation of the Log4J package, and helps in decoupling the application from package specific APIs.

As shown in Figure 4-3, the `LoggingManager` class provides a facade for the logging mechanism. It has static methods for logging information into centralized logging store/files. Any object may be passed to `LoggingManager` as long as it implements the `ILoggable` interface. `LoggingManager` instantiates the Logger of Log4J package and invokes the appropriate method to log messages.

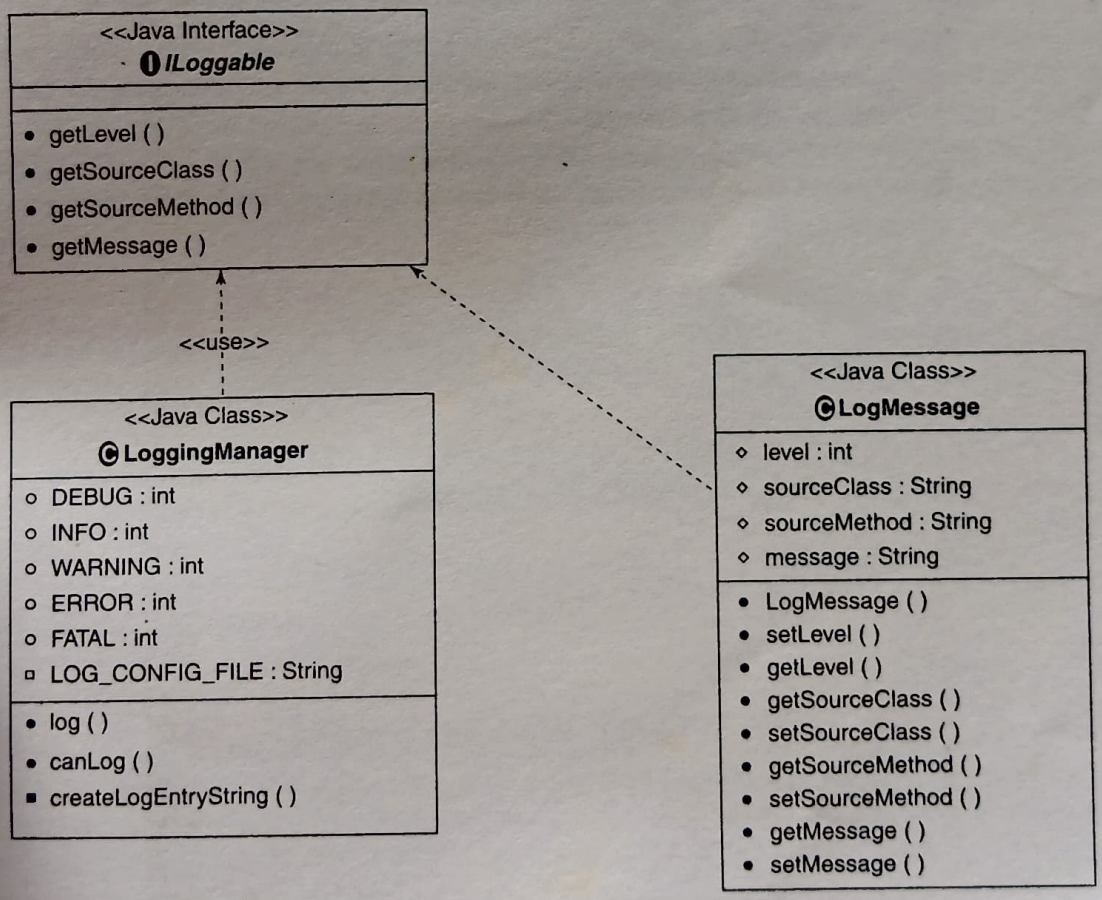


Figure 4-3 Logging framework component.

Code Listing 4.1 depicts the partial implementation of the `LoggingManager`.

```
public class LoggingManager
{
        static public final int DEBUG = 0;
        static public final int INFO = 1;
        static public final int WARNING = 2;
        ------------------------------------------
        /** Name of the Log4j config file **/
        private static final String LOG_CONFIG_FILE = "config/Log4jConfig.xml";
        /**Provides a static initializer for log4J initialization**/
        static
        {
          ClassLoader classLoader =LoggingManager.class.getClassLoader();
          try {
            URL fileUrl = classLoader.getResource(LOG_CONFIG_FILE);
            if (fileUrl != null)
                {
                  DOMConfigurator.configure (fileUrl);
                }
            else
                {
                  Logger rootLogger =
                  Logger.getRootLogger();
                  rootLogger.setLevel(Level.OFF);
                  System.err.println("Log4j not Initialized : Could not find
                  any Log4j Config File.");
                }
            catch(Exception ex) {
              System.err.println("Exception Occcured in Log4j initialization");
            }
          }
        /** Log method for logging the application messages**/
        public static void log(Loggable itemToLog)
        {
          String sourceClass = itemToLog.getSourceClass();
          if(sourceClass == null)
          {
            sourceClass = "";
          }
          String sourceMethod = itemToLog.getSourceMethod();
          if(sourceMethod == null)
          {
            sourceMethod = "";
          }
```

```
        String message = itemToLog.getMessage();
        if(message == null)
        {
          message = "";
        }
        Logger logger = Logger.getLogger(sourceClass);
        String logEntryString = createLogEntryString(sourceClass,
        sourceMethod, message);
        int loggingLevel = itemToLog.getLevel();
        switch (loggingLevel)
        {
          case DEBUG : logger.debug(logEntryString);break;
          case INFO : logger.info(logEntryString);break;
          ------------------------------------------------
          ------------------------------------------------
        }
      }
}
```

Code Listing 4.1 `LoggingManager`—façade to the Logging package API.

`LogMessage` class takes care of setting the log level, source class, source method and logging message. As and when required, as depicted in Code Listing 4.2, enterprise application layers create the `log` messages and call the `log` method of `LoggingManager` to write the statements into a log file. It is always better to check whether the logger is enabled for the specific level before logging the message. In LoMS, the `canLog` method of `LoggingManager` checks whether the level passed as input is enabled to log the message.

```
    // Log entry
if(LoggingManger.canLog(LoggineManger.DEBUG, CLASS_NAME))
{
        String methodName = "getCustomerDetails";
        LogMessage logMessage = new LogMessage(LoggingManager.DEBUG,
CLASS_NAME, methodName,
       "Viewing the customer details");
        LoggingManager.log(logMessage);
}
```

Code Listing 4.2 Calling logging operation.

The log file name and path are configured in the `Log4J` configuration file. Loggers can assign different types of log levels. Typical log levels are `TRACE`, `DEBUG`, `INFO`, `WARN`, `ERROR` and `FATAL`. In LoMS, `DEBUG` log level is used. The logging framework exposes the `log` method to all other application layer components, and internally `LoggingManager` will call the printing methods of a logger instance depending on the log level.

In

A logging request can be enabled by setting the log level equal to, or higher than, the level of its logger. The typical order of level is defined as DEBUG < INFO < WARN < ERROR < FATAL.

Large-scale enterprise applications usually generate thousands of log requests that may indirectly hit the performance of the application. The deployment team has to appropriately set the log level at the time of deployment to control the volume of log generated. Readers should note that the above represents one of the several ways of implementing the logging strategy for an enterprise application. Another approach to implement logging in an enterprise application is to use *aspect-oriented programming* (AOP) to achieve better modularity. AOP is a programming paradigm that enables separation of concerns into *aspects*. An aspect is a feature that cuts across the core features of an application, e.g., logging and error handling.

Exception handling

Exception handling is a mechanism used by an application component to take appropriate corrective or preventive measures, due to occurrence of an exception at runtime of the application. Chapter 3 has presented the readers with various important points that need to be taken care of while designing an exception handling strategy. Exception handling is one of the mandatory infrastructure components that typically gets fabricated as part of the application framework components. Figure 4-4 depicts exception handling as implemented in LoMS. Various categories of exceptions and their interrelationships are also shown in Figure 4-4. The implementation of the exception handling components is available for reference in the LoMS codebase.

Session management

In Chapter 3, you have familiarized with the session management concepts, and approaches, which need to be considered while designing a session management strategy. As mentioned earlier, LoMS uses "in-memory" storage technique for session management, and JBoss caching solution to implement it. In the session management implementation of LoMS, the application creates a unique `SessionID` using session management components after successful authentication of a user. The `SessionID` gets stored in an `HttpSession` object, along with the application state information of that user. Whenever an application component requires information from a session object, it retrieves the `SessionID` from the `HttpSession` object, and passes it to the caching component to further fetch the related session details.

Figure 4-5 depicts the overall session management component, which has been designed and implemented as part of the LoMS application framework. The `Session` class contains the logic for creation of a session, invalidating a session, retrieval of information, and adding and removing the information details from `Session`.

Code Listing 4.3 depicts the `Session` object constructor, which ensures the creation of a new `SessionInfo` object, for every session, which gets stored in cache with `SessionID`.

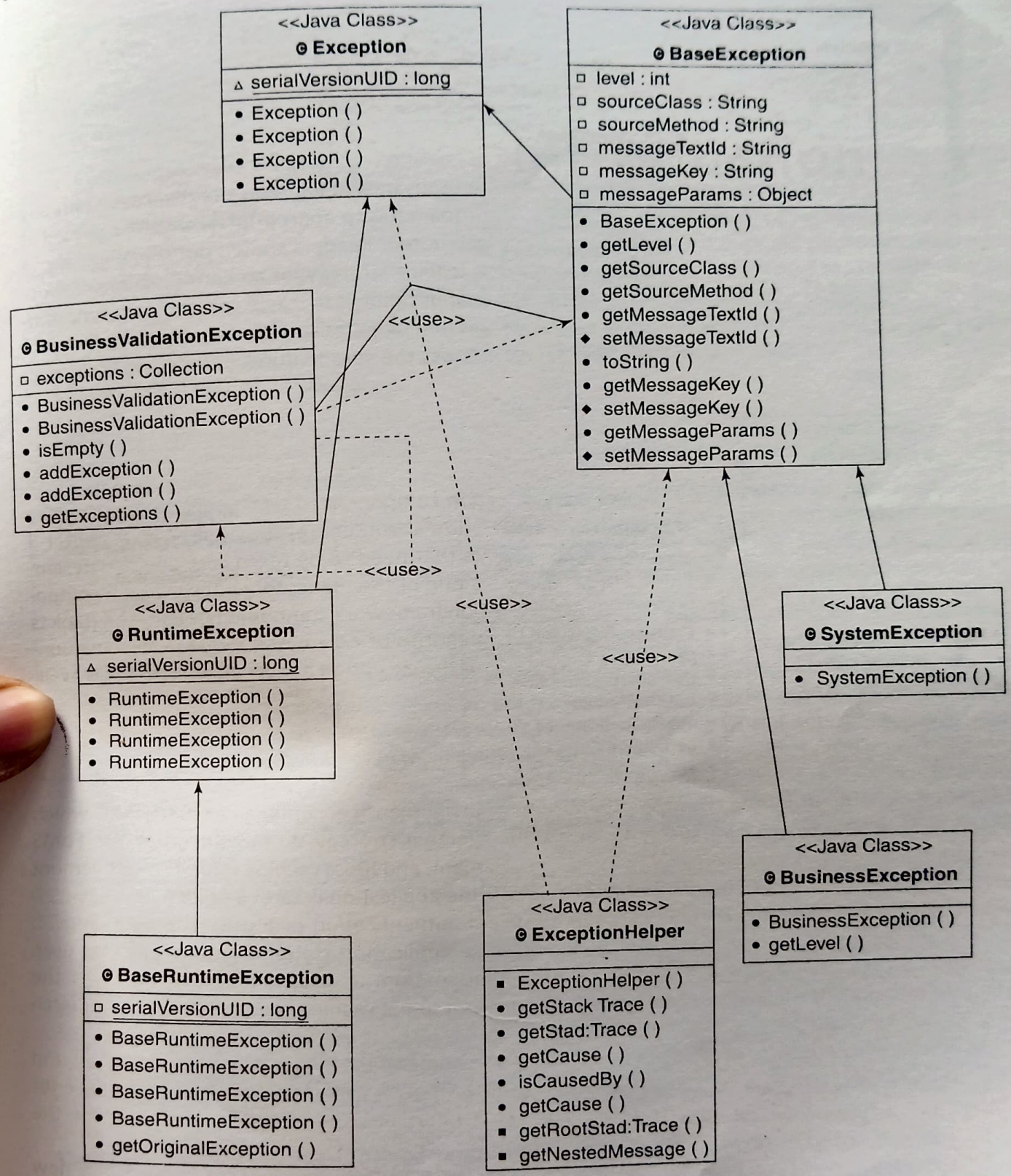


Figure 4-4 Exception handling component.

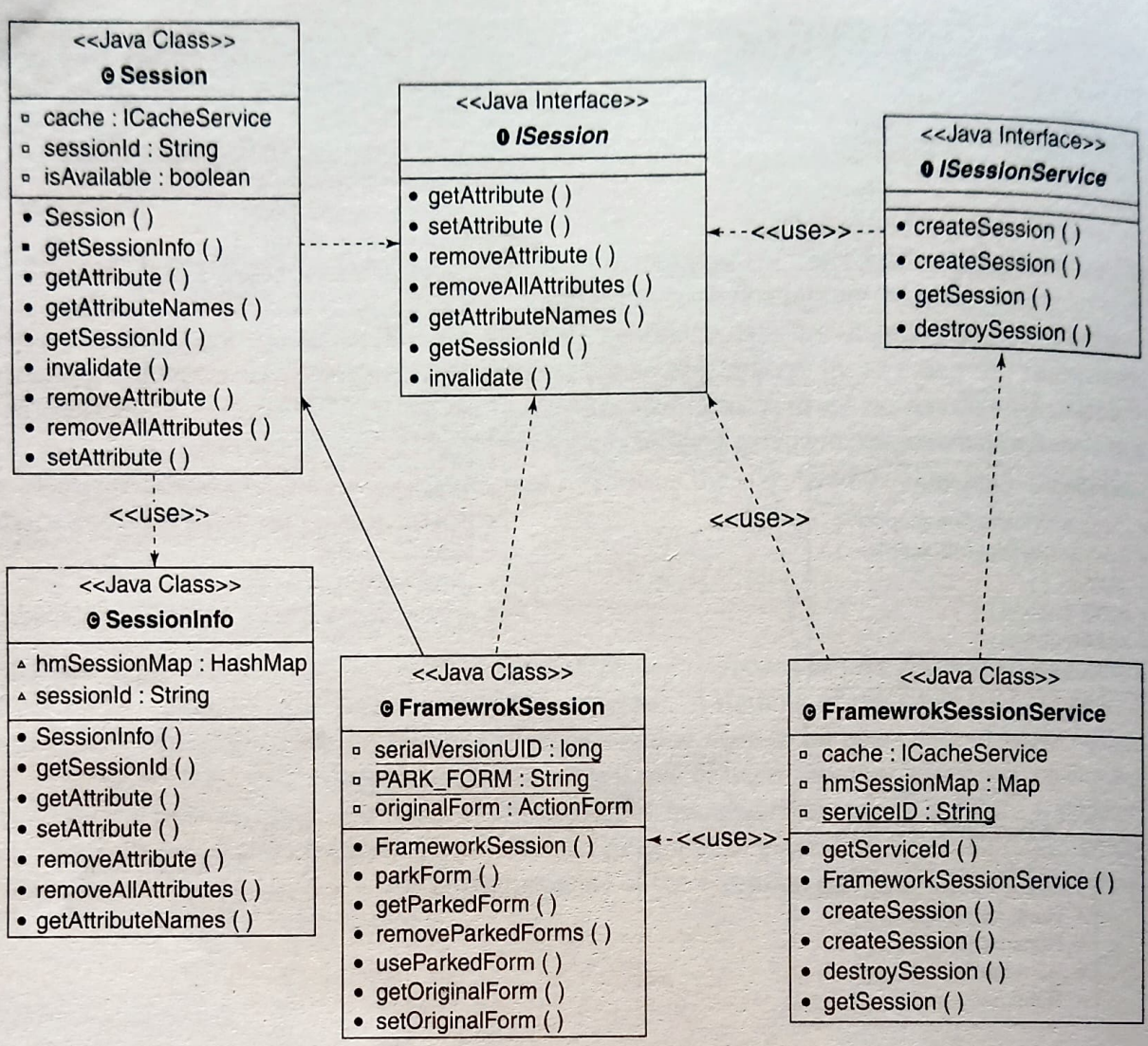


Figure 4-5 Session management component.

```
public Session (String sessionId, ICacheService cache) throws
SessionServiceException {

LogMessage logMessage = new LogMessage(LoggingManager.DEBUG,
CLASS_NAME,"Session","Session() Constructor");

LoggingManager.log(logMessage);

try {
     this.sessionId = sessionId;
     this.cache = cache;
     SessionInfo objSessionInfo = new SessionInfo(sessionId);
     cache. put(sessionId, objSessionInfo);
     isAvailable = true;
    }
catch (Exception ex) {
```

```
throw new SessionServiceException ("Session","Session Constructor","Error
in creating a new session with Id" +ex.getMessage(), null);
    }
}
```

Code Listing 4.3 Creation of `SessionInfo` object.

`SessionInfo` constructor creates a `HashMap` object to store session attribute values. `SessionService` class has the functionality for storing all the session objects based on the `SessionID`.

When it comes to the requirement of clustering enterprise applications, it requires handling the http session in a reliable fashion. Session attributes must be serializable, if they are to be used in a clustered environment. In case of large session objects, session data, which are not required to get replicated, can be declared as transient that means that this data will not be replicated across the clustered environment.

Enterprise applications optimally take the advantage of built-in session management features of the application server. Any commercial application server typically has the features, as shown in Table 4-1.

Table 4-1 Session management features

Features	Description
Session affinity	*Session affinity* is a mechanism, where all requests that are part of a session are directed to a particular node of the clustered environment. This helps enhance the application performance by caching the session objects.
Session tracking	*Session tracking* is a mechanism to use cookies for session management. The application server generates a session ID and returns this ID in a cookie to the browser. The browser stores the cookie and includes in it every request, which enables the server to associate the request with the session. The cookie is destroyed in the events such as termination of the session, browser closure, expiry of cookie, or session time out.
In-memory session and persistent session	*In-memory session* and *persistent session* are mechanisms wherein the session attributes are stored in the memory or persisted in a database respectively.

Xp

One of the other very important features of session management is managing synchronized access. Consider the case of two servlets changing the same session properly at the same time, or one servlet invalidating the session while another one is still trying to use it; data inconsistencies may occur as a consequence. To avoid such issues, the session component has to synchronize access following the `request.getSession()` call, along with checking the validity of the session.

Caching

Chapter 3 provided a primer on caching and related tools to implement it as a framework component. In LoMS, the caching component is implemented as a tree cache, where the cache is organized as a tree. Each node in the tree is implemented as a map for storing key-value pairs.

As shown in Figure 4-6, `ICacheService` is an interface that exposes the methods to get, put and remove object from a cache. Lazy loading mechanism is used for populating the cache. The entire static data loading is defined in a configuration service. When there is a request for data, the configuration service queries the cache for the object. If the object is not present in the cache, the configuration service accesses the data source to load the data into cache. The configuration service starts retrieving the data from cache for subsequent requests for the same data.

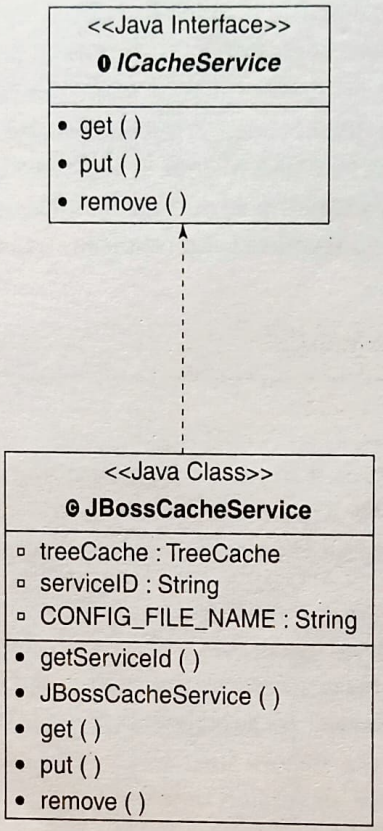


Figure 4-6 Caching—infrastructure component.

An instance of `treeCache` is created within the constructor of `JBossCacheService`, as shown in Code Listing 4.4.

```
public JBossCacheService() throws CacheServiceException {
LogMessage logMessage = new LogMessage(LoggingManager.DEBUG, CLASS_NAME,
"JBossCacheService","Entering Constructor of JBossCacheService");
LoggingManager.log(logMessage);
try {
     treeCache = new TreeCache();
     PropertyConfigurator config = new PropertyConfigurator();
     config.configure(treeCache, CONFIG_FILE_NAME);
     treeCache.setUseInterceptorMbeans(true);
     treeCache.createService();
     treeCache.startService();
    }
catch (Exception ex) {
```

```
        throw new CacheServiceException("JBossCachingService"," Cache
        Constructor", "","Error in Creating Cache Service"+ex.getMessage(),null);
    }
}
```

Code Listing 4.4 `treeCache` instantiation.

Caching modes, locking level and transaction isolation level properties are configured in the JBoss caching service configuration file. Caching mode is required to declare whether caching is required locally or across the cluster. Readers can delve into the code and configuration files provided in the reference code for more implementation details.

An applications' data may be cached at the client-side or at the server-side. Client-side caching is required when there is a strict requirement to conserve network bandwidth. Server-side caching is useful when multiple users connecting to the server require access to the same data. Server-side caching is applicable to all the layers of an *n*-tier enterprise application.

Tt JBoss Cache, JCS (Java Caching System), OSCache are a few of the server-side caching frameworks. Using any of them to implement the caching functionality saves significant development efforts and time and, in general, results in a more robust solution.

To decide whether to opt for data caching, designers have to look into some key questions—Is the amount of cached data manageable? Is the cached data read-only? What are the consequences of stale data? Caching, only if used optimally, provides the expected performance and scalability improvements.

4.3.2 Presentation Layer Components

Construction of presentation components can be done in a variety of ways. Simplest of the options is to construct the application using JSP, Servlet and plain old Java object (POJO) components. These are enough to build a simple presentation layer. But enterprise application developers prefer to use the frameworks to develop the presentation layer components, as they come with loads of out-of-the-box benefits.

The design team has to take care of various things to construct a successful presentation layer. They have to select the right design patterns, decide the right look and feel, give due consideration to input validation, achieve maximum reuse, ensure ease of navigation, session management and caching of data, internationalization and several other issues.

Frameworks and technologies

There are a variety of Java based frameworks available for all of the technical layers of an enterprise application—so also for the presentation layer. This provides flexibility in terms of choice, but also complicates the choice of the framework. Selecting the right framework is the key to ensure performance, maintainability and extensibility of an enterprise application. It also brings standardization and structure to the construction.

Tt Presentation layer construction can be done using frameworks like Struts, JSF, Tapestry, Stripes and many more.

Selection of a framework typically happens during the architecture phase of an enterprise application development as part of laying down the technical architecture. The selection of the presentation layer

framework usually depends on features such as internationalization support, navigation support, request processing, asynchronous JavaScript and XML (AJAX) support, data validation support, testing support and multichannel support. The selection of a framework may not be always based purely on technical reasons. It might be influenced by several other factors such as availability of framework documentation, maturity of the framework, community/vendor support, usability, QoS considerations, availability of skilled workforce, compliance to industry standards and company policies. To start with, let us explore a few of the popular presentation frameworks and evaluate them on the features offered by them.

Struts and JSF are the two most popular presentation layer contenders among the Java presentation frameworks. They both exhibit a lot of similarities. Both are MVC based frameworks, and provide support for managing the request life cycle, request validation, configurability of page navigation and many more things. Let us explore how they differ, as shown in Table 4-2.

Table 4-2 JSF vs. struts

Features	JSF	Struts
Acceptance	JSF is governed by JCP and is specified in JSR 127 as a Java EE standard	Struts, by far, is the most mature and popular framework. But it is not part of the Java EE specifications
Working	Follows an event-driven model, where components of a JSF page can trigger appropriate events	Follows an action-driven model, where a page request is treated as a single event
Support	Relatively new, but gaining maturity rapidly	Very mature and stable, supported by a large developer base and many IDEs
User Interface	Based on a standard UI component model	Does not have a standard UI component model
Client support	JSF supports the non-HTML rendering kit, which makes it easier to plug-in support for WML or XML based clients also	Struts tags generate only HTML
Development	JSF being based on a standard UI component model, supports RAD and reusable components	Struts is based on custom tag libraries and do not support RAD and reusable components

JSF is surely the future, as it is governed by the JCP as a standard specification. But it is yet to be a universal choice for building the presentation layer. If application presentation layer requires features, such as sophisticated presentation components, use of portlets and support for multiple clients, then JSF may be the preferred choice, otherwise Struts is also a good choice.[2]

Package structure

In LoMS, Struts framework is used along with custom application framework components for laying the foundation of the presentation layer. Application specific components are created on top of these

[2] You can learn more about Struts and JSF at http://struts.apache.org/ and http://java.sun.com/javaee/javaserverfaces/, respectively.

framework components. The application framework components and the Struts framework components help increase the reusability of components and standardization of construction. You will encounter a few of these components in the next section. To start with, let us take a look at Figure 4-7 that depicts the package structure of the presentation layer components used in LoMS.

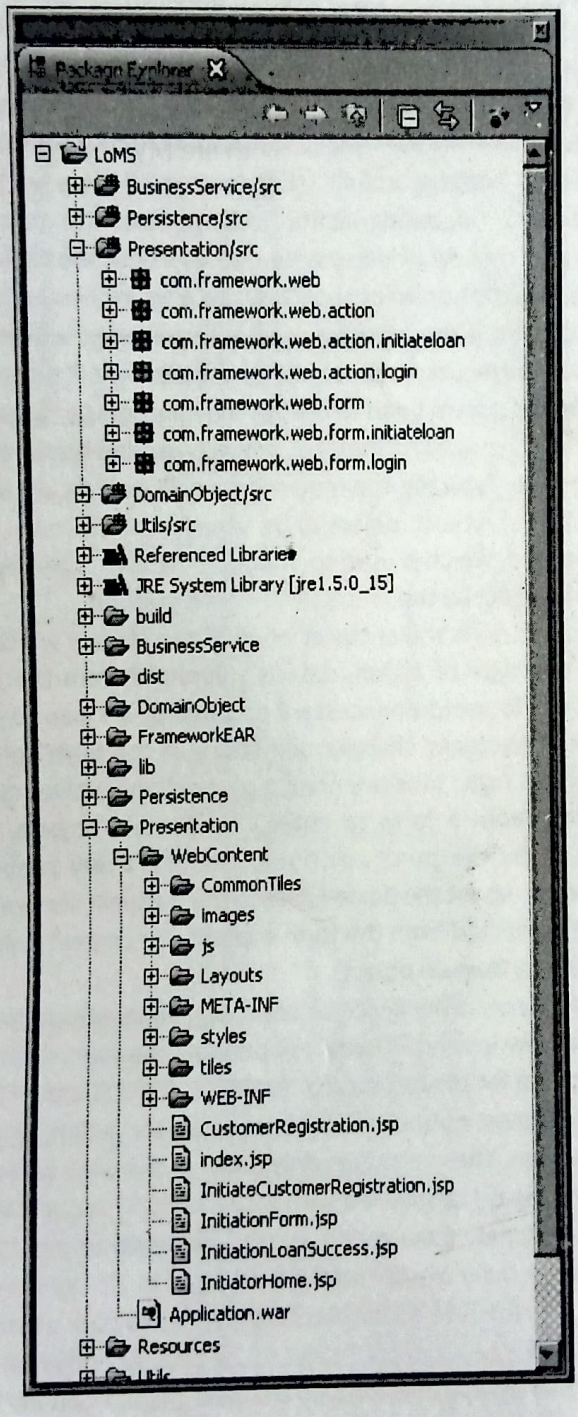


Figure 4-7 Package structure of presentation layer.

It is evident from Figure 4-7 that the presentation layer has broadly two types of artifacts—Web content and source code for the presentation layer. Web content typically comprises of static images,

tiles, style sheets and related configuration files. The source code has the JSPs and other Java artifacts related to the application framework components and the application components.

Application framework components

In LoMS, presentation layer application framework components extend the Struts framework. Figure 4-8 shows the static view of the presentation layer related application framework components.

The application framework has `FrameworkBaseAction`, which is inherited from Struts `Action` class, and it is the mother of all types of application actions and specific classes required for a generic application development. `FrameworkBaseAction` is inherited by specialized actions. The specialized actions are categorized as "viewing action" (`FrameworkTilesAction`), "submitting action" (`FrameworkSubmitAction`), "updating action" (`FrameworkUpdateAction`) and "parking form action" (`ParkFormAction`). All of these base classes utilize the template pattern to define the overall process and provide specific hooks for subclasses to plug in their custom logic.

`FrameworkTilesAction` is the ancestor of all action classes, which are created for displaying a page. When a display action is executed, the required business data is retrieved from the related business service, and then loaded to a form bean to render the contents of the page. Subclasses only need to be concerned with the `retrieveData` method, which is used to retrieve all business data required by this action to render the page. Typically, the required domain objects are retrieved through the business service and stored in the `context` object to be used by `loadForm`. Every display action class needs to implement this method, which is used to populate the form used by this action with business data stored in the `context` object by the `retrieveData` method.

`FrameworkSubmitAction` is the ancestor of all action classes which are created for processing the POST requests. In this type of action, data is submitted from the browser to the server to fulfill some business function. To avoid unnecessary processing, `FrameworkSubmitAction` tries to detect whether the user has actually changed something in the form before it invokes the worker method to process the business logic. Implementing a post action involves two methods. The method `unloadForm` transfers data from a form to criteria or domain objects, and stores them in the `context` object to be used by the `persistData` method. Every post-action class implements the `persistData` method to submit the posted data to the business service for persistence. The data should have been previously collected from the form and put in a context object by the `unloadForm` method in the form of criteria or domain objects.

`FrameworkUpdateAction` is the ancestor of action classes which are created for handling the actions that only update the view locally without repopulating the form with fresh data from the backend, or submitting the form data for processing. For example, the dropdown list of a selection box may need to be rebuilt to show different options in response to a user action, or certain page components may need to be shown or hidden. These changes only occur in the view and do not affect the underlying model. This action should always be invoked with an HTTP POST request in order to retain the state of the form. To implement an update view action, a subclass needs to provide the specific logic in the `process` method, which is the main worker method declared in `FrameworkBaseAction`.

`ParkFormAction` is the concrete action class that is used to park a form. It is used in the place of regular post actions that extend `FrameworkSubmitAction`. Actions that use `ParkFormAction` normally redirect to the display action that renders the next page. Form parking means saving a form in a Web session temporarily, instead of submitting the form data to the business tier to be processed immediately. This might be required in a situation where the current unit of work spans more than one page (and therefore the intermediate steps must be temporarily cached in the session). The business delegate is only invoked from the post action on the last page in the group to process the data

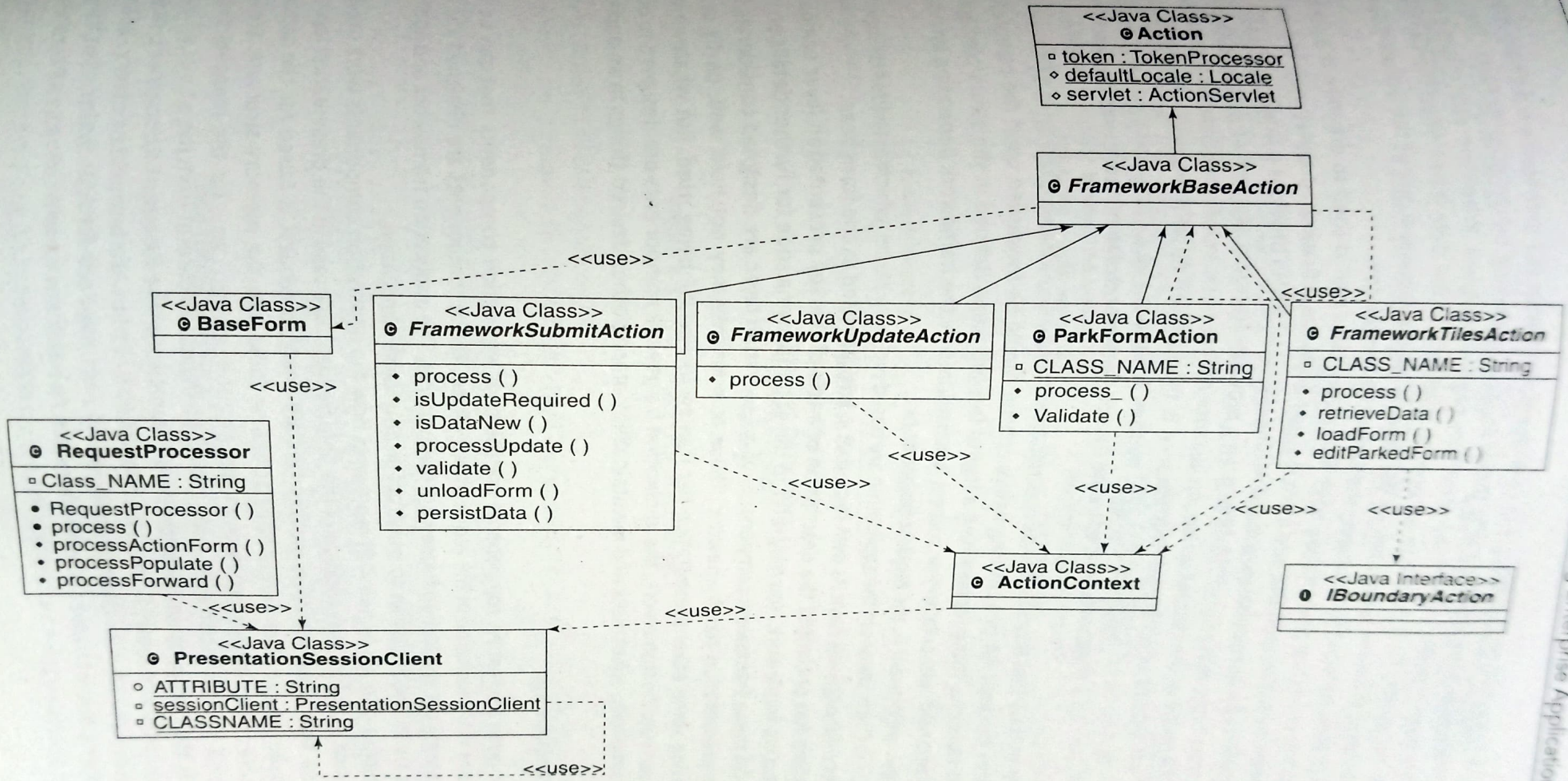


Figure 4-8 Presentation layer—application framework components.

collected by all pages in a group. There can be a situation wherein the user leaves a primary page to go to a secondary page from which he may return to resume working on the primary page. The form for the primary page is parked before the secondary page is displayed. When the user returns to the primary page, the parked form is used to restore the page back to the state it had when the user left it. In both these cases, the form should still be declared as having request scope in the action mapping. The framework is responsible for managing the cache of parked forms.

`ActionContext` is a presentation layer framework class that is used to organize the standard parameters used by Action methods into a single structure to simplify method signatures. This helps in achieving simplicity for developers in terms of writing the code. It also provides a single-point of access to the `PresentationSessionClient`. It also provides transient storage of any action-specific data during the scope of the current action. An instance of `ActionContext` is created by the framework at the entry-point of an action. This instance is then passed to all the worker methods of the action. `ActionContext` has `getSession` method to return the `FrameworkSession` object. The `ActionContext` instance is created by the `execute` method of the `FrameworkBaseAction` class.

The only method that may need to be overridden in `FrameworkBaseAction` is `getForward`, which provides the logic for determining where control should be forwarded upon the completion of the action. It returns a string representing a logical forward name defined in the struts configuration file. If null is returned, either failure or success is substituted by the framework depending on whether error messages are present in the request object or not.

All `ActionForm` classes in the application are based on `BaseForm`, which extends `ActionForm` in Struts. Every JSP page will have its own form that is basically used as the form bean.

This section has presented the description of a few of the key presentation layer components of the application framework. You may refer to the accompanying code for further details on implementation. All these framework components of presentation layer are designed considering Struts as the base presentation tier framework. These components may not hold well, partly or completely, in using some other presentation tier framework like JSF being used. But whatever be the choice of layer-specific framework, the application framework components are designed to achieve reusability, simplicity, robustness and maintainability among other desired things in an application framework.

Application components

Presentation layer application components are the core application components that can be traced back to the use cases captured in the requirement phase. These components are designed and implemented by reusing the application framework components. The application framework and application components are stitched together to construct an enterprise application.

The sequence diagram (Figure 4-9) represents how the application components work together in the presentation layer for the realization of the "initiate loan" use case in the presentation layer.

When the actor bank customer invokes "Initiate Loan", a request is raised for the action path `initiationHome.do`. The `RequestProcessor` looks into the memory structure, created by the `ActionServlet` by parsing through the `struts-config.xml` for the mapping details of `initiationHome.do` action path. Code Listing 4.5 depicts an excerpt from the `struts-config.xml`, which is the configuration file of the Struts framework. The `RequestProcessor` populates the `com.framework.web.form.initiateloan.InitiateLoanForm` form bean and invokes the `execute(ActionMapping mapping, ActionForm form, HttpServletRequest request, HttpServletResponse response)` of `com.frame work.web.action.initiateloan.ViewInitiationFormAction` action class.

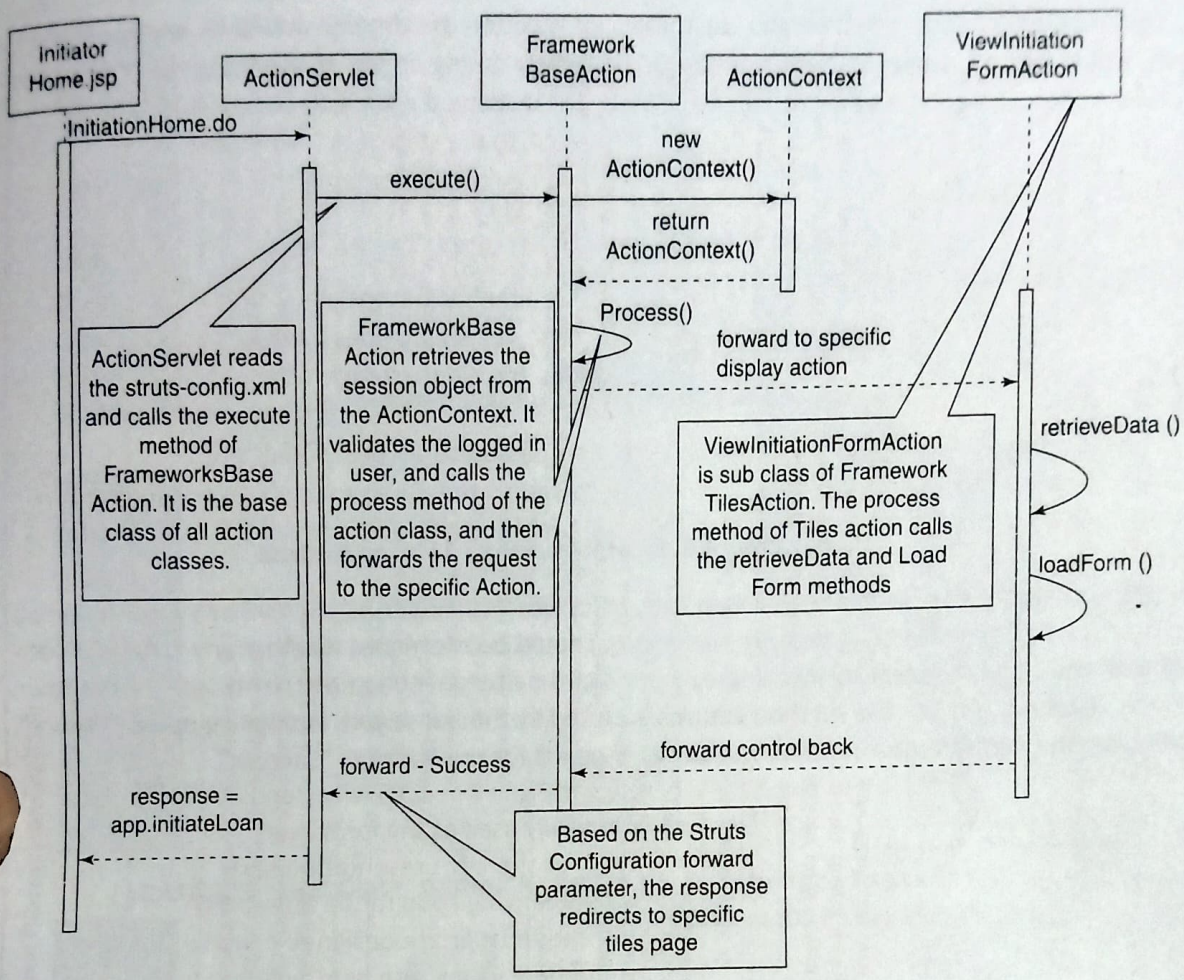


Figure 4-9 View initiation form sequence.

```
...
<form-bean name="initiationForm"
      type="com.framework.web.form.initiateloan.InitiateLoanForm" />

...
<action path="/initiationHome" name="initiationForm" scope="request"
input="InitiatorHome.jsp"
type="com.framework.web.action.initiateloan.ViewInitiationFormAction"
            validate="false">
                  <forward name="Success" path="app.initiateLoan" />
                  <forward name="Failure" path="/index.jsp" />
</action>
```

Code Listing 4.5 Excerpt from `struts-config.xml`.

The `execute(ActionMapping mapping, ActionForm form, HttpServletRequest request, HttpServletResponse response)` method of `com.framework.web.action.FrameworkBaseAction`, which is the super class of `com.framework.web.action.initiateloan.ViewInitiationFormAction` action class, instantiates the `ActionContext` populating the `ActionMapping`, `ActionForm`, `HttpServletRequest`, `HttpServletResponse` and `Locale`.

The method checks whether the customer information is already available in the session, which would be set while authenticating the customer during login. If available, it invokes the `process(ActionContext context)` method. These steps are listed in Code Listing 4.6.

```
...
if (null != user) {
                    if (LOGIN_PATH.equals(requestPath) &&
(context.getSession().getAttribute("SESSION_EXIST") == null)) {
                        // ...
                    } else {
                        forward = process(context);
                    }
            }
```

Code Listing 4.6 Excerpt from the `execute` method.

The `execute(ActionMapping mapping, ActionForm form, HttpServletRequest request, HttpServletResponse response)` method determines whether any error is associated with the request object by invoking `errorsExist(ActionContext context)`, as shown in Code Listing 4.7. If so, the method returns `ActionForward` object for the mapping "Failure". Otherwise, the method returns `ActionForward` object for the mapping "Success".

```
...
if (forward == null) {
forward = errorsExist(context) ? FAILURE_FORWARD :SUCCESS_FORWARD;}
return mapping.findForward(forward);
```

Code Listing 4.7 The `execute` method continued ...

The corresponding view, `InitiationForm.jsp` is rendered to the customer. The above example illustrates how the presentation layer's application components are built by leveraging application framework components. Let us now conclude the presentation layer components section by looking at some sample software construction maps that are self-explanatory. They capture the big picture of several application and application framework components, participating in a specific scenario in an easy-to-understand fashion. Developers can complement them with other design diagrams, as necessary, to aid the construction task.

Figure 4-10 is a software construction map to capture the AJAX implementation in "validate SSN" scenario of the "Initiate Loan" usecase. In this scenario, a user-supplied SSN is validated in order to fetch the preapproved loan amount for a customer. The "validate SSN" scenario is implemented using Web services.[3]

The next software construction map, as shown in Figure 4-11, captures the interactions among the session and caching management components in LoMS application.

The software construction map in Figure 4-12 illustrates the login and concurrent user session validation usecase, which is another crucial requirement in any multiuser enterprise application.

[3] You will explore more on this in the integration layer components section in this chapter.

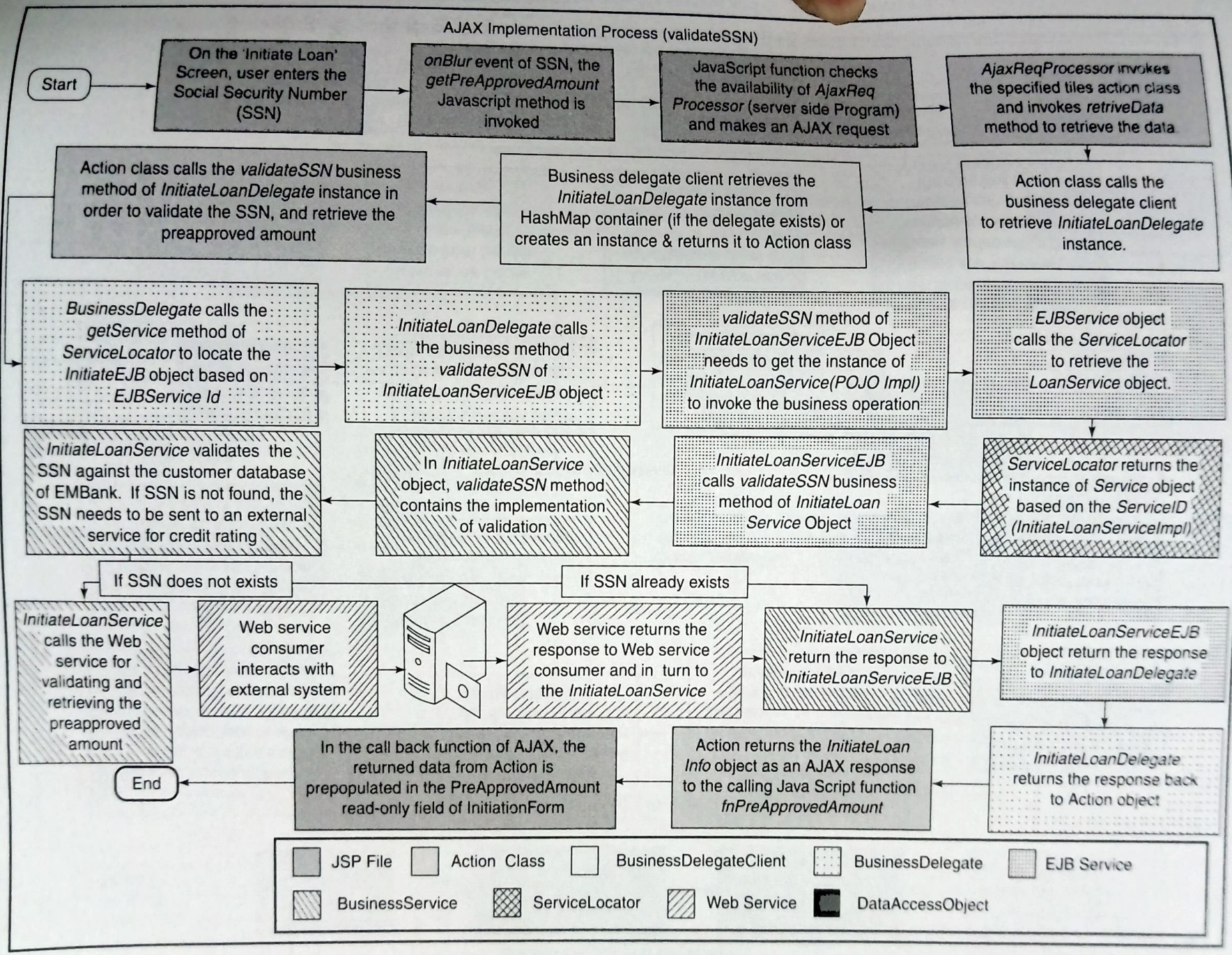


Figure 4-10 Software construction map for AJAX implementation.

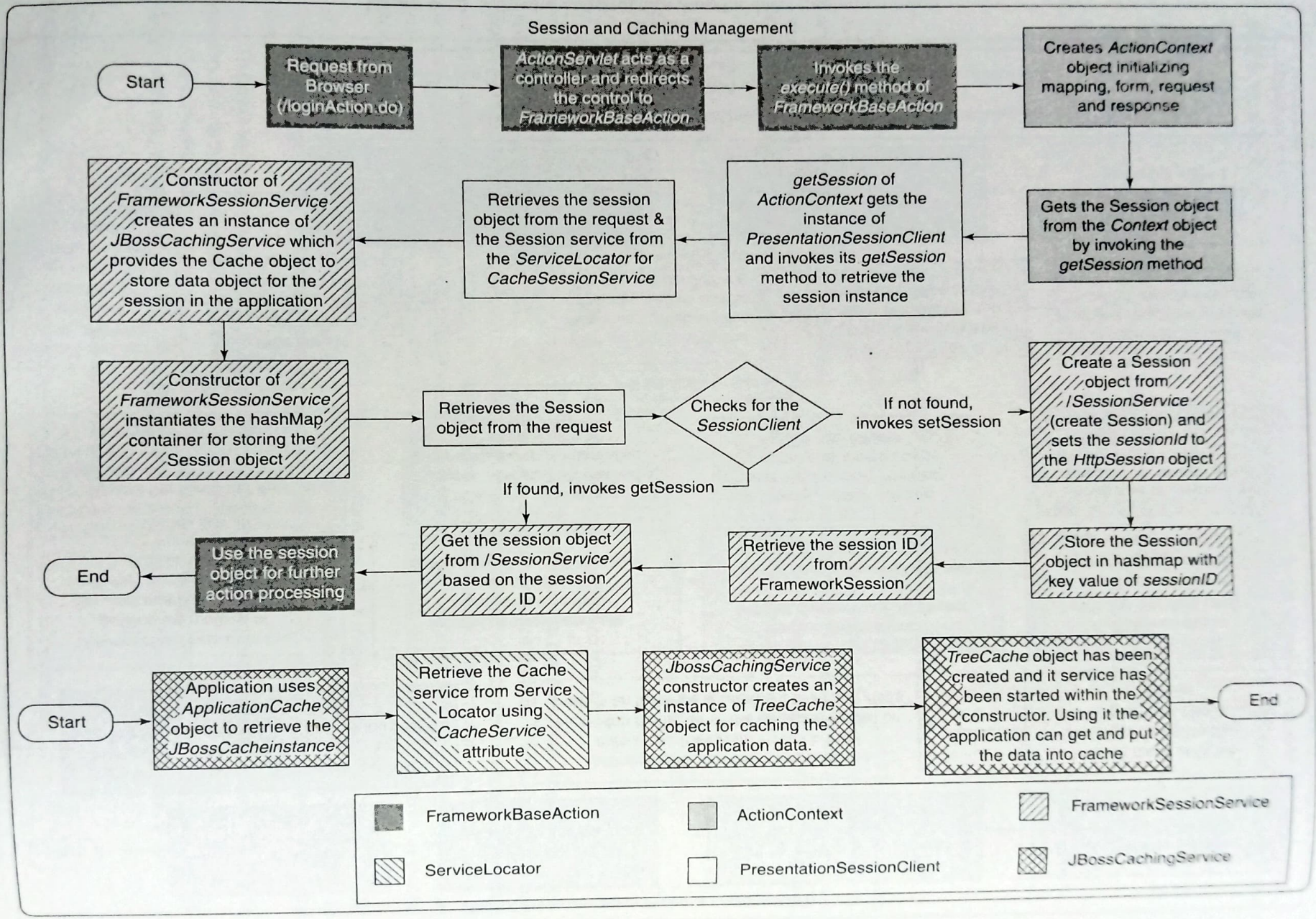


Figure 4-11 Software construction map for session and caching management.

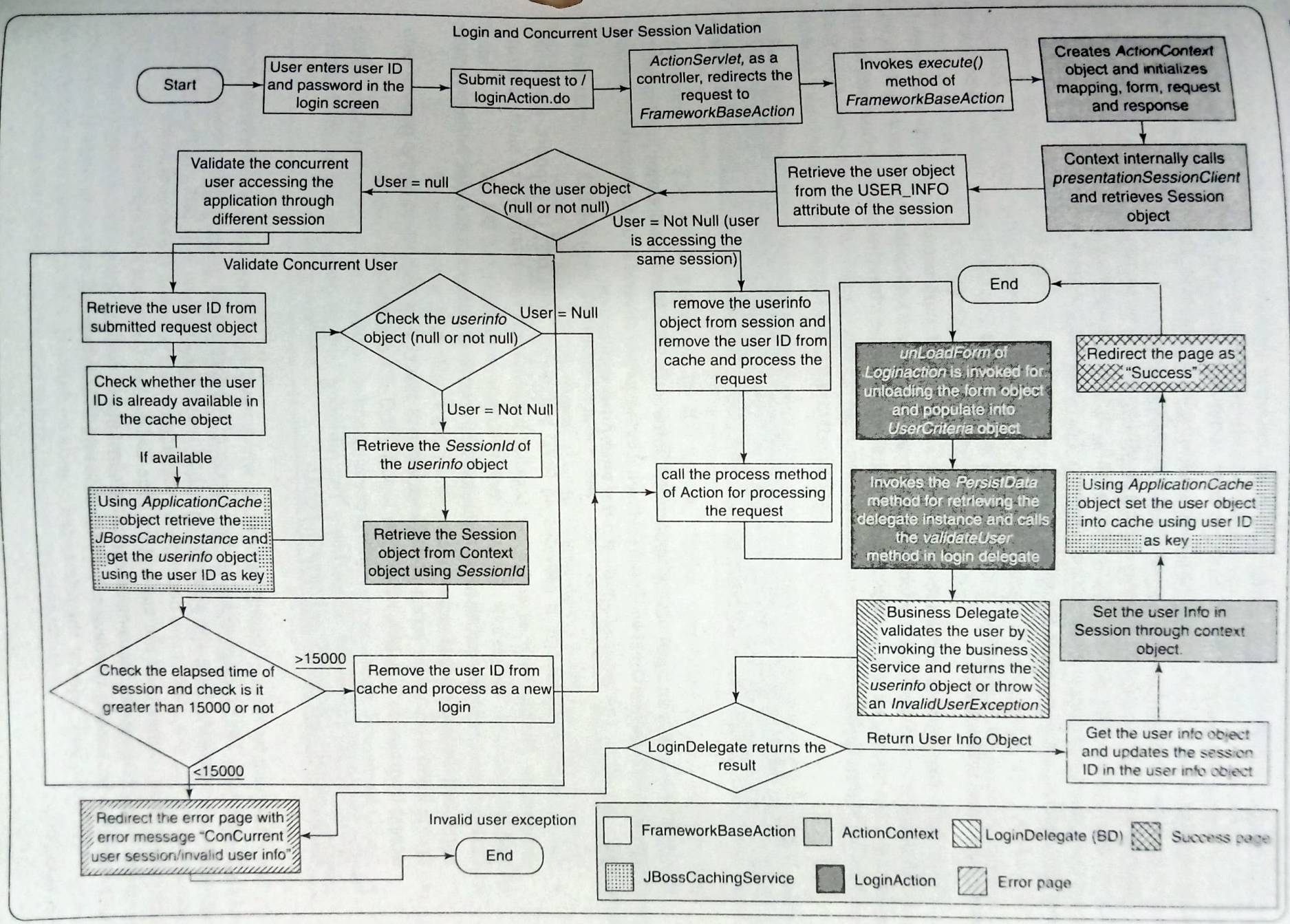


Figure 4-12 Software construction map for concurrent session management.

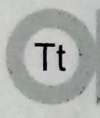
4.3.3 Business Layer Components

Like other layers, the construction of business layer components can also be done in several ways. Implementation using POJO is among the simplistic of the options available. The other usual choices for implementation are extending readily available frameworks.

Design team has to consider several aspects to ensure the construction of a robust and flexible business layer such as applying the right design patterns, designing optimized and appropriate transaction management and concurrency control mechanisms, designing business workflows and business rules repositories, and many more things.

Frameworks and technologies

As discussed in the previous section, enterprise application developers usually prefer to use frameworks to develop application layer components to leverage framework's easy-to-use and out-of-the-box capabilities. The developer community has multiple choices to select from the available business layer frameworks/technologies.



There are two strong players in the area of developing business layer components: Spring framework and EJB 3.0.

Spring framework is an open-source application framework. This framework is not just limited to the design and implementation of the business layer. It comprises of several sub frameworks to provide various services across the layers of a typical *n*-tier enterprise application:

- For Web/presentation tier, it provides an MVC framework with rich UI support. It extends support for JSP tag libraries and other popular presentation tier frameworks such as Struts and JSF.
- Spring framework's transaction management framework provides support for both programmatic and declarative transaction management mechanisms.
- Spring framework's data access framework extends support for almost all of the popular data access frameworks including Hibernate, TopLink, JDO, JPA and iBatis.
- Spring framework also supports batch processing and remoting by using spring batch and spring remote access framework, respectively. The remoting feature of Spring framework supports technologies such as RMI, RMI-IIOP and SOAP.

Two important features of Spring framework, which need special mention, are *inversion of control* (IoC) and AOP. The concept of IoC is the reverse of the traditional programming model, where application components have the control, and they maintain the flow and dependencies among themselves. In IoC, the control and dependency management is with the framework, which calls the application code, rather than the application code calling the framework components. It is something like doing class wiring by using configuration, instead of coding. The configuration files contain the dependencies among the Java objects. The Spring framework IoC container uses this dependency information and "injects" the referenced objects on demand, and is referred to as *dependency injection* (DI).

Spring framework's AOP framework provides the basic support for aggregating in one place the cross-cutting concerns of an enterprise application by using aspects. Other AOP frameworks, such as AspectJ, are relatively more comprehensive.[4]

EJB 3.0 API is one of the core APIs of the Java Enterprise platform meant for distributed computing. The basic idea of using EJB is to simplify the development process by providing services for scalability, security, life cycle management and transaction management to the application so that its focus can be limited to the business logic. The previous versions of EJB were complex, but EJB 3.0 has been introduced as a lean and simpler API. The introduction to metadata annotations has significantly reduced complexity by removing the need for creating deployment descriptors, home interfaces and remote/local object interfaces, which are now generated by the EJB container itself. Classes and interfaces of EJB are now created using POJO and plain old Java interface (POJI), and use the dependency injection mechanism through which the container makes the bean instances.[5]

Both EJB and Spring framework have their own pros and cons. But the comparison between the two is always a moving target, as they are getting simpler and richer version to version. They are not mutually exclusive, however, and can be combined to get the best of both the worlds. In the LoMS application framework, both EJB 3.0 and Spring framework are used in an integrated manner. Table 4-3 provides a comparative analysis of EJB 3.0 and the Spring framework.

Table 4-3 EJB 3.0 vs. Spring framework

Features	EJB 3.0	Spring framework
Acceptance	EJB 3.0 is governed by JCP and is specified in JSR 220 as Java EE standard	Spring framework is an open-source framework, and is not a Java standard. It is a popular alternative to EJB 3.0
Transaction support	EJB 3.0 uses transaction support provided by the EJB container. It only supports JTA transactions	Spring framework supports transactions using Spring AOP's transaction aspects
State management	An EJB 3.0 application maintains state by using a Stateful Session EJB. Stateful Session EJB is one of three different kinds of EJBs in the JEE technology stack	Spring framework maintains state by using non-singleton (or *prototype* beans) and scoped beans
Dependency injection	EJB 3.0 has primitive support for dependency injection	Spring framework has exhaustive support for several types of dependency injections

Package structure

In LoMS, both the Spring framework and EJB components are extended to create a custom application framework for laying the foundation of business layer components. Application specific components

[4] For further information on Spring framework and AspectJ, you can refer to the reference documentation on http://static.springframework.org/spring/docs/2.0.x/reference/index.html and http://www.eclipse.org/aspectj/, respectively.

[5] For further information on EJB, you may refer to http://java.sun.com/products/ejb/.

are created on top of these components. For understanding application-specific components, excerpts from the "Initiate Loan" use case are provided below. To start with, let us look at Figure 4-13, showing the package structure of the business layer components.

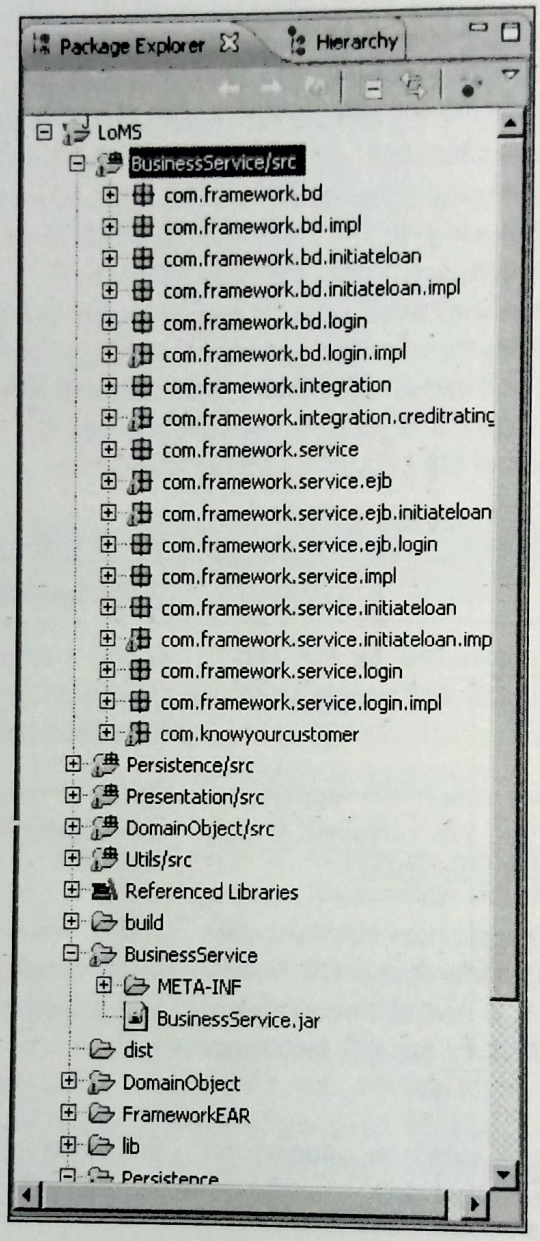


Figure 4-13 Package structure of business layer.

Application framework components

In LoMS, the business layer application framework extends EJB and Spring framework components along with several other Java components. Figure 4-14 depicts the static view of the business layer related application framework components. In this example, the business layer framework of the application comprises of Business Delegate components, session façade components, business service components and business model components.

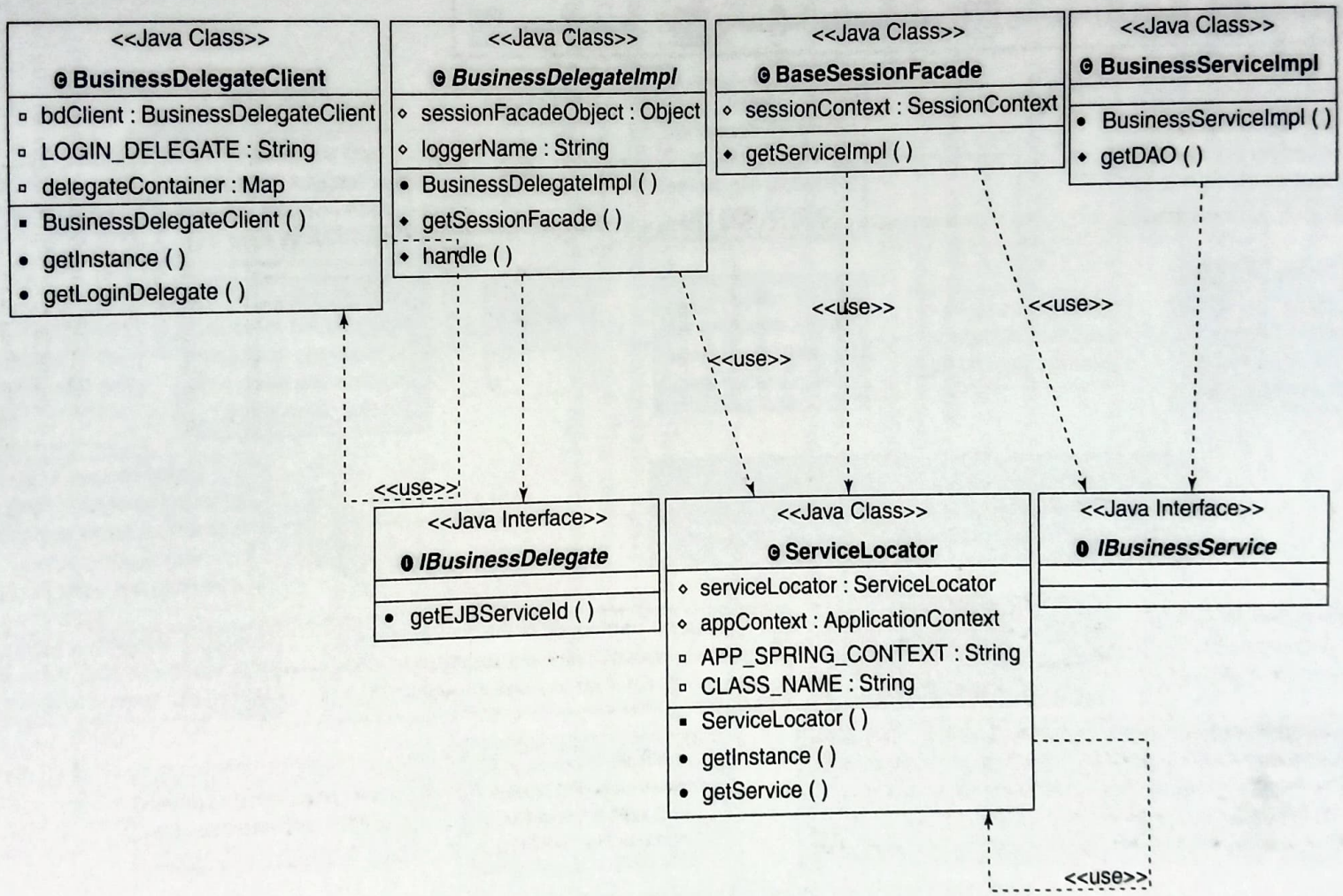


Figure 4-14 Business layer—application framework components.

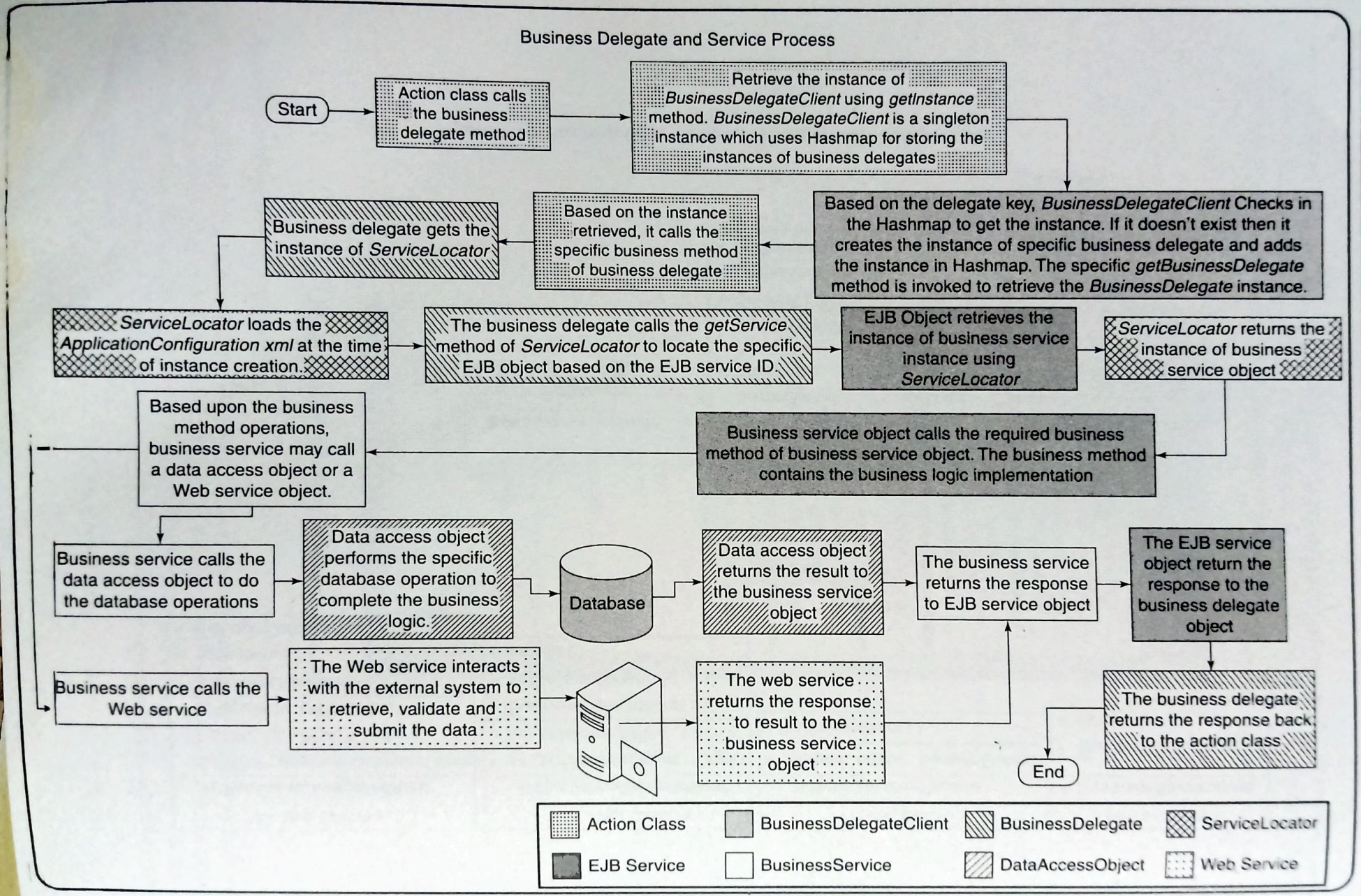


Figure 4-15 Software construction map for Business Delegate and Service Process.

There are three elements of Business Delegate components:

- `IBusinessDelegate` is the business delegate interface. All business delegate interfaces extend this base interface. This interface has `getEJBServiceId` method to return the EJB Service ID name.
- `BusinessDelegateImpl` is an implementation of `IBusinessDelegate`. All business delegate classes extend this base delegate. It has the implementation of `getSessionFacade` method to get the session bean object through `ServiceLocator`.
- `BusinessDelegateClient` class is used by the presentation layer to get the Business Delegate objects, which in turn are used to invoke business methods.

There are two elements of Business Service components:

- `IBusinessService` is the business service interface. All the framework application business service interfaces extend this interface.
- `BusinessServiceImpl` is the implementation of the `IBusinessService` interface. It contains `getDAO` method implementation to get the specific data access object.

Figure 4-15 depicts the big picture of the business delegate and the service process using a software construction map. There are three important elements of Business Model components, as shown in Figure 4-16:

- `IBusinessModel` is the business model interface. This interface extends the serializable interface.
- `BusinessModel` is an implementation of the business model interface. All the framework application business model implementation classes extend this class.
- `ICriteria` is the criteria interface. The objects that implement this interface act as a request object containing the request parameter information to perform business operations.

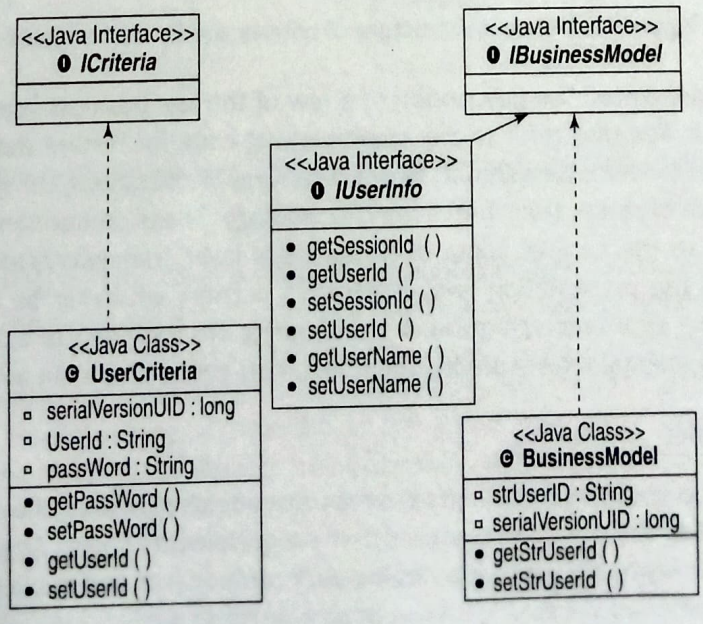


Figure 4-16 Business layer—business model application framework components.

The package structure of the criteria and domain objects is shown in Figure 4-17.

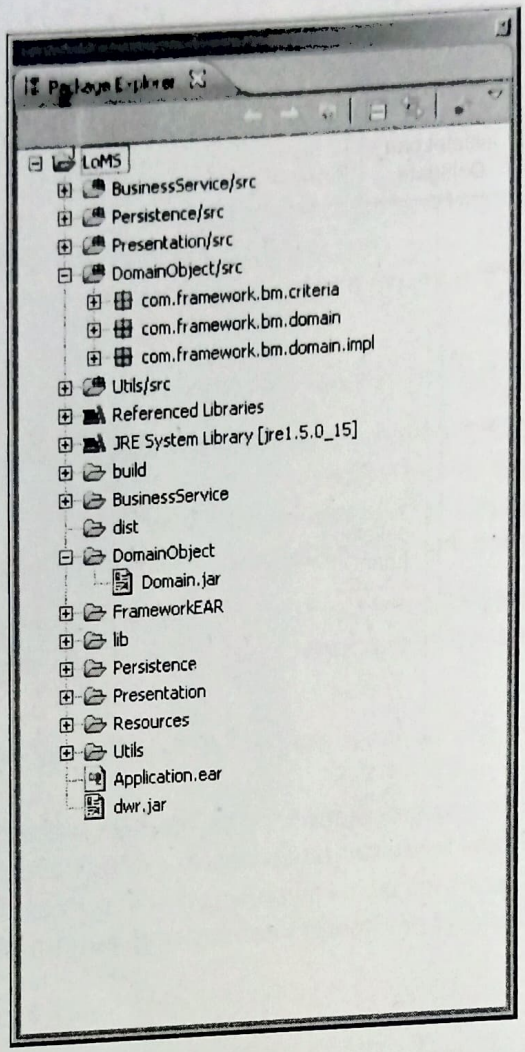


Figure 4-17 Package structure of criteria and domain objects.

This section has presented the description of a few of the key business layer components of the application framework. You may refer to the accompanying code for further details about the implementation. All these framework components of business layer are designed, considering EJB and Spring framework as the base business layer framework/technology. These components may not hold well, partly or completely, in the case of some other business layer framework/technology being used. As explored before in the presentation layer framework section, whatever be the choice of a layer-specific framework, the application framework components are designed to achieve reusability, simplicity, robustness and maintainability among other desirable properties in an application framework.

Application components

Application components in the business layer are the components that form the core of an application, and can be traced back to the use cases captured in the requirements phase. As you already know, as is the case with any other layer, the application framework components and application components are stitched together to construct the business layer of an enterprise application.

Let us further consider the "initiate loan" use case to understand how the business layer application components work together. The sequence diagram, as in Figure 4-18, represents the

sequence of interactions among the components that implements the "initiate loan" use case in the business layer.

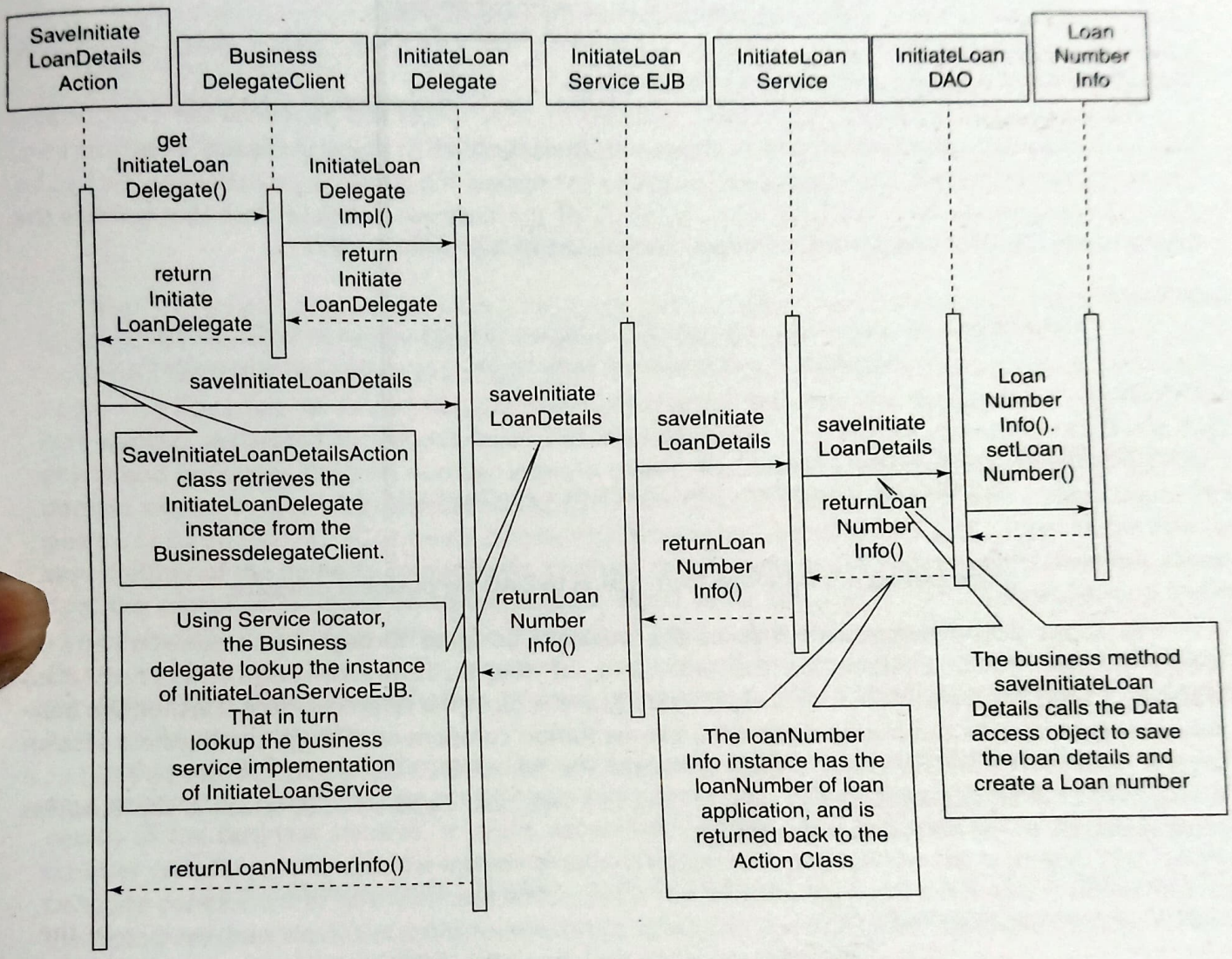


Figure 4-18 Initiate loan service—submit and save loan initiation.

The `InitiationForm.jsp` raises a request for the action path `initiationAction` when the form is submitted. The `RequestProcessor` of the Struts framework instantiates and populates the `com.framework.web.form.initiateloan.InitiateLoanForm`, and invokes the `execute(ActionMappingmapping, ActionFormform, HttpServletRequest request, HttpServletResponse response)` method of the `SaveInitiateLoanDetailsAction` action class. The method instantiates the `ActionContext` and retrieves the session. The method invokes the `isUpdateRequired(context)` to verify whether the form has been modified, as shown in Code Listing 4.8.

```
if (isUpdateRequired(context)) {
   processUpdate(context);
}
```

Code Listing 4.8 Verifying the form for modifiability.

This validation ensures that no form is submitted more than once without modification. The `processUpdate(ActionContext context)` unloads the form by invoking `unloadForm(ActionContext context)`, and invokes the `persistData(ActionContext context)` to persist the data in the database by invoking the data access object (DAO) components through business delegate and business session façade.

The `unloadForm(ActionContext context)` instantiates and populates the `InitiateLoanInfo` domain object from the form object and stores the domain object in the `ActionContext`. The `persistData(ActionContext context)` invokes the `saveInitiateLoanDetails(InitiateLoanInfo initiateLoanInfo)` of the business delegate class that persists the data through the DAO component, as shown in Code Listing 4.9.

```
try {
      loanNumber = context.getSessionClient().getDelegateClient()
                  .getInitiateLoanDelegate().saveInitiateLoanDetails
(info);
   }
 catch (BusinessException ex) {
            addError(context, ex.getMessageTextId());
  }
```

Code Listing 4.9 Dataflow from action to DAO via business delegate.

The Struts action component invokes the business delegate through `BusinessDelegateClient`. The `getInitiateLoanDelegate()` of the `BusinessDelegateClient` class instantiates the `InitiateLoanDelegateImpl`, and returns the reference to it. Further, the business delegate invokes the business service implementation component through the business session façade. `InitiateLoanDelegateImpl` invokes the `saveInitiateLoanDetails(InitiateLoanInfo initiateLoanInfo)` of `InitiateLoanServiceEJB`, which is the business session façade, as shown in Code Listing 4.10.

```
IInitiateLoanEJBLocal initiateLoanEJB = (IInitiateLoanEJBLocal)
super.sessionFacadeObject;
```

Code Listing 4.10 Business delegate implementation.

As shown in Code Listing 4.11, the `sessionFacadeObject` is initialized to the `InitiateLoanServiceEJB` business session façade, as `getEJBServiceId()` returns `InitiateLoanServiceEJB`.

```
sessionFacadeObject =
ServiceLocator.getInstance().getService(getEJBServiceId());
```

Code Listing 4.11 Business session façade.

The business session façade invokes the `saveInitiateLoanDetails(InitiateLoanInfo criteria)` of `InitiateLoanServiceImpl`, which is the business service implementation component for loan initiation, as shown in Code Listing 4.12.

```
IInitiateLoanService initiateLoanService = (IInitiateLoanService)
super.getServiceImpl(serviceID);
    return initiateLoanService.saveInitiateLoanDetails(criteria);
```

Code Listing 4.12 Session façade invocation.

The `super.getServiceImpl(serviceID)` instantiates the `InitiateLoanServiceImpl` business service implementation component, as shown in Code Listing 4.13.

```
businessService = (IBusinessService)
ServiceLocator.getInstance().getService(serviceID);
```

Code Listing 4.13 Business service component initiation.

The `saveInitiateLoanDeatils(InitiateLoanInfo criteria)` instantiates the `InitiateLoanDAO` through service locator, and invokes the `saveInitiateLoanDetails(criteria)` of `InitiateLoanDAO`. The `saveInitiateLoanDetails(InitiateLoanInfo criteria)` of `InitiateLoanDAO` persists the data into the database and retrieves the loan number generated by the database. The method instantiates the `LoanNumberInfo` domain object and populates the loan number into the object. The method returns the `LoanNumberInfo` domain object to the `persistData(ActionContext context)` of the Action class. The `persistData(ActionContext context)` stores the `LoanNumberInfo` into the session. In case if either of the delegate components, business components or DAO components raise any exception, the exception is added on to the request object using the `addError` method defined in the `FrameworkBaseAction` class. The `execute(ActionMapping mapping, ActionForm form, HttpServletRequest request, HttpServletResponse response)` method of the Action class determines whether an error is associated. The method returns the corresponding `ActionForward` object.

In the overall flow of the business layer components, readers have noticed that without the business delegate, presentation-tier components have direct exposure to the underlying implementation details of the business services. It might necessitate changing the implementation of the presentation-tier components in case the implementations of the business services get changed. The Business Delegate component in turn interacts with service components using service locator, which abstracts the lookup mechanism. In the LoMS framework, a service locator is used for all kinds of look up services, e.g. business service lookup, EJB lookup and data access components lookup.

In case where the Spring framework is used, the IoC container of framework takes care of providing all the dependencies for a given components, and the nitty-gritty of locating and instantiating the dependencies are transparently managed by the container. LoMS leverages these features of the Spring framework in its implementation.

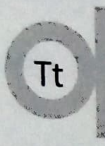
4.3.4 Data Access Layer Components

Like other layers of an enterprise application, construction of the data access layer components can be done in several ways. Developers have multiple choices ranging from POJOs to the available persistence layer frameworks/technologies. In an enterprise application, the data access layer should be designed in a manner that it is decoupled not only from the business components but also from the underlying relational tables or any other data source.

Frameworks and technologies

The simplest of the data access layer technologies is *Java Database Connectivity* (JDBC), wherein SQL queries are directly embedded in Java code. In such a scenario, there is a tight coupling between Java objects and the underlying relational data stores. The concept of *object relation mapping* (ORM) has come into practice to avoid such coupling. ORM provides a bridge between the Java object world and the data store's relational world by providing mappings between the two through the XML configuration files or annotations.

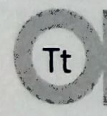
The ORM approach enables developers to think in terms of objects rather than tables or relations, when it comes to persistence of objects. This enables developers to apply object-oriented best practices in implementing a data access layer.



Hibernate, Java Persistence API (JPA), EJB, TopLink and iBatis are some of the popular frameworks/technologies, which provide ORM based implementation of the data access layer.

Hibernate framework is an open-source ORM framework that allows developers to implement data access layer classes using object oriented features such as polymorphism, inheritance and encapsulation. Hibernate also supports Hibernate Query Language (HQL), which is like a primitive OO extension to the SQL. It also provides native SQL support. Hibernate framework is currently the most popular ORM framework for Java based persistence solutions.

JPA is a specification, which has been introduced as part of Java EE 5 platform, standardizes the way persistence layers are developed. JPA has Java persistence query language (JPQL) with which developers can query the persistent entity objects.



The Spring framework provides Spring DAOs and Hibernate DAOs to build the data access layer.

Let us now explore a few differences between JPA and other data access layer frameworks, as presented in Table 4-4. Some best practices, which are required to construct a robust and scalable data access layer, are as follows:

- **Identifying callbacks**. Every entity, which needs to be persisted, has a life cycle associated with it. The entity may undergo certain operations or manipulations during the life cycle. For example, a prerequisite for the phone number of a customer to be persisted is to take off the hyphens and the brackets from the phone number field. Persistence frameworks provide support to capture such manipulations via the "callback" methods. JPA and Hibernate framework provide them as annotations.
- **Identifying embedded objects**. Identification of embedded entity objects can result in improving reusability. For example, an address entity can be embedded as customer's communication address, loan guarantor's address, customer's permanent address, etc. JPA has @Embedded annotation to do this.

Package structure

In LoMS, POJO based implementation of the DAO pattern is used to design both the framework components and the application components of the data access layer. Readers have already seen the application specific components of the data access layer in the previous section, where excerpts from the "Initiate Loan" use case were discussed.

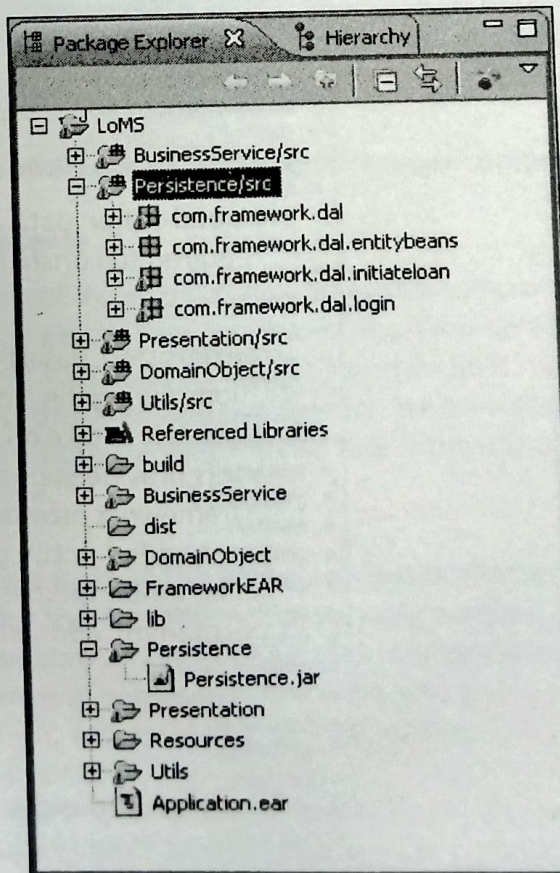
Table 4-4 JPA vs. other data access layer frameworks

Features	JPA	Other data access layer frameworks
Acceptance	JPA is a Java standard specification (JSR 220) and has been introduced in Java EE 5 platform	Other popular data access layer frameworks are Hibernate, iBatis and Oracle TopLink
Control on queries	Control on queries is limited as it generates the query automatically	A strong control over queries is possible as the onus is on the developer to map Java objects and queries
Applicability	Usage of JPA is preferred in applications, where control on queries including their fine tuning is not the main criteria	Frameworks such as iBatis are used in database centric applications where tight control on queries is required

In this section, readers will see the package structure of the data access layer used in LoMS, and later explore the application framework components specific to the data access layer. Figure 4-19 depicts the package structure of the data access layer components.

Application framework components

In LoMS, the data access layer framework is designed using POJOs. Figure 4-20 shows the static view of the framework components of the data access layer.



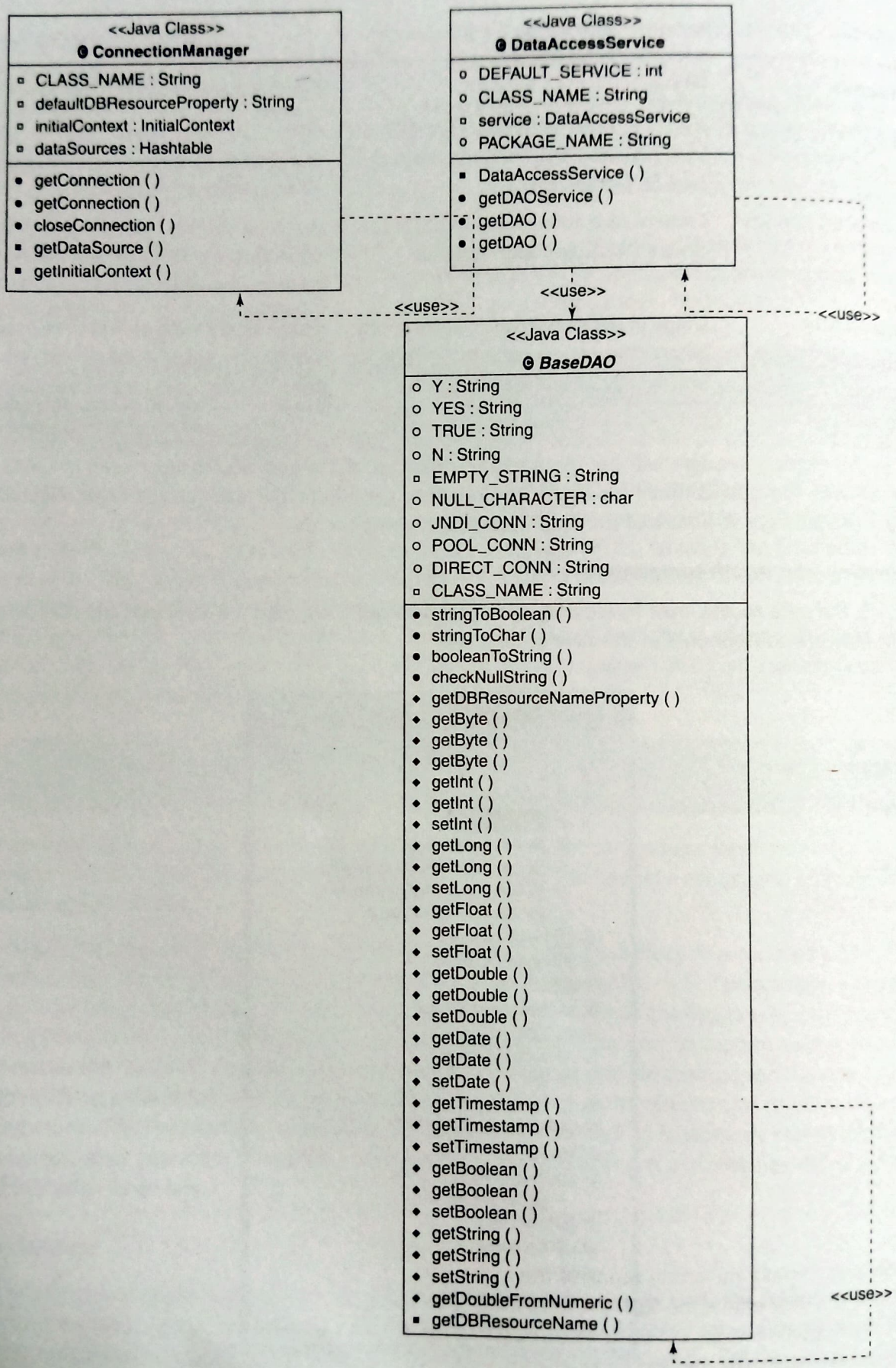


Figure 4-20 Data access layer—application framework components.

The DAO implementation encapsulates the mechanisms used to access the underlying data stores. If the data source is a relational database, as in LoMS, simple JDBC, JPA or other ORM frameworks can be used to implement the mechanism to access data stores.

The DAO implementation may also encapsulate a service that integrates with another application to fetch data required by the enterprise application. In such a case, either a low-level socket based mechanism can be used or a high-level API (such as RMI or JMS) can be used to implement the underlying access mechanism.

4.3.5 Integration Layer Components

Construction of integration layer components can be done in a variety of ways, and at different levels (of application layers), as discussed in Chapter 3, where you have learnt about several integration mechanisms and technologies along with the design elements that facilitate the integration of applications by realizing a robust integration layer. In this section, you will go through an integration scenario and understand how the application framework components and application components work together to implement it.

The sample integration scenario is as follows: LoMS has to determine the preapproved amount for a loan applicant as part of the "Loan Initiation" use case. The preapproved amount is determined based on the credit rating of the customer. An external credit rating agency provides the credit rating of an individual based on their transaction history and liabilities. LoMS has to access the KnowYourCustomer Web service hosted by this agency to retrieve the credit rating of the customer.

Integration layer of LoMS uses JAX-WS API to implement the SOAP-based Web services integration between LoMS and credit rating system. As readers have learnt in Chapter 3, the JAX-WS API is used to expose a Web service as a POJO interface. When the Web service consumer invokes the service method of the POJO interface, it automatically generates a SOAP request message to communicate with the provider.

Package structure

Figure 4-21 shows the package structure of the integration layer components, as used in LoMS.

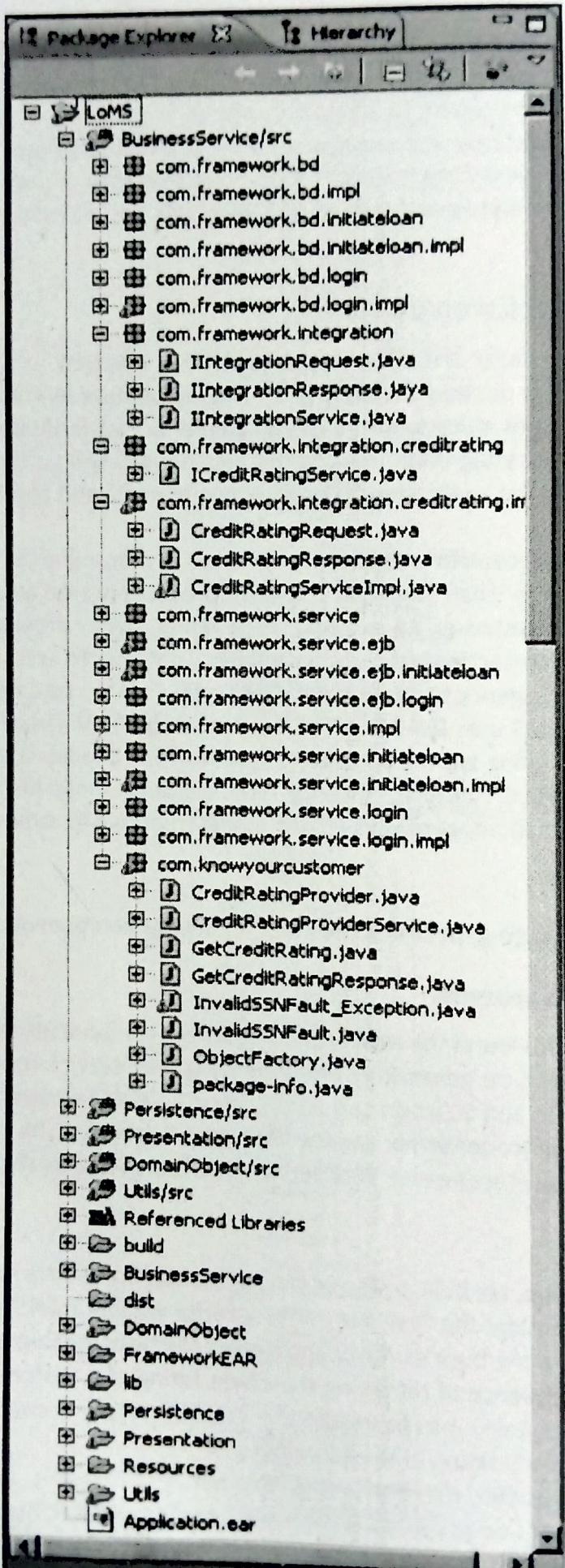
Application framework components

Figure 4-22 shows the static view of the application framework components related to integration layer. The `IIntegrationService` generalizes the underlying Web service consumers' implementation. `IIntegrationRequest` and `IIntegrationResponse` hold the parameters that get communicated between the Web service consumer and the Web service provider. The consumer uses the JAX-WS API to invoke the `KnowYourCustomer` Web service that is provided by the credit rating agency.

Application components

Integration layer components are built on top of the application framework components specific to the integration layer. Let us consider the "validate social security number (SSN)" scenario in "initiate loan" use case to understand how the business layer application component integrates with the Web service. Figure 4-23 depicts the sequence of retrieving the credit rating of a customer based on his SSN. The `InitiateLoanServiceImpl` has to invoke the `KnowYourCustomer` Web service to validate the SSN and retrieve the credit rating of the customer.

Once the user enters the SSN, the form invokes the `com.framework.web.action.intiateloan.ValidateInitiationAction` class through the AJAX processor. The `ValidateInitiationAction` class, through the `InitiateLoanDelegate` and `InitiateLoanServiceEJB`, invokes the `validateSSN(InitiateLoanInfo initiateLoanInfo)` of `InitiateLoanService`.


Package Explorer
Hierarchy
LoMS
BusinessService/src
com.framework.bd
com.framework.bd.impl
com.framework.bd.initiateloan
com.framework.bd.initiateloan.impl
com.framework.bd.login
com.framework.bd.login.impl
com.framework.integration
IIntegrationRequest.java
IIntegrationResponse.java
IIntegrationService.java
com.framework.integration.creditrating
ICreditRatingService.java
com.framework.integration.creditrating.in
CreditRatingRequest.java
CreditRatingResponse.java
CreditRatingServiceImpl.java
com.framework.service
com.framework.service.ejb
com.framework.service.ejb.initiateloan
com.framework.service.ejb.login
com.framework.service.impl
com.framework.service.initiateloan
com.framework.service.initiateloan.impl
com.framework.service.login
com.framework.service.login.impl
com.knowyourcustomer
CreditRatingProvider.java
CreditRatingProviderService.java
GetCreditRating.java
GetCreditRatingResponse.java
InvalidSSNFault_Exception.java
InvalidSSNFault.java
ObjectFactory.java
package-info.java
Persistence/src
Presentation/src
DomainObject/src
Utils/src
Referenced Libraries
build
BusinessService
dist
DomainObject
FrameworkEAR
lib
Persistence
Presentation
Resources
Utils
Application.ear

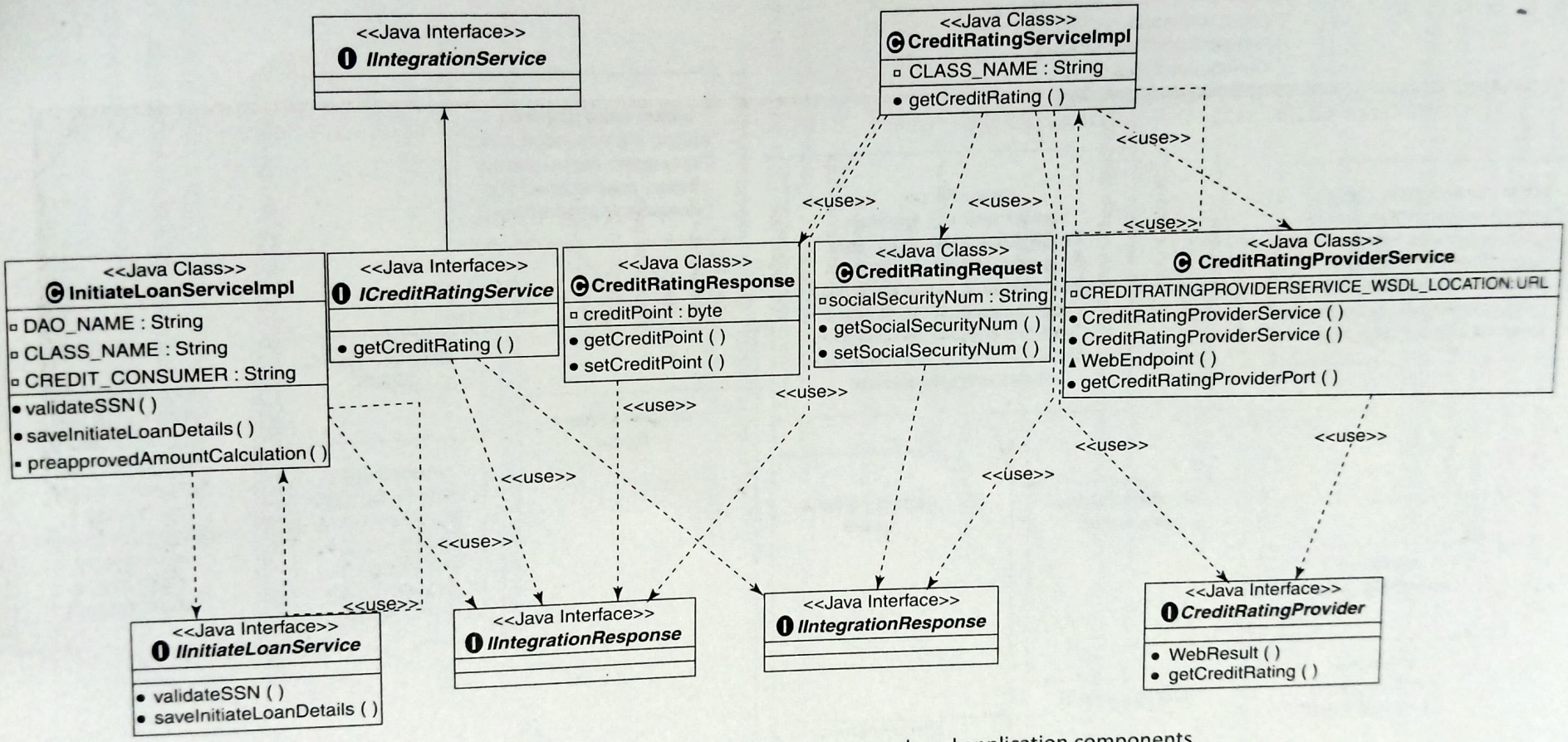


Figure 4-22 Integration layer—application framework and application components.

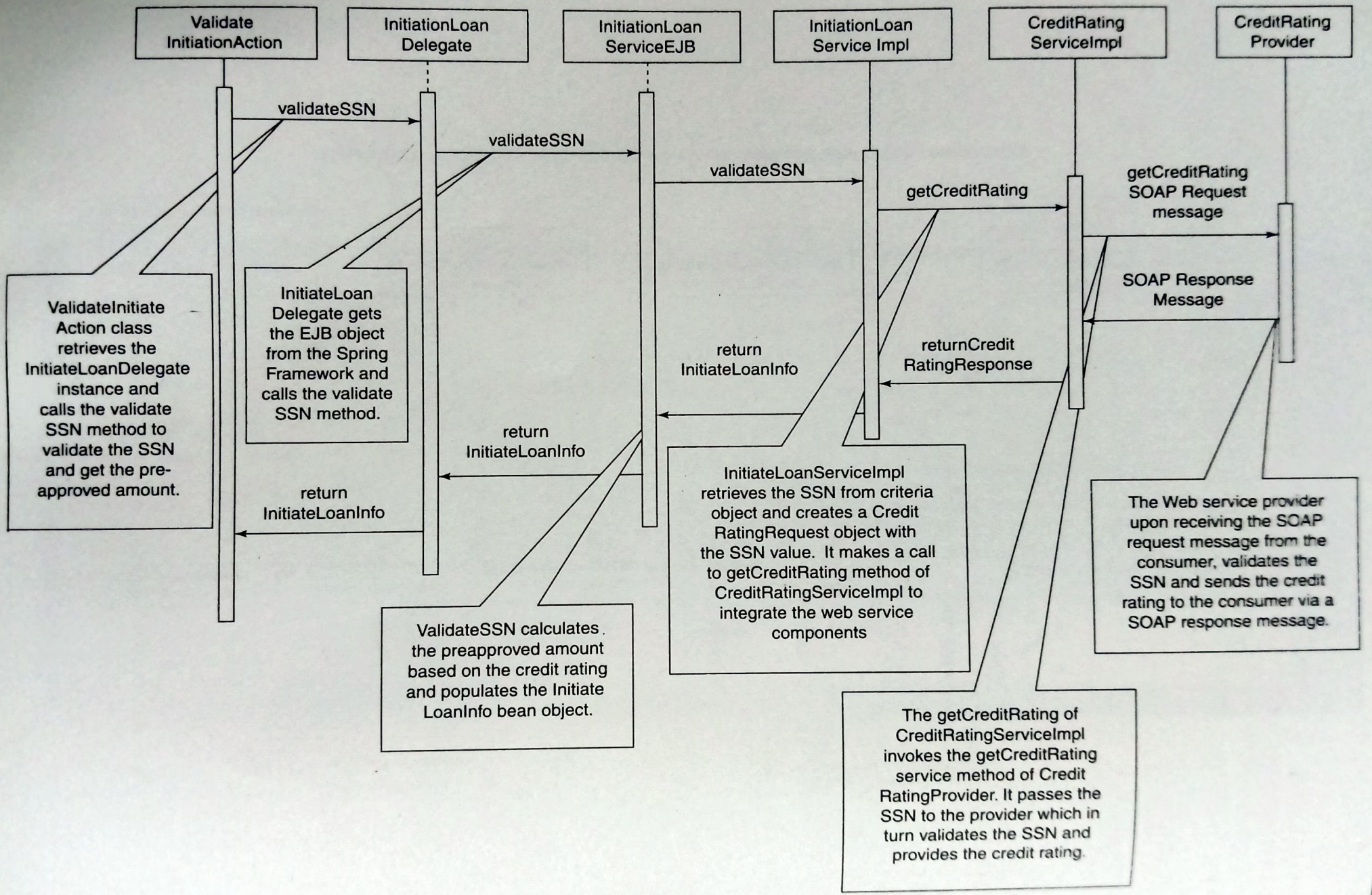


Figure 4-23 Retrieving the credit rating using Web services.

The `InitiateLoanServiceImpl` business service implementation class invokes the Web service consumer component `CreditRatingServiceImpl` to validate the SSN and retrieve the credit rating of the customer. The `validateSSN(InitiateLoanInfo criteria)` of `InitiateLoanServiceImpl` class instantiates the `CreditRatingRequest` object, which has the properties (the applicant's SSN, in this case) to be sent to the Web service consumer, as shown in Code Listing 4.14.

```
InitiateLoanInfo loanInfo = criteria;
CreditRatingRequest request = new CreditRatingRequest();
request.setSocialSecurityNum(loanInfo.getSsn());
```

Code Listing 4.14 Request for SSN.

The method then invokes the `getCreditRating` method of the `CreditRatingServiceImpl` Web consumer class passing the "request" as a parameter. The code snippet for the same is shown in Code Listing 4.15.

```
ICreditRatingService consumer = null;
      try {
            consumer = (CreditRatingServiceImpl)
            ServiceLocator.getInstance().
getService(CREDIT_CONSUMER);
      } catch(ServiceNotAvailableException e) {
            logMessage = new
LogMessage(LoggingManager.Error,CLASS_NAME,"InitiateLoanInfo","Credit
Rating Consumer Exception"+e);
            LoggingManager.log(logMessage);
            throw new BaseRuntimeException(e);
         }

      ...
         IIntegrationResponse response = null;
         try {
         response = consumer.getCreditRating(request);
         catch(InvalidSSNException ex){
         logMessage .setMessage("Exception Occurred due to Invalid SSN"+ex);
         LoggingManager.log(logMessage);
         throw ex;
         }
```

Code Listing 4.15 Passing the request as a parameter.

The `getCreditRating(IIntegrationRequest request)` of `CreditRating ServiceImpl` class retrieves the SSN, as shown in Code Listing 4.16.

```
CreditRatingRequest creditRequest = (CreditRatingRequest)request;
String ssn = creditRequest.getSocialSecurityNum();
```

Code Listing 4.16 Retrieval of SSN.

The `getCreditRating(IIntegrationRequest request)` invokes the Web service provider, as shown in Code Listing 4.17.

```
com.knowyourcustomer.CreditRatingProviderService service = new
      com.knowyourcustomer.CreditRatingProviderService();
com.knowyourcustomer.CreditRatingProvider port =
      service.getCreditRatingProviderPort();
```

Code Listing 4.17 Invoking the Web service provider.

The `getCreditRating (IIntegrationRequest request)` invokes the `getCredit Rating(String ssn)` Web method of the service provider. The provider returns the credit rating for the given SSN. If the SSN is found to be invalid, the provider throws the `InvalidSSNFault_ Exception` to the consumer. The `getCreditRating(IIntegrationRequest request)` method handles the exception and throws the `InvalidSSNException`. The code snippet for the same is shown in Code Listing 4.18.

```
try{
    creditRate = port.getCreditRating(ssn);
}
catch(InvalidSSNFault_Exception e){
InvalidSSNFault fault = e.getFaultInfo();
throw new InvalidSSNException(CLASS_NAME, "validateSSN", fault.
getErrorDescription(), null);
}
```

Code Listing 4.18 Verification of SSN.

The `getCreditRating(IIntegrationRequest request)` of `CreditRating ServiceImpl` populates the credit rate to the `CreditRatingResponse` and returns the same to the business service implementation class. The code snippet for the same is shown in Code Listing 4.19.

```
CreditRatingResponse creditResponse = new CreditRatingResponse();
creditResponse. setCreditPoint (creditRate);
return creditResponse;
```

Code Listing 4.19 Populating the response object.

The `validateSSN(InitiateLoanInfo criteria)` of `InitiateLoanServiceImpl` business service implementation class handles the response and retrieves the credit rating. The method invokes the private `preapprovedAmountCalculation(creditRate)` to determine the amount the customer is eligible for, as depicted in Code Listing 4.20.

```
CreditRatingResponse creditResponse = (CreditRatingResponse)response;
```

```
double preapprovedAmount = preapprovedAmountCalculation(creditResponse.
  getCreditPoint ());
loanInfo.setPreapprovedamount(preapprovedAmount);
```

Code Listing 4.20 Retrieval of pre-approved amount.

The method then populates the preapproved loan amount in `loanInfo` and returns the same which will be rendered in the `InitiateForm.jsp`.

So far, we have explored the construction in a layer-by-layer manner—both application framework components and the application components. In the course of this discussion, you have noticed that how an individual component interacts with other components—within the layer and outside the layer. You have also noticed that how these interactions—both static and dynamic—are captured using class diagrams, sequence diagrams and software construction maps.

As mentioned earlier, coding and integrating these units of code to build the system is the core of the construction phase. Other key construction phase activities that we will explore in the following sections are code review, unit testing, build process, static code analysis and dynamic code analysis.

4.4 Code Review

The review process plays a pivotal role in measuring the degree of compliance with the specifications provided, and verifying the quality of artifacts produced during development. The code artifacts are the primary target of review in the construction phase.

4.4.1 Objectives

The objectives of code review are manifold—completeness, correctness, consistency, logic, maintainability, traceability and robustness of code, as represented in Figure 4-24.

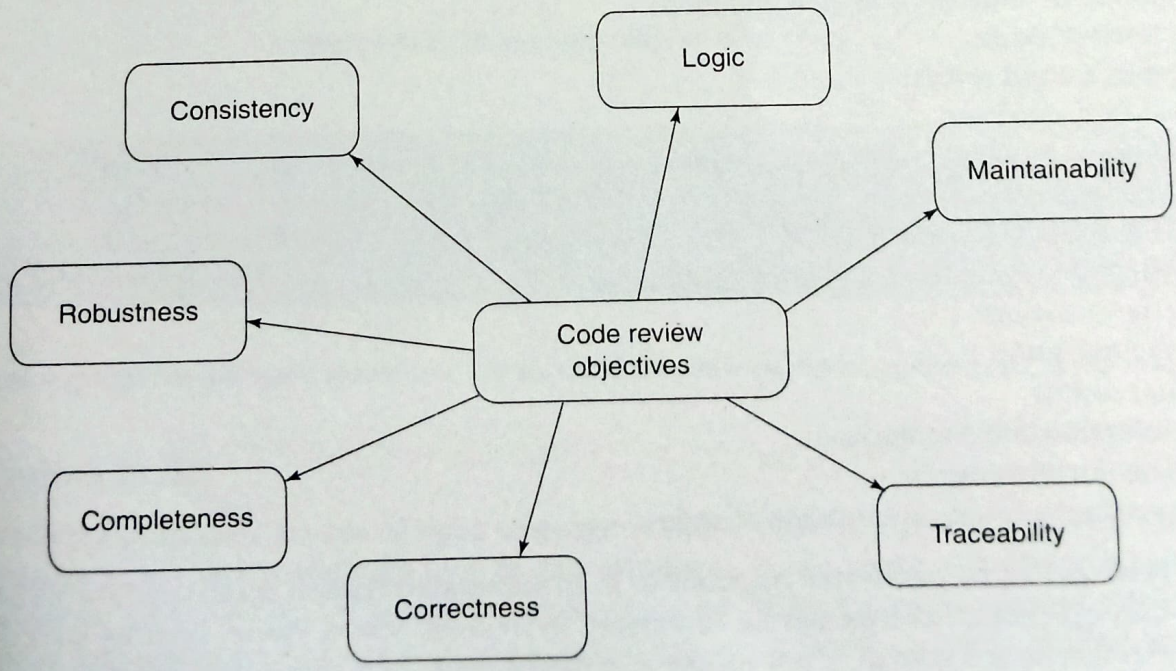


Figure 4-24 Code review objectives.

We outline these code review aspects below:

- *Completeness* review ensures that code is in line with the design of a software component and meets all key requirements.
- *Correctness* review typically vouches for conformance to coding best practices such as use of language-specific idioms, avoiding hard coding and eliminating unused variables.
- *Consistency* review ensures uniformity in coding, commenting, error and exception handling.
- Review of code from *logical* perspective is to ensure that the code results in expected behavior.
- *Maintainability* review is performed to ensure that the code is easily understood from the perspective of maintenance, i.e., readability of the code and is supported by adequate documentation. For example, use of descriptive identifiers enhances the readability significantly.
- *Traceability* review takes care of finding out whether any requisite functionality is missing, and whether the source code of the software unit can be traced back to its functional requirements and to corresponding design elements.
- Reviewing code from the *robustness* perspective is to ensure that the code is able to handle errors and unexpected events at runtime.

4.4.2 Process

Code review starts with identification of right set of reviewers and assigning ample time to perform it with the help of appropriate checklists and review guidelines. There are several things that a reviewer has to look for, which are captured in the review checklist. A few important ones are as follows:

- Code naming conventions
- Computational errors
- Comparison errors
- Control flow errors
- Infinite or improperly terminated loops
- Interface errors
- Input/output errors
- Unused variables
- Wrong initialization and default values
- Data reference errors
- Data declaration errors
- Unhandled exceptions and error conditions
- Error messages
- Help messages
- Hard coding
- Contention and deadlocks
- Inline documentation
- Beautification, e.g., indentation of code

Code reviews may be performed manually or in an automated fashion against certain set of guidelines and best practices. Reviews can be conducted in multiple forms. Review can be structured or unstructured, formal or informal, a self review or a peer review, or a code walkthrough driven by the developer or by the reviewer.

4.5 Static Code Analysis

Static code analysis is an activity of analyzing the code, in source or byte-code form, to identify various issues related to non-runtime aspects of the code. As depicted in Figure 4-25, static code analysis is a wide area that focuses on analysis of a codebase from several perspectives such as coding style, bug finding, finding security related vulnerabilities and code quality analysis. Static code analysis is an automated activity, which is performed using tools.

Tt

Open source tools, such as Findbugs, CheckStyle and PMD, are used for static code analysis. While these tools can also be configured through rules to identify certain types of security vulnerabilities, specialized tools such as Fortify 360 and CodeSecure are specifically intended for identifying a vast range of security vulnerabilities.

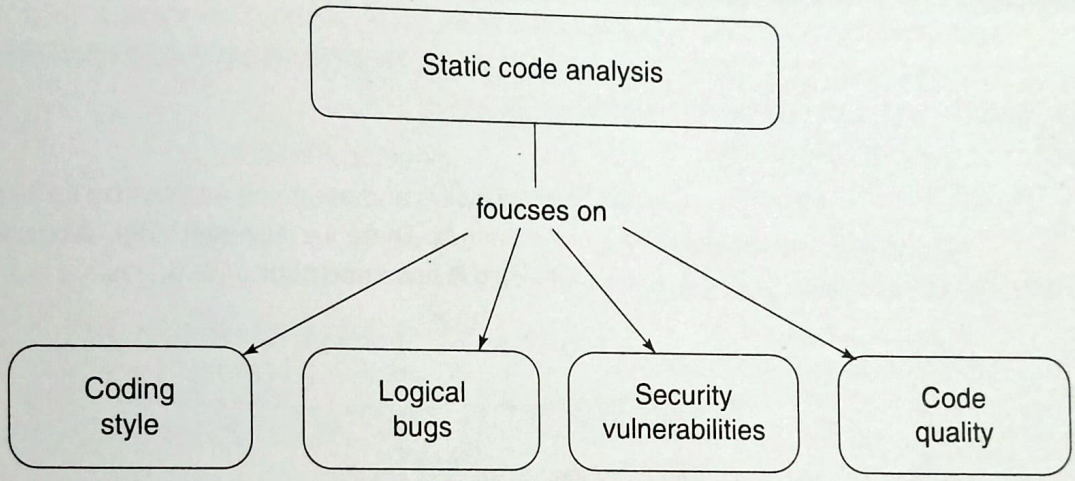


Figure 4-25 Static code analysis objectives.

In

Two keywords, *false positives* and *false negatives*, are usually encountered druing automated static code analysis. False positive is a reported problem by a tool that is actually not a problem. False negative represents a problem which actually exists, but is not reported by a tool.

Let us now explore the primary areas that are focused on while performing the static code analysis.

4.5.1 Coding Style

Coding style check focuses on use of right language features such as use of apt control constructs, appropriate language idioms and right type of data structures. It also checks whether the code is correctly formatted for readability. It reports inconsistencies based on the preconfigured rules for style specification. Inconsistencies may be related to naming the identifiers, improper use of whitespace or the structural aspects of the program. Rules related to style are typically identified in the very beginning of the construction phase, and are enforced with the help of tools.

4.5.2 Logical Bugs

While finding *logical bugs* is essentially the endeavor of manual code review activity, a small subset of bugs may also be identified through encoded rules verified as part of static code analysis. These include identification of defects such as not releasing resources when they are no longer required, not using `StringBuffer` for string concatenation and unhandled potential `NullPointer` exceptions.

4.5.3 Security Vulnerabilities

The growing security threats are making the Internet facing enterprise applications more and more vulnerable, and necessitate a dire need to ensure secure code. Security code review and code analysis from a security perspective both aim to unearth security vulnerabilities in code.

There are myriads of security vulnerabilities which give hackers a chance to exploit them to abuse enterprise resources. Figure 4-26 depicts an application security vulnerability cloud. OWASP, an open community which focuses on improving the security of Web application software, has defined the top 10 security vulnerabilities that are prevalent in Web applications.[6]

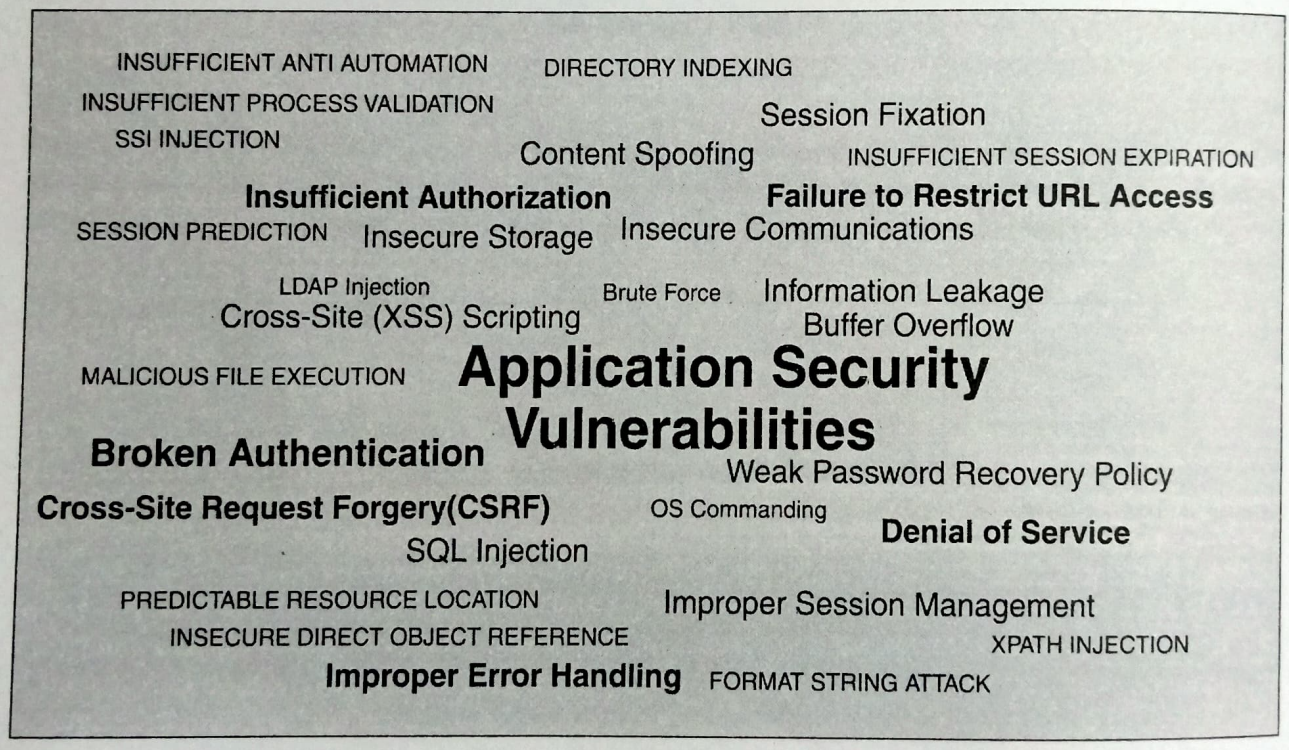


Figure 4-26 Potential security vulnerabilities in an enterprise application.

Security is an essential component of enterprise applications, and starts from its inception. Developers and reviewers also have to follow various secure coding practices to ensure the development of secure code. Let us now look at a few of the vulnerabilities and secure coding practices to avoid them:

- **Cross-site scripting.** Popularly known as XSS, it is a vulnerability which allows an attacker to pass malicious scripts (potentially harmful Javascript code) through the form fields in

[6] You may refer to http://www.owasp.org/ to know more about OWASP and top security vulnerabilities.

the UI of an application. When a user requests for the HTML page containing the form, the values (malicious script) corresponding to these fields are passed back to the user's browser, the malicious script gets executed on the client machine. Code Listing 4.21 depicts an XSS prone code.

```
response.write(request.getParameter("customerName"));
```

Code Listing 4.21 XSS vulnerable code.

In the above code, the string value for the parameter customerName can potentially be replaced with a malicious script. A proper input validation should be done to prevent any malicious input from the front end.

To prevent propagation back to the user of malicious scripts that may have somehow made it into the application, the presentation logic on the server side that retrieves values of the form fields will scan for special characters such as "<" and ">" symbols, and replace them with their corresponding escape sequences "<" and ">". These characters are typically used in a HTML page to mark the beginning of a script. Typically, frameworks such as Struts and Spring have support for such encoding, which neutralize the potential malicious script. This technique is illustrated in Code Listing 4.22.

```
response.write( encoder(request.getParameter("customerName")));
```

Code Listing 4.22 Secure code—XSS fixed.

The method encoder takes care of reading the input string and converting the special characters to their corresponding escape sequences.

- **SQL injection.** *SQL injection* is one of the most dreaded application security vulnerabilities. This allows the attacker to change the logic of an underlying SQL query by injecting malicious SQL from the form fields of the front end of an application. This may result in unauthorized access or modification of precious data. To avoid SQL injection, input provided to form fields should be validated for size, type, format, range, etc. Use of bind variable also helps in avoiding this vulnerability. Code Listing 4.23 presents an example of code vulnerable to SQL injection.

```
PreparedStatement pstmt =
conn.prepareStatement("insert into CUSTOMER (CUSTNAME) values ('"
  +custname + "')");
```

Code Listing 4.23 Code vulnerable to SQL injection.

A hacker may insert malicious SQL code in the above code through the variable `custname`. This can be avoided by rewriting the code using bind variables as presented in Code Listing 4.24. The bind variables use the input value exclusively, and do not allow interpreting the input in any way.

```
PreparedStatement pstmt =
conn.prepareStatement ("insert into CUSTOMER (CUSTNAME) values (?)");
```

```
String name = request.getParameter("custname");
pstmt.setString (1, custname);
```

Code Listing 4.24 Secure code—SQL injection fixed.

- **Improper session handling.** Developers and reviewers have to take care of session related requirements and design such as whether the session time out has been set, whether the session of a user request has been validated before giving him/her access to a particular resource, etc.
- Authentication and authorization mechanisms should be checked for any inherent weakness.
- Internal error messages should not be leaked because of inefficient exception management.
- Reviewers and developers have to ensure that sensitive data like user credentials are not present in the code. Further, they should be stored or communicated in the encrypted form.

There are several other kinds of vulnerabilities that may exist in enterprise applications. The list is ever increasing, and organizations have to be on the guard against such dreaded vulnerabilities which may compromise application security.

The security code review can be automated or performed manually. Comprehensive manual code review can be quite exhausting, and requires a lot of time and patience from reviewer. Typically, for mundane tasks such as finding XSS or SQL injection pitfalls, automation is the best choice. Automated security code review is performed with the help of static code analyzers. These tools are quite effective and save time significantly. Both of these are "white box" approaches to security code review. Usually, a mix of manual and automated approach is used to analyze the code from the security perspective.

Security code review is a very important task during construction phase to ensure the detection and fixing security vulnerabilities, as early as possible. Deferring it to the end of software development life cycle (SDLC) may considerably increase the costs of remediation or may delay the application going live. This activity is different from "black box" security testing, more popularly known as *penetration testing*, which is typically performed just before the rollout of an application to the production environment. Penetration testing is done much after security code review in a typical SDLC.[7]

4.5.4 Code Quality

Code quality analysis is important to ensure that code being delivered is of the highest quality in terms of its modularity, extensibility, maintainability, reusability, testability and performance. Some of the key metrics that are used to measure the code quality are as follows:

- **Class size.** This metric is the measure of the size of a class in terms of lines of code. Big classes, say more than 1000 lines of code, have poor readability and maintainability. This may result in a higher cost of maintenance and introduction of more defects in the maintenance process.
- **Cyclomatic complexity.** This metric is the measure of number of linearly independent paths through a method. High value of this metric typically results in more testing effort to cover all the possible paths. Loops should be minimized, unnecessary conditions should be eliminated, and nested loops should be avoided to lower this metric. Typically, the maximum range of this metric should be between 8 and 12.

[7] You will learn more about penetration testing in Chapter 5.

- **Comments-to-code ratio**. This metric is the measure of ratio between comments to total lines of code. A lower comment ratio usually results in difficulty of understanding the code, and in turn, impacts the maintainability of code.
- **Number of attributes**. This metric is the measure of number of attributes in a class. A higher number may indicate that the class represents more than one entity, and may adversely impact its maintainability. If the functionality of a class can be logically broken down, then the class should be split into two or more classes. Otherwise, if the functionality of the class cannot be logically broken down, an abstraction level can be added.
- **Coupling between objects**. This metric represents the number of other classes to which a class is coupled. The count is based on number of reference types used in attribute declarations, formal parameters, local variables, return types, and throws declarations. More coupling between objects typically leads to less maintainability and more testing effort.

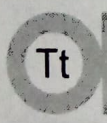
4.6 Build Process and Unit Testing

As in the case of construction of an enterprise application (where multiple teams are working on the code), it is a good practice to run unit test cases (along with the build process) to ensure the health of the increment at each step of construction phase. Continuous integration and unit testing are prominent practices of iterative software development to avoid last minute surprises. Let us first explore the build process, followed by unit testing of an enterprise application. Thereafter, you will explore how they work together during the construction of a typical enterprise application.

4.6.1 Build Process

Every enterprise application requires an accurate, automated and maintainable build process in order to effectively and efficiently build an application. The build process is not limited only to the compilation of source code, but also comprises of other key activities such as retrieving the appropriate set of code units from source code control systems, packaging application components and libraries, component specific compilation, and deploying the application into a development environment.

Though the build process can be executed manually, an automatic build is almost invariably preferred for building an enterprise application. A manual process may lead to confusions, errors, and a relatively longer duration to build applications. Automation usually helps in saving time and improving quality by standardizing the entire build process.



Tools such as Apache Ant and Maven are used to automate the build process of Java based enterprise applications.

A build utility, such as Ant, defines build scripts using XML and performs common build operations such as accessing source code control repositories to extract source code units, definition of build *class-paths* and source compilation. Figure 4-27 presents a partial snapshot of a sample ant build script—`build.xml`. It represents the target for the final deployable unit of a JEE based application, i.e., an enterprise archive (EAR) file, which comprises of all required Java archive (JAR) and Web archive (WAR) files of an enterprise application. Similar build scripts are also required to create the individual JARs/WARs for application layers. Dependencies among application components are also taken care of by build scripts by appropriately defining the build dependencies in the XML script.[8]

[8] To know more about Apache Ant, you may refer to the Apache Jakarta Web site, http://ant.apache.org/

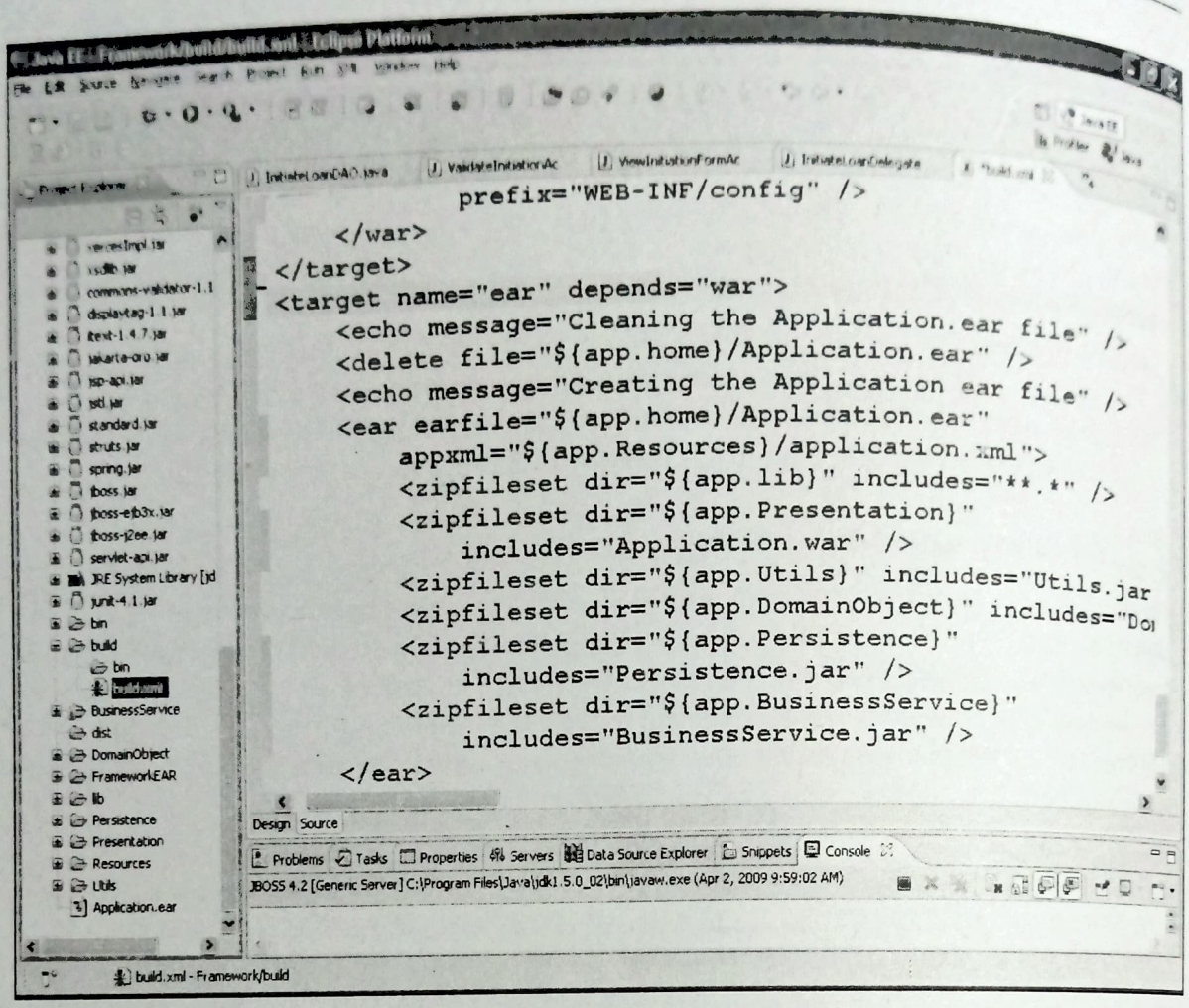


Figure 4-27 Sample `build.xml` (partial) file.

In Build scripts can be executed as commands on a need basis, or they can be scheduled based on a trigger condition or a designated time. The scheduled build processes are more appropriate in an open source development scenario, where several individuals or teams across geographies are checking in code into a central source code repository

The build process does not normally conclude after compiling and packaging the code. It is usually followed by running test cases and deploying the application. All this happens along with other peripheral work around this activity, which typically includes documentation activities such as generating Javadocs and release notes.

Tt For creating sophisticated build processes, the Jython scripting language can be used to extend Ant tasks. A right mix of Ant and Jython provides the power of Ant coupled with sophisticated scripting capabilities of Jython.

A build utility, such as Ant, can also run automated unit test cases as part of the build process, which helps in automating the entire process with minimal human intervention. Let us now understand unit testing—the first level of testing of an individual component/module of an application.

4.6.2 Unit Testing

After the development of a unit of code, it needs to be tested for various scenarios covering all the aspects of the functionality of the unit. Unit testing is the first formal level of testing in a SDLC, where individual units of software are tested, independent of other part(s) of an enterprise application. Units can be modules, procedures, methods or classes based on the paradigm of software development. As the code evolves, it is much easier and economical to find and eliminate the bugs early. Usually, it is the developer's responsibility to deliver unit-tested code.

Tt Unit testing can be a manual process or is automated by using testing frameworks/tools like JUnit.[9]

Unit testing brings in a lot of confidence with respect to the test coverage of the application. But what does one need to test during unit testing? For example, in the case of presentation components, one has to ensure presence of all form fields, and correct translation and localization in case the application is internationalized. For the business and integration components, control flows and looping constructs in a software unit need to be verified to ensure the total coverage of the unit. Unit testing is not only about testing positive cases but also negative ones. Erroneous and exceptional situations also need to be unit tested.

Unit testing is an elaborate process which spans across the life cycle of application development. Three primary steps typically followed in the unit testing process are as follows:

1. **Test planning.** *Test planning* is the first step, which is typically carried out towards the end of the requirements engineering phase. It involves creation of test plans which document the steps to be followed in order to carry out tests. It also includes other aspects of test planning that typically include what needs to be tested, whose responsibility is to test, what the prerequisites are and assumptions for testing, which criteria pass or fail the test, etc.
2. **Test cases and test data preparation.** Test cases documents need to be tested in order to ensure a correctly functioning unit of an enterprise application. This typically comprises of testing the interfaces and properties exposed by the software unit along with the internal structure of the unit in terms of control flows and application logic.

Table 4-5 represents typical elements, which need to be captured in a test case. It may be noted that these test case elements are not specific only to unit testing, and they can be used for other types of testing as well.

Table 4-5 Test case elements

Test case elements	Description of elements
Test case description	Describe the steps to execute the test case
Input test data	Data with which a given test case has to be executed
Expected result	The expected result after execution of the test case
Actual result	The actual result of execution of a test case
Pass/fail flag	Flag the outcome of a test case as "pass" or "fail"

Test data is the sample data with which test cases are executed to validate functionality. Test data should be selected very carefully to ensure complete coverage of functionality of the unit under test. Unit testing tools come with handy features such as support to input test data sources such as spreadsheet, providing flexibility during the execution of test cases.

3. **Execution of test cases and fixing the bugs.** Execution of test cases with selective test data is the third step that is performed after the construction of a unit of software. The actual results of test cases are compared with expected results, and test cases are designated as pass or fail. Bug fixing is done for all the failed test cases. The magnitude of change in the unit of software varies based on deviation between expected results and actual results. It may be as simple as fixing a simple logical error, or refactoring the code, or may lead to change in the underlying software design. This cycle is iterative in nature, and is repeated till a unit of software passes respective test cases and is flagged bug free.

In

It is worthwhile to mention that there is one more school of thought/paradigm that is related with testing — the *test driven* development. This is a technique to develop a software, which involves writing automated test cases before writing actual code. These test cases usually contain assertions that result in true or false to indicate the success or to failure of a test case. Typical Java IDEs like Eclipse comes with built in JUnit framework to facilitate the test driven development. To know more about test driven development you can refer http://www.testdriven.com.

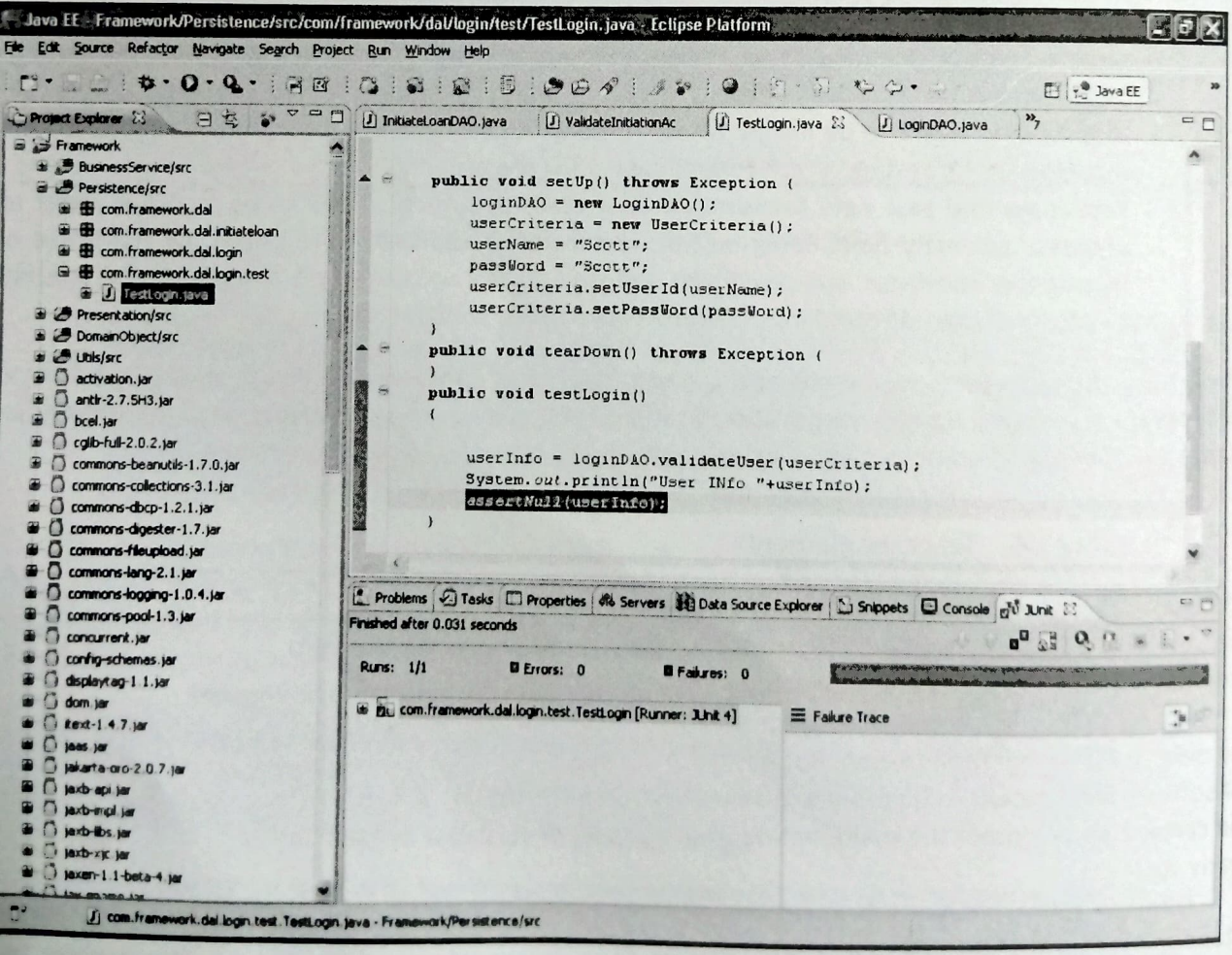
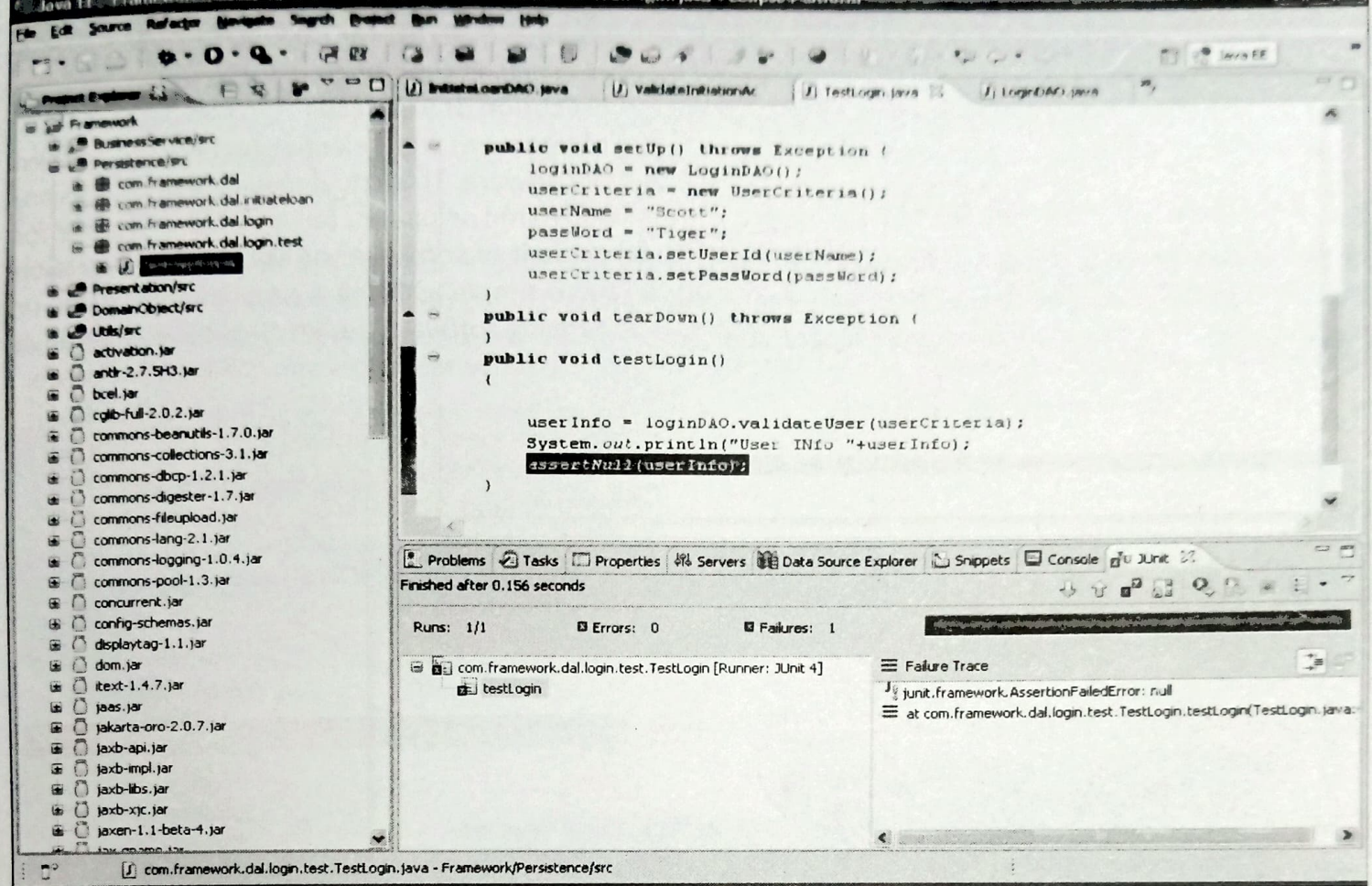


Figure 4-28 Unit test case run using JUnit—passed iteration.


Framework
BusinessService/src
Persistence/src
com.framework.dal
com.framework.dal.initiateloan
com.framework.dal.login
com.framework.dal.login.test
Presentation/src
DomainObject/src
Utils/src
activation.jar
antlr-2.7.5H3.jar
bcel.jar
cglib-full-2.0.2.jar
commons-beanutils-1.7.0.jar
commons-collections-3.1.jar
commons-dbcp-1.2.1.jar
commons-digester-1.7.jar
commons-fileupload.jar
commons-lang-2.1.jar
commons-logging-1.0.4.jar
commons-pool-1.3.jar
concurrent.jar
config-schemas.jar
displaytag-1.1.jar
dom.jar
itext-1.4.7.jar
jaas.jar
jakarta-oro-2.0.7.jar
jaxb-api.jar
jaxb-impl.jar
jaxb-libs.jar
jaxb-xjc.jar
jaxen-1.1-beta-4.jar
public void setUp() throws Exception {
loginDAO = new LoginDAO();
userCriteria = new UserCriteria();
userName = "Scott";
passWord = "Tiger";
userCriteria.setUserId(userName);
userCriteria.setPassWord(passWord);
}
public void tearDown() throws Exception {
}
public void testLogin()
{
userInfo = loginDAO.validateUser(userCriteria);
System.out.println("User INfo "+userInfo);
assertNull(userInfo);
}
Problems
Tasks
Properties
Servers
Data Source Explorer
Snippets
Console
JUnit
Finished after 0.156 seconds
Runs: 1/1
Errors: 0
Failures: 1
com.framework.dal.login.test.TestLogin [Runner: JUnit 4]
testLogin
Failure Trace
junit.framework.AssertionFailedError: null
at com.framework.dal.login.test.TestLogin.testLogin(TestLogin.java:
com.framework.dal.login.test.TestLogin.java - Framework/Persistence/src

Tt Tools like CruiseControl can be used to automate build and unit testing processes. Such tools are typically used in projects where there is a requirement to support continuous integrations of software units.

4.7 Dynamic Code Analysis

Dynamic code analysis enables a development team to obtain a holistic view of an enterprise application in the running state, contrary to the static code analysis that involves analyzing code without even compiling it. As illustrated in Figure 4-30, *code profiling* and *code coverage* are the focus areas of dynamic code analysis.

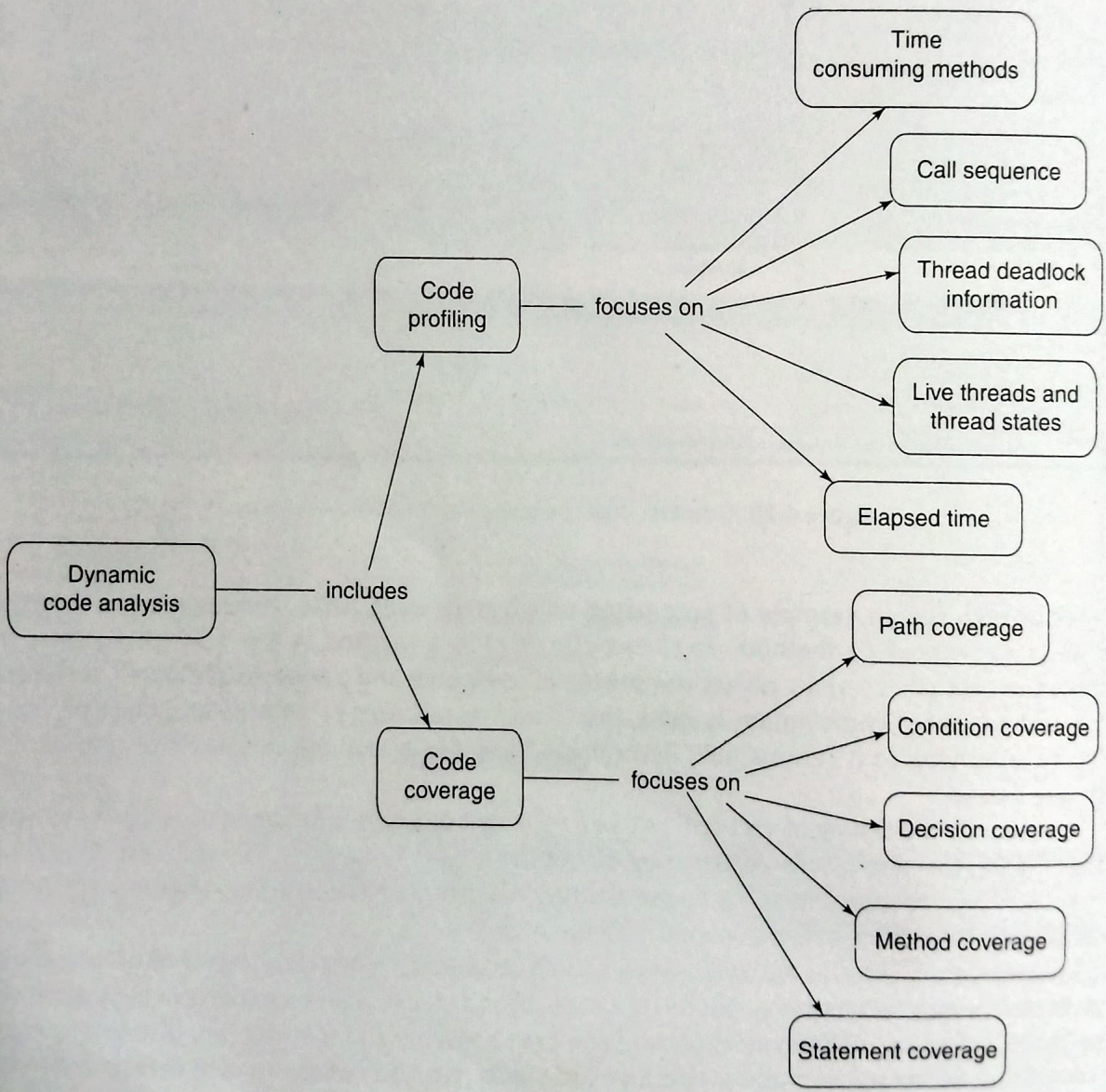


Figure 4-30 Dynamic code analysis objectives.

4.7.1 Code Profiling

Code profiling is one of the focus areas of dynamic code analysis, which is primarily concerned with the performance analysis of an application that enables the development team to quantify the performance aspects of an enterprise application. Code profiling captures various run-time attributes of the code, with which development team can identify the performance bottlenecks in an enterprise application. A few of the attributes, as shown in Figure 4-30, are described below:

- **Time consumed by methods.** This metric captures the time taken by the CPU to complete the execution of methods. It helps in identifying the most time-consuming methods and analysis of performance bottlenecks with respect to computational time.
- **Call sequence.** This attribute lists the actual sequence of method calls in a particular execution of the program. This helps in analysis of paths traversed during the code execution.

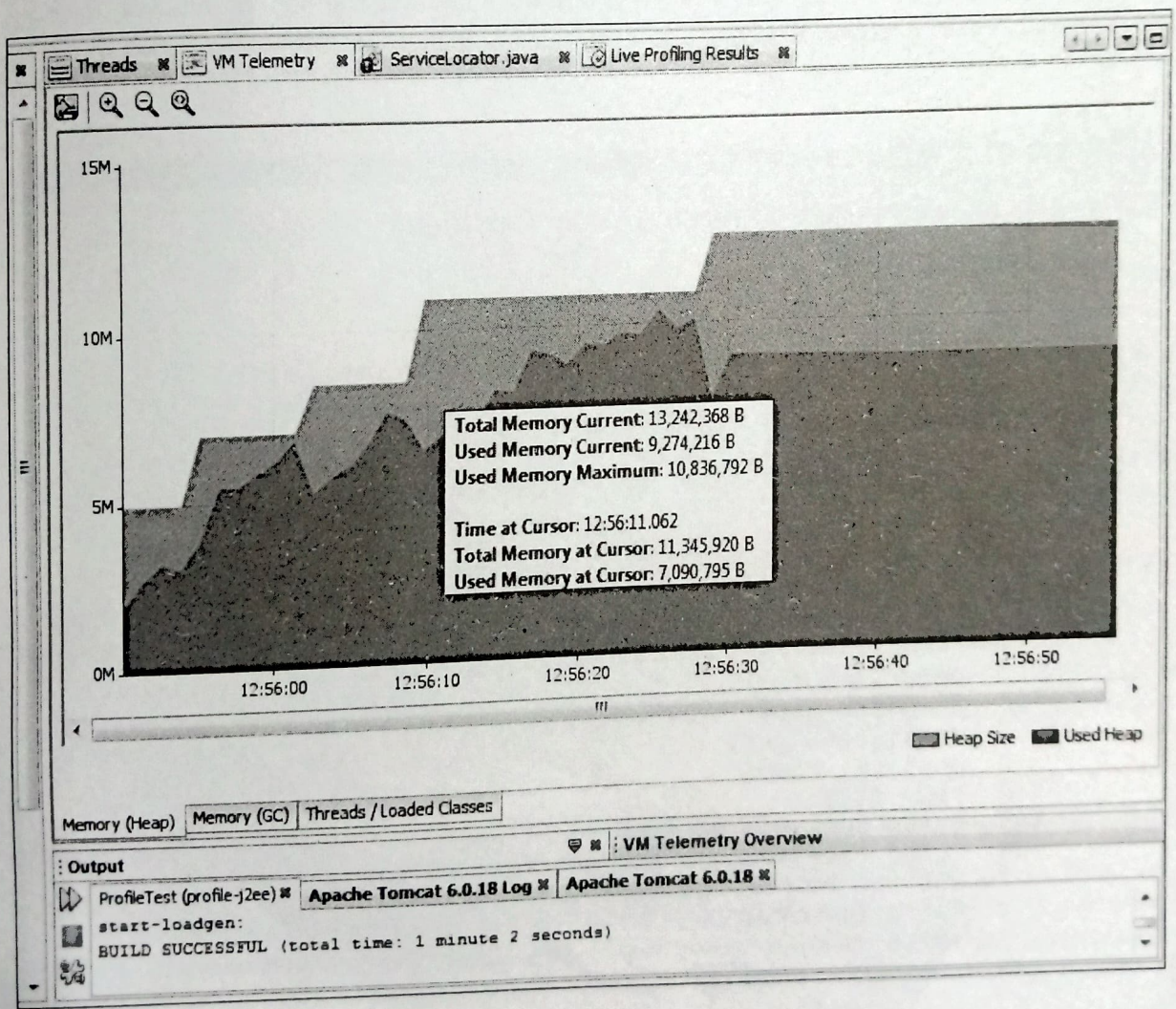


Figure 4-31 A snapshot from Java Profiler.

- **Thread deadlock information.** This data provides information on deadlocks due to resource contention or any other reason. This helps in identifying problematic sequences or scenarios which are potential sources of performance bottlenecks.
- **List of live threads and thread states.** This data provides information on live threads in a given analysis scenario, and the various states in which threads are running.
- **Elapsed time.** This metric, which includes sleep periods, program pauses and I/O waits, provides an indication of the impact on end user's experience. It helps in understanding if there are bottlenecks in execution of those result in perceptible time lags by end users.

Java Profiler tools typically provide a graphical "call graph" depicting the selected methods under analysis that provides an easy way to navigate through the code. This eases the task of analyzing the performance bottlenecks.

As shown in Figure 4-31, memory profiling provides information such as object instance counts on the heap, size of objects, etc. These statistics helps in finding issues such as memory leaks, loitering objects, or heap memory utilization that, in turn, provide ways to optimize memory usage.

Code profiling of an enterprise application is not just limited to the application code for business logic. The underlying database may also have several application components such as stored

```
C:\WINDOWS\system32\cmd.exe - sqlplus
SQL> SELECT custname FROM customer
  2  WHERE preapploanamt BETWEEN 370000 AND 500000;

10 rows selected.


Execution Plan
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=CHOOSE
   1    0   TABLE ACCESS (BY INDEX ROWID) OF 'CUSTOMER'
   2    1     INDEX (RANGE SCAN) OF 'CUSTOMER_A_INDX_2' (NON-UNIQUE)



Statistics
----------------------------------------------------------
          0  recursive calls
          0  db block gets
          4  consistent gets
          0  physical reads
          0  redo size
        589  bytes sent via SQL*Net to client
        503  bytes received via SQL*Net from client
          2  SQL*Net roundtrips to/from client
          0  sorts (memory)
          0  sorts (disk)
         10  rows processed
```

procedures and triggers. These components are typically written using languages such as PL/SQL and T-SQL, which also may need to be analyzed to further fine tune the performance.

Tt Commercial databases usually come with built-in profilers which can profile SQL queries and other database objects such as stored procedures and functions.

Figure 4-32 represents a sample output from the *Trace* utility of Oracle, which is used to profile a SQL query (depicted in the first two lines of the output). The output depicts the query execution plan that is a snapshot of how the database optimizer is running the query. It depicts which tables and indexes are scanned to retrieve the required data. It also provides statistics such as physical reads, database blocks affected and network roundtrips. Dynamic analysis greatly helps the development team identify and iron out any performance bottlenecks. The profiler utilities of databases typically have the capability to display metrics both in textual and graphical format.

What are the benefits for an application development team in capturing the code profiling metrics? These metrics provide clues to identify and plot the user behavior in terms of the frequently-used use cases. More tuning efforts can be focused on such use cases. These metrics also provide data on computational time, deadlocks and time consumed by the methods under analysis, which can be used for performance tuning. For example, if an operation to generate some data is expensive then the option of caching that data can be looked at. In some cases, the option of pre-computing some results can also be looked at to enhance performance. The metrics are not only used to find performance bottlenecks, but also used to benchmark the code. The next set of code releases can be compared with this benchmark to find further performance bottlenecks, if any.

One should remember that code profiling only provides objectivity to the performance analysis and helps in nailing down the performance bottlenecks. It does not add wings to your code. One has to take a cue from this analysis and spend effort to do performance tuning. It is also important to remember that performance tuning is not just about the code, but there are also network settings, operating system settings, application and database server configurations, JVM/container settings, etc., which, as a whole, determine the performance of an enterprise application. One has to consider a holistic view of all these elements and act upon the performance tuning of an enterprise application.

Even before all these tuning efforts, developers can really make a difference to the performance of an application by practicing a few basic good coding practices. Moving the code which is not required in the loop to outside the loop, avoiding method calls in the loop termination test, using primitive types in loop conditions and using "static final" for algebraic expressions so that the compiler can pre-compute the constant expressions, are a few examples of good coding practices to enhance the performance at the code level.[10]

4.7.2 Code Coverage

Code coverage is defined as the level, or percentage, to which the source code of an enterprise application is tested. This does not check the functionality of an enterprise application, but finds out what percentage of the codebase is not being executed as part of execution of test cases. This helps in ensuring that the test cases created for checking the functionality are adequate. Development teams may create

[10] To know more about for Java related performance issues and solutions, you may visit http://java.sun.com/docs/performance/

more test cases, if required, to increase the coverage level. The process of code coverage, as shown in Figure 4-30, primarily furnishes following metrics:

- **Statement coverage.** This measures the degree to which the statements of the application codebase under test have been covered during test execution.
- **Method coverage.** This measures the degree to which the functions or methods of an application codebase under test have been covered during test execution.
- **Decision coverage.** This measures the degree to which the Boolean conditions of all decision and looping constructs of an application codebase under test have been covered during test execution.
- **Condition coverage.** This measures the degree to which the Boolean conditions of all Boolean expressions of an application codebase under test have been covered during test execution. This is not similar to decision coverage, which is to measure whether all branches of decision and looping constructs are covered.
- **Path coverage.** This measures the degree to which the permutations of all possible logical flows in a given codebase under test have been covered during test execution.

The above list represents the primary metrics that capture code coverage. Apart from this, there exist several other metrics such as data flow coverage, race condition coverage that may also be captured. These metrics are very helpful to improve testing productivity and ensure that the application is tested for all conditions, branches, methods, etc.

Xp

At least 80% coverage of the underlying codebase is usually considered as a good level of code coverage.

Typically, both code profiling and code coverage tools use the same engine to perform dynamic code analysis, and hence often come bundled together. The enterprise application under analysis is run-on application servers, with which these tools are integrated and configured to perform dynamic code analysis.

Tt

- IBM Rational PurifyPlus suite comes with three tools, namely, IBM Rational Purify, IBM Rational Quantify and IBM Rational PureCoverage. Rational Purify can identify problems related to memory leaks, un-initialized memory, bugs in third-party libraries, etc. Rational Quantify has the ability to collect performance data from code, and identify performance bottlenecks. Rational PureCoverage indicates the amount of code tested and reports the level of coverage.
- Quest JProbe suite comes with JProbe Profiler, JProbe Memory Debugger and JProbe Coverage. JProbe Profiler is used to identify the performance problems, JProbe Memory Debugger identifies memory usage problems, and JProbe Coverage is used to identify the untested code.

Dynamic code analysis complements static code analysis, both of them together form a crucial part of analyzing software components developed during the construction phase. The testing and analysis activities should be done with a right mix of automated and manual modes to establish the best possible and optimized code in order to take it further in the next phase of raising enterprise applications—the testing phase that helps certify an enterprise application as "ready for service".

Summary

This chapter started with crucial activities related to construction readiness such as preparation of construction plan, definitions of package structures, establishing a configuration management mechanism, and eventually setting up a development environment for the construction team. It also introduced the concept and importance of software construction maps and presented some examples of them for better understanding of the concept. Here, you have learnt the construction of the application framework components and application components of the different layers of an enterprise application. The chapter also provided examples from the LoMS case study to illustrate the construction of infrastructure, presentation, business, data access and integration layer components. You have understood how to craft the infrastructure layer as part of the application architecture framework. We also explored class diagrams, sequence diagrams, software construction maps and relevant code across all layers of an enterprise application with specific examples from LoMS.

This chapter also introduced the code review process and static code analysis. In addition, coding style, logical bugs, application security vulnerabilities and code quality analysis have been introduced. This chapter also explained the concept of unit testing and build process, and how these processes work together. The concept of dynamic code analysis, which primarily involves code profiling and code coverage, is also discussed in this chapter.

Review Questions

1. List the prerequisites for starting the construction phase.
2. Explore the features of Java enterprise application development IDEs.
3. What is a software construction map?
4. What is aspect-oriented programming?
5. What are the typical components of a presentation layer package structure?
6. Describe one framework per layer used to build the application layers.
7. What is a dependency injection?
8. What are the objectives of code review?
9. What are the differences between a static code analysis and a dynamic code analysis?
10. Describe the three most dreaded application security vulnerabilities.
11. Explore JUnit framework.
12. Explore Jython and Ant scripts.
13. List the metrics of code coverage.
14. How does code profiling help in tuning the performance of an application?
15. List the tools used for static and dynamic code analysis.

Further Readings

Apache Ant: http://ant.apache.org/
Apache Log4J: http://logging.apache.org/log4j
Application security by OWASP: http://www.owasp.org/
AspectJ: http://www.eclipse.org/aspectj/
Dependency injection: http://www.martinfowler.com/articles/injection.html
EJB: http://java.sun.com/products/ejb/
IBM Rational PurifyPlus : http://www-01.ibm.com/software/awdtools/purifyplus/

Testing and Rolling Out Enterprise Applications

5

Learning Goals

After completing the chapter, you will be able to:

- Describe different types and methods of testing enterprise applications.
- Explain testing levels and testing approach.
- Classify enterprise applications environments.
- Explain integration testing.
- Explain different types of system testing.
- Explain user acceptance testing.
- List the tools used for different types of testing.

You have so far learnt that thorough requirements engineering, flexible and robust architecture and design, and good coding practices are essential to raise successful enterprise applications. Along with these crucial elements, comprehensive testing of enterprise applications is equally important. Testing of enterprise applications is performed not only to ensure that the application delivered has the right set of functionalities but also to guarantee the required quality attributes or "ilities". Testing constitutes approximately 30 percent of the overall efforts spent in raising enterprise applications and hence designated as one of the most important life cycle phases of raising enterprise applications.

This chapter will introduce the concept of testing enterprise applications and the testing approach. Different types of testing—integration testing, performance testing, penetration testing, interface testing, user acceptance testing—are discussed. Various enterprise application environments and tools used for testing are discussed. Towards the end of the chapter, you will learn about the procedures for rollout of enterprise applications.

5.1 Testing Enterprise Applications

Companies across industries depend on mission-critical enterprise applications to drive the business and business initiatives. The goal invariably is to strive for high speed of development, yet creating reliable high-quality applications, and most importantly, staying within budgets. Companies would not want to compromise on the quality of these applications, considering the potential fallout of losing customers and business.

The testing of enterprise applications encompasses the answers to the questions listed in Table 5-1, which includes identification, planning and execution of testing.

Table 5-1 Testing enterprise applications

What	What needs to be tested? What kind of testing to be done? On what conditions do we enter into testing? On what conditions do we exit the testing?
How	How does the software need to be tested? How much testing is required before rollout? How long does each type of testing need to be done?
Where	Where does the software need to be installed and tested?
When	When should we plan to start the testing? When should we end the testing?
Who	Who are involved in the testing?

Let us address the above questions one by one to have a complete understanding of testing enterprise applications.

5.1.1 Types and Methods of Testing

Each enterprise application undergoes various types of testing to ensure that it demonstrates the desired functionalities and quality attributes. Different types of testing are performed in different phases of raising enterprise applications. Testing can be categorized using various perspectives. Two of the most important perspectives to categorize the testing are "What" needs to be tested and "How" to test.

Based on "What" needs to be tested, testing can be divided into two categories:

(a) **Functional testing.** Testing the functionalities as per the stated requirements is termed as *functional testing*. This testing typically involves testing the screen flows, data flows, business logic, user and system validations and codified business processes.

(b) **Nonfunctional testing.** *Nonfunctional testing* caters to testing all other things except functionalities to ensure that the quality attributes are as per the requirements. Typically, these include security, performance, maintainability, reliability, availability, scalability, etc.

Based on "How" to test, the method of testing can be divided into two categories:

(a) **White-box testing.** This method of testing tests the internal structure of the application in detail. This method typically checks for the flow of data, control and branching sequences. Knowledge of algorithms and data structures used in the application is required in *white-box testing*.

(b) **Black-box testing.** This method of testing, on the other hand, tests the software without need for understanding the internal structure and focuses on testing the application from an external perspective. This testing method treats the unit/component/application to be tested as black box and assumes no knowledge of algorithms and data structures used in the application.

In *Gray-box* testing is a hybrid form of white-box and black-box testing methods. In this method of testing, the test planning is done keeping the internals in mind, but finaily testing happens like in black-box testing.

5.1.2 Testing Levels

Testing has to be done at several levels before rolling out enterprise applications. This answers the questions: "How" much to test and "When" to test enterprise applications. Figure 5-1 depicts the different levels of testing that enterprise applications undergo starting from unit testing, integration testing, system testing, acceptance testing and finally the production testing.

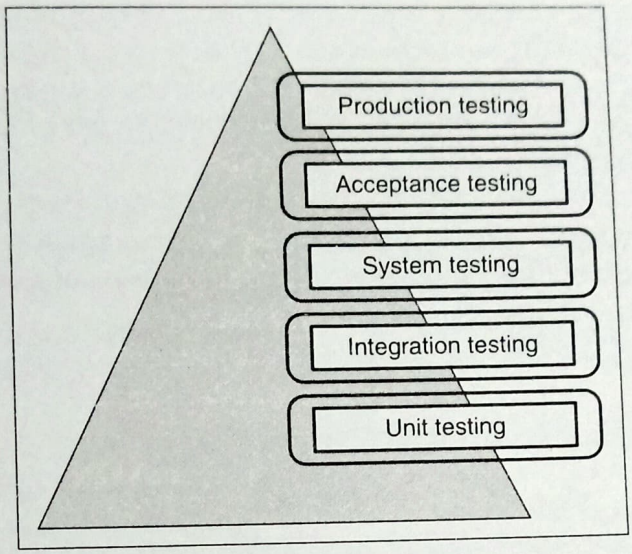


Figure 5-1 Testing levels of enterprise applications.

The following are different types of testing levels:

(a) **Unit testing.** *Unit testing* is performed to ensure that a standalone function/code/module/class works as per the specifications. This testing is the first level of testing that happens in the construction phase of enterprise applications development. You can refer to Chapter 4 on the construction for details on unit testing.

(b) **Integration testing.** *Integration testing* is to ensure that the software code/modules/functions/classes are grouped together consistently. The units of these groups are tested including the interaction among these units to find out any inconsistencies among these units. In essence, integration testing is performed to test that a combination of classes/units correctly works together.

(c) **System testing.** A *system testing* takes place after the integration testing is completed. This type of testing involves testing the entire system as a whole by providing different types of input to the enterprise application and verifying that the different functionalities of the system work end-to-end. Interfaces to and from the system under test are tested in this phase. This is the third level of testing in which the system is also investigated holistically from the perspective of functional as well as nonfunctional requirements.

(d) **Acceptance testing.** *Acceptance testing* is done towards the end of testing phase, just before the rollout of the enterprise application, by the end users and primarily includes the test cases based on a selective subset of end user requirements. This is the fourth level of testing, and only upon successful completion of this testing the software product is deemed accepted by the client.

(e) **Production testing.** The last level of testing in the testing phase is the production testing. It's basically a pilot release of the enterprise application in the production environment with a subset of user base, product base and transaction volumes to have a firsthand feel of the live enterprise application.

5.1.3 Testing Approach

Testing need not have to occur only after the construction of enterprise applications. Testing typically starts as early as inception of enterprise applications. In fact, testing needs to be planned and executed in parallel to the overall life cycle phases of raising enterprise applications. One of the popular models of testing is the *V-Model*, which is essentially a software development model with a focus on testing spread across the entire life cycle.

We covered the various types of testing, testing methods and levels of testing in the preceding subsection. Let us now explore the approach used to test enterprise applications. Figure 5-2 depicts the generic approach used for enterprise application testing, comprising of four stages—test strategy, test planning, test execution and test analysis:

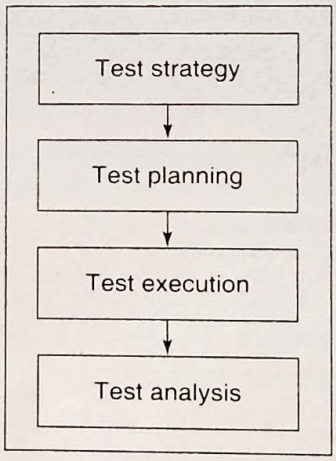


Figure 5-2 Enterprise application testing approach.

(a) **Test strategy.** *Test strategy* refers to the overall approach to test the enterprise application. It takes into account the environment, the types of testing to be done, the testing team, the effort, the cost and the timeline of testing, defect tracking, tools to be used, test planning and when to end the testing. The output of this phase is a test strategy document.

Test strategy also includes the entry and exit criteria for various types of testing. Entry criteria and exit criteria of testing answer the questions related to "when" to start the testing and "when" to end the testing. Before getting into any type of testing, the software should be ready in certain aspects. These aspects are termed as *entry criteria* that specify the prerequisites for any type of testing activity. No amount of testing can guarantee the complete elimination of defects, but it is important to know when to stop. Exit criteria specify when to stop a particular testing phase/cycle and move on to the next one.

(b) **Test planning.** Test planning is used to plan the execution of tests as decided in the test strategy. A good test plan can even find the gaps in requirements or bring clarity to requirements. It contains information about the tests to be performed and provides the overall direction to the testing activity along with test cases and test procedures. Test cases specify the specific features and scenarios for testing, the steps required to carry out the test and the test data along with the expected results. Test procedure explains the steps involved in executing the test for a specified test case.

(c) **Test execution.** It is easy to execute the tests if there is a detailed test plan prepared. Test plans have to be kept up to date and expected results correctly documented. Development team may be responsible for executing the unit testing and integration testing. Typically separate teams are formed including external ones to execute system testing for a large enterprise application. It is important to ensure that all the test preparations are done and that entry criteria for the tests are met along with the test data required for testing. Each of the test cases is then executed and the results are logged. All the integration points also need to be checked. Defects are entered in the defect tracking system and passed on to the application development team. Test execution is done for both functional and nonfunctional testing. Testing is considered complete when the quality goals set in the test strategy are met.

(d) **Test analysis.** The defects captured are analyzed for complexity, type, severity, impact and many other things. Defects found during testing can be a good leading indicator for what can be expected in acceptance testing. There is a saying that the more number of defects you find upfront, there are still some more defects crossing that stage. Corrective actions are taken based on the defect analysis.

Table 5-2 provides a bird's eye view of the enterprise application testing activities spectrum and how they are typically distributed across the development lifecycle.

Table 5-2 Enterprise application testing matrix

	Strategy	Planning	Execution and analysis
Inception	• Acceptance test strategy	• Acceptance test planning	
Architecture and design	• Unit test strategy • Integration test strategy • System test strategy	• Unit test planning • Integration test planning • System test planning	
Construction			• Unit testing
Testing			• Integration testing • System testing • Acceptance testing

Several tools are used in testing: automated regression testing tools, traceability tools, code coverage tools, performance and load testing tools, defect tracking tools, static analysis tools, compliance checking tools, test case generation tools, test data generation tools, etc. Depending upon the need and the complexity of the enterprise application, different tools can be used to increase the effectiveness of testing and the overall productivity of raising enterprise applications.

5.2 Enterprise Application Environments

You are already familiarized with the technical architecture building blocks and the deployment view in Chapter 3. The physical representation of these building blocks is referred to as enterprise application environment. There are several types of application environments that an enterprise application passes through in its lifecycle:

- Development
- Test
- Quality assurance
- Staging
- Production

These application environments are shown in Figure 5-3 and explained in Table 5-3.

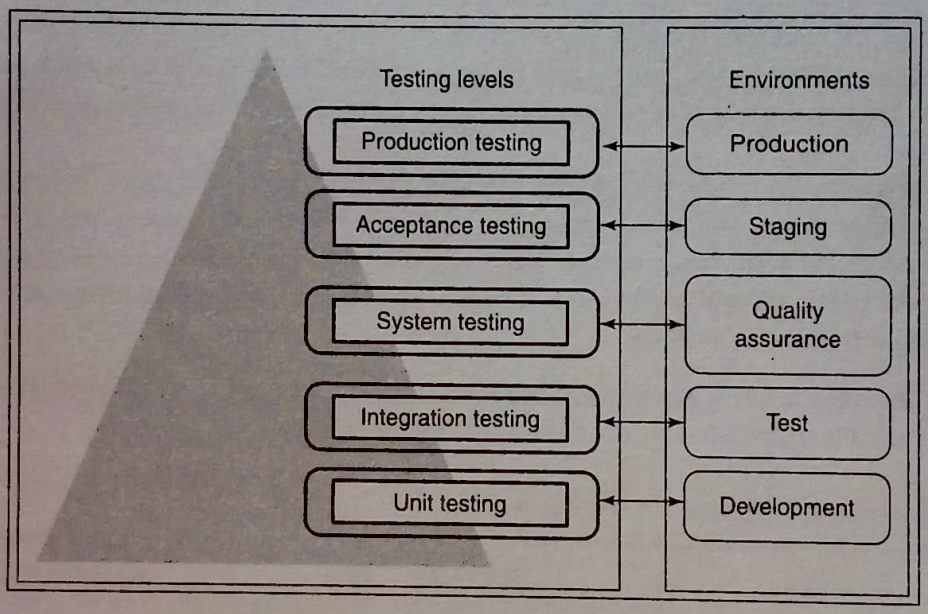


Figure 5-3 Enterprise application environments and testing levels.

Table 5-3 Purpose of application environments

Application environment	Purpose
Development	Development environment is a high-access, low-budget and low-security environment used mainly by the developers for construction activities including unit testing.
Test	This environment is used by the developers to perform integration activities and to do integration testing. Test environment will typically have moderate access control and moderate security.
Quality assurance (QA)	QA environment, having high-access controls, is typically used for QA activities by an external QA team. Most of the nonfunctional requirements are tested in this environment.

(*Continued*)

Table 5-3 (*Continued*)

Staging	Staging environment mimics production environment in all aspects, possibly except for capacity and security. This environment may be used for training end users. This ensures all physical configurations are tested and ready for service.
Production	Production environment is the environment where the final enterprise application eventually resides. This environment has high security and high access control. The capacity of this environment is designed to cater to real-life requirements. This also has to ensure high availability.

You may wonder why so many environments are required to raise enterprise applications. Each environment serves a specific purpose in the overall enterprise application landscape and is normally mapped to one or more of the testing levels.

Apart from the testing levels, other considerations, such as security, availability, access control, ownership of environment, and available budgets and capacity, determine the type of enterprise application environments. By now, you would have noticed that the maturity of the enterprise application increases as it moves from development environment to the production environment.

Let us now discuss how the LoMS application production environment needs to be built. To start with, let us go back to the nonfunctional requirements of LoMS discussed in Chapter 2. A few key nonfunctional requirements are as follows:

- LoMS shall be able to process a peak of 60 and an average of 40 concurrent transactions per second, support 250 concurrent user sessions and process 1000 loan applications in a day.
- LoMS should be able to support an average growth of 20% in volume of loan applications every year.
- LoMS is an Internet facing application with availability requirement of 99.999%.

Based on the above nonfunctional requirements, the server configuration of the production environment has to be worked out. For example, the following need to be considered in the planning of the LoMS application's production environment:

- **Concurrency.** Based on the requirements for concurrency, the production environment should be planned with load balancing capabilities.
- **Capacity.** Based on the average and peak load requirement, the production servers—both database and application—need to be sized.
- **Security.** Since LoMS is an Internet-facing application, the firewalls and DMZ need to be in place.
- **Scalability.** The configurations of the production servers need to be modeled based on the current and future requirements of the user base and number of loan applications expected.
- **Availability.** Based on the availability requirements of LoMS, it should be fault tolerant. For this, LoMS production servers should be clustered.

The deployment architecture of enterprise applications has to be planned based on such factors. Figure 3.33 (in Chapter 3) on architecting and designing depicts the deployment view of LoMS application.

5.3 Integration Testing

The software units/components are integrated into assemblages after unit testing. The interfaces among these underlying units/components are tested to ensure the working of these units, when

combined together, and is referred to as *integration testing*. Integration testing teams can use software construction maps, along with other UML diagrams such as sequence diagram, to understand the big picture of these units/components.

Integration testing is a classical example of gray-box testing. Integration testing team needs to know about the individual units/components to test the assemblages but need not always necessarily delve into the specific details of individual units. Entry criteria for integration testing are completion of integration of components, and addressing the requisite functionality as part of the construction phase.

Let us take the sample scenario of submitting and saving the loan details of the "initiate loan" use case. The sequence diagram is presented in Figure 5-4.

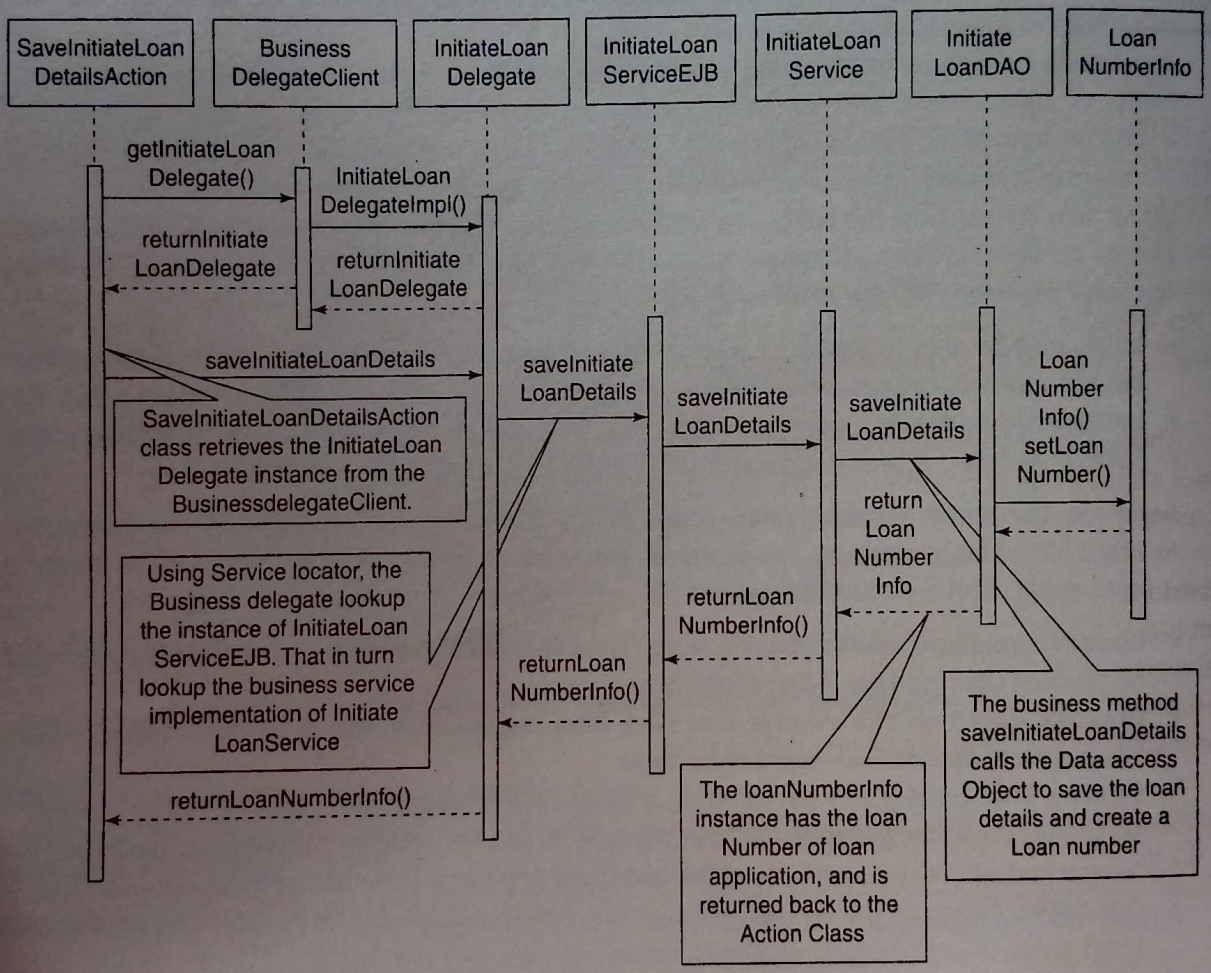


Figure 5-4 Scenario for submit and save loan details.

Entry criteria for the integration testing of this scenario is to have the order form displayed with all the details available to initiate the loan. All the components depicted in the sequence diagram (Figure 5-4) must be completed and integrated to start the integration test.

Table 5-4 Sample integration test cases

Test case	Components under test	Expected result	Result
Enter the details of loan application in the loan initiation form and submit the same.	`InitiationForm.jsp` and `SaveInitiateLoanDetailsAction` class	`SaveInitiateLoanDetailsAction` should be invoked	Pass/Fail
Store loan application attributes of InitiateLoanDAO in the underlying loandetails table	`InitiateLoanDAO` and `loandetails` table	All loan application attributes are stored in loandetails table and respective loan number is generated	Pass/Fail

Similarly there are various scenarios, which exist across the entire application that need to be tested as part of the integration testing. Multiple cycles of integration testing are conducted till all the scenarios are successfully verified. Exit criteria for integration testing are when the system provides the expected result for all the test conditions and scenarios under integration testing.

Xp

Should integration testing be done after all the components are developed as a big bang or it be incremental in nature? we suggest the incremental approach as it helps validate the application and framework components at the earliest!

Though some tools help in automating integration testing, it is more a manual intensive testing. Once all types of components related to enterprise application are integrated and tested, the application is ready to move into system testing phase.

In iterative methodologies, the same component and assemblages have to go through multiple transformations throughout their life cycle. It is important to ensure that a change in component does not break another part of the application. To ensure this, after each iteration the application needs to be regression-tested. Regression testing includes running the previous test cases and comparing the results with the previous test results. Usage of regression testing tools helps significantly in saving effort for testing frequently changing and complex components and assemblages.

Tt

HP Quick Test Professional and IBM Rational Functional Tester are some of the regression testing tools.

5.4 System Testing

System testing is testing the enterprise application and its interfaces as a whole. It is a black-box testing, where a completely integrated enterprise application is tested for the specified functional and nonfunctional requirements.

Though testing functionality, using end-to-end scenarios, is integral to the system-testing phase, this chapter focuses more on the nonfunctional testing aspects. To answer "what" kinds of testing need to be done, let us explore a few of the nonfunctional testing types that are carried out in the system testing phase of enterprise applications in the following sections.

5.4.1 Performance Testing

Performance testing is one of the most important aspects of system testing carried out from the perspective of validating the responsiveness of the system under a given workload profile, scalability to handle growth in number of users and data volume and stability without incurring visible performance degradation. Typically performance testing comprises of activities such as load testing, volume testing, stress testing and endurance testing. All these types of testing enable verifying the application's performance in a holistic manner.

Load testing

Load testing is where the enterprise application is subjected to a variety of loads, mainly in terms of the number of users accessing it, to baseline the performance of the enterprise application. Typically the application is subjected to a model of expected usage to determine the performance capabilities of the system.

Volume testing

Volume testing is somewhat similar to load testing carried out by varying the volume of data processed by the system. This type of testing is usually done on all key, frequently-used functionalities.

Stress testing

Stress testing is done to ensure that the enterprise application can handle the spikes in load due to unusual circumstances like a sudden surge in user hits. It is to test the breaking point when the enterprise application is put to extreme stress in form of a large volume users and data. It provides very useful information to plan the configuration of system live/production environment.

Endurance testing

Endurance testing is performed primarily to identify problems related to memory leaks by putting the enterprise application through sustained load conditions.

Performance testing has the following objectives:

- Determine enterprise application performance at a given user load and data volume—load testing and volume testing
- Determine the breaking point of the system under mounting load—stress testing
- Simulate the conditions to determine the reasons for performance bottlenecks—endurance and other forms of performance testing mentioned above
- Benchmarking the performance

Xp

Performance should not be an afterthought. It should be engineered right from the inception of an enterprise application.

Performance testing is a part of the overall performance engineering life cycle, as shown in Figure 5-5. It starts with the elicitation and validation of performance NFRs in the first phase of raising enterprise applications. Performance test strategy and plans are laid out in the architecture and design

phase. Design considerations specific to performance perspective are also taken care of in this phase. Size of infrastructure to meet the requirements is also determined in this phase. Performance testing happens as part of system testing in the testing phase. The testing results help in determining the performance bottlenecks in the application that are tuned to meet the expected performance requirements. These results also help in benchmarking the performance of the application and capacity planning before the rollout of the application.

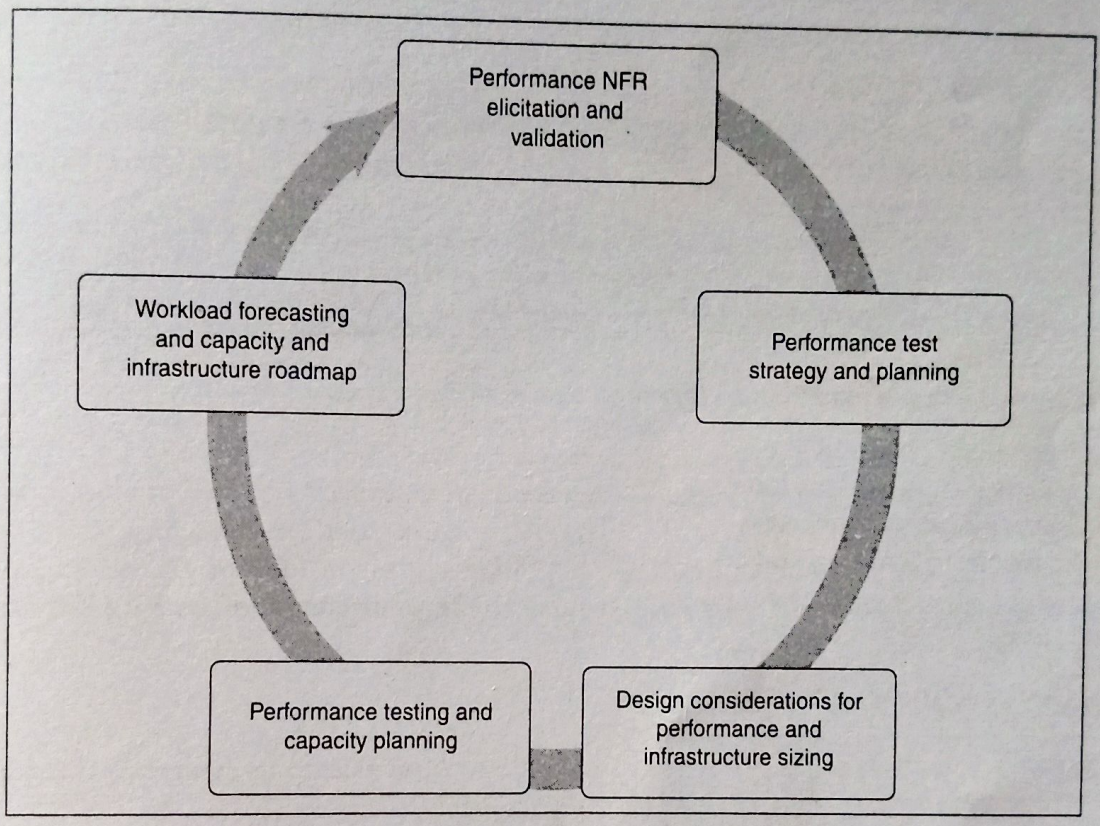


Figure 5-5 Performance engineering life cycle.

Performance engineering is a continuous endeavor to keep enterprise application up and running, and sustaining the required performance levels of the application. Performance testing is usually performed in the quality assurance environment with the use of specifically designed test scripts and tools. The test results are used to gauge and investigate the performance related issues. The output of performance testing is the recommendation for corrective action.

Tt

Apache JMeter and Loadrunner are examples of tools used to test the performance of enterprise applications. They provide support for various kinds of performance testing, including load testing, volume testing, stress testing, etc.

The performance NFRs of LoMS are listed in Table 5-5 for easy reference.

Table 5-5 Performance NFR metrics for LoMS

Maximum number of concurrent users	250
Peak throughput (TPS or hits/sec)	60
Average throughput (TPS or hits/sec)	40
Maximum refresh/response time (in seconds)	5
Number of loan applications per day	1000

Based on the above performance NFRs, the load test is designed to be performed using JMeter with the following load conditions:

- Number of users: 250 (in JMeter, users correspond to threads, i.e. virtual users)
- Ramp up period: 60 seconds

Xp

For a performance test to simulate the real behavior of the system, it is important to identify the appropriate workload mix for the system. Workload mix is the collection of various activities that system users may perform in the system during a given period of time.

The test plan that is used for load testing comprises of the following test cases:

- Login to the LoMS application
- Initiate loan application form
- Loan application submission
- Logout from LoMS application

These test cases are further broken down into the individual user activities/transactions, as shown in Table 5-6.

Table 5-6 Performance (load) test cases

Test case	User transactions	JMeter transaction names
Login to the LoMS application	1. Load the login page of the application in the browser by entering the address of the Web site. 2. Provide user name and password of a valid user and submit.	1. Load login page 2. Login transaction
Initiate loan application form	Clicks on initiate loan link in the menu and waits for the loan application form to be loaded. This test case is executed by a virtual user a predefined number of times in a loop.	3. Load the home page
Loan application submission	Fills the application loan form and submits. The loan application ID gets generated in response. This test case is executed by a virtual user a predefined number of times in a loop.	4. Initiate loan action
Logout from LoMS application	Clicks on logout link.	5. Logout transaction

The above user transactions are recorded in JMeter using an HTTP Proxy server element of JMeter, as shown in Figure 5-6.

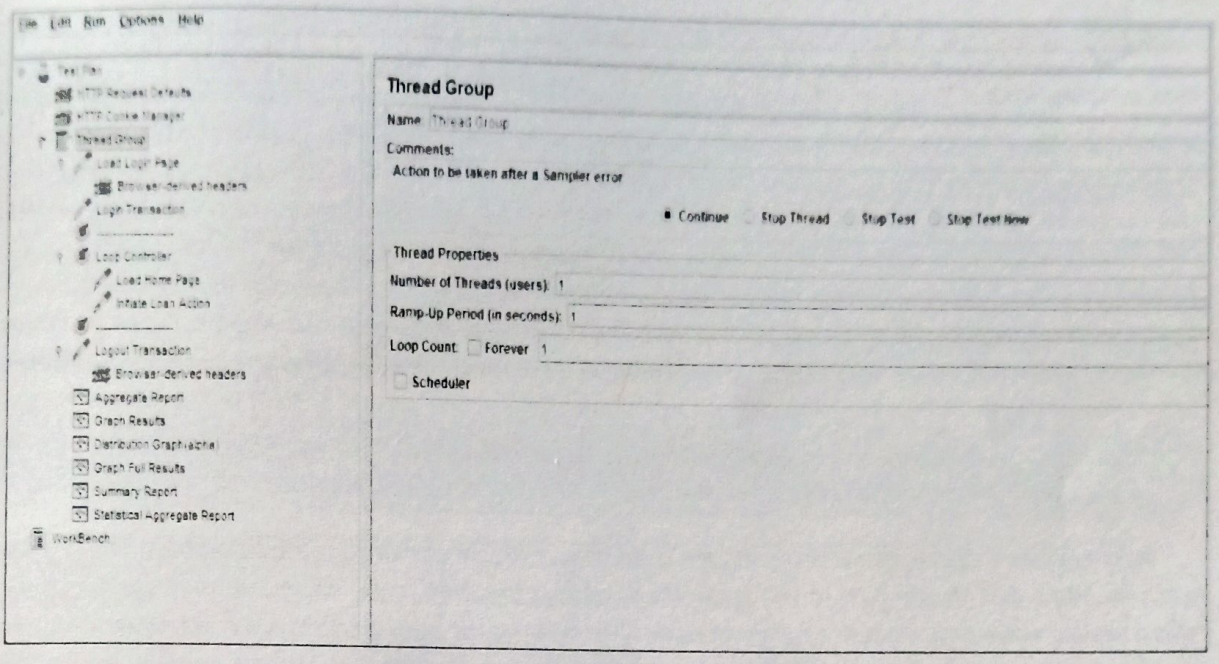
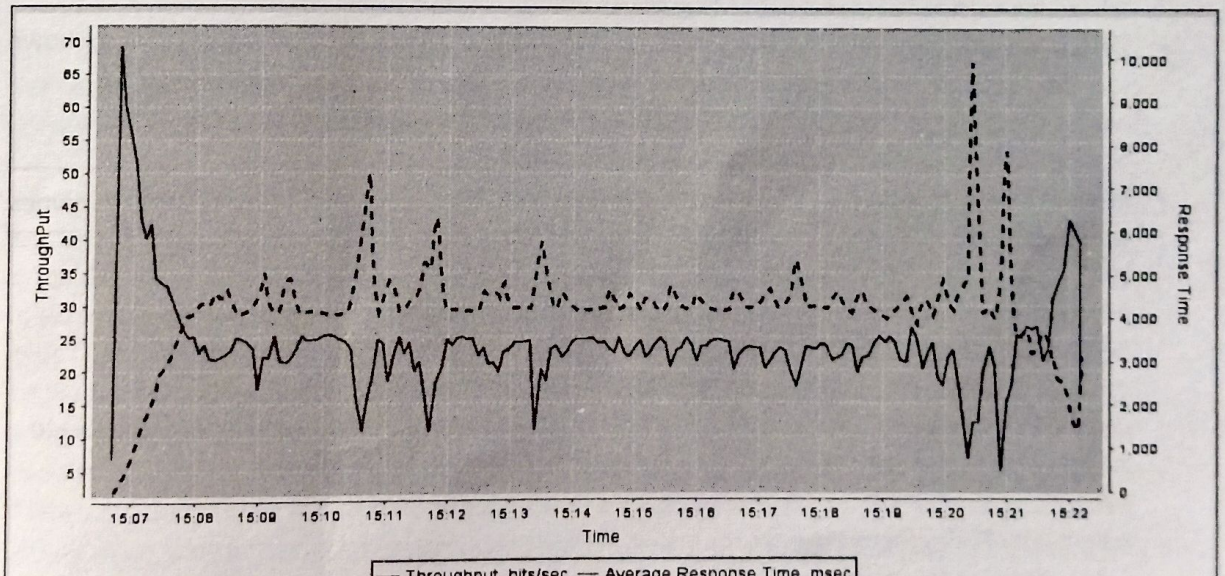


Figure 5-6 JMeter test plan setup.

The user transactions are captured in a single-thread group element and appropriate listeners are added to it to capture the metrics of the performance run. A statistical aggregate graph is generated using JMeter, as shown in Figure 5-7, which shows the variation of the key application performance metrics over the duration of the performance test run.



URL	# Samples	Average	Deviation	Min	Max	Errors	Throughput	KB/sec
Logout Transaction	250	1439.476	1575.5899775	46.0	9781.0	0	43.7/min	2.43
Initiate Loan Action	25000	7052.13508	1844.8387296	15.0	16906.0	0	26.5/sec	83.48
Load Home Page	25000	1465.07172	556.18102076	0.0	9922.0	0	26.9/sec	156.08
Login Transaction	250	3327.4	2156.7666596	15.0	7172.0	0	3.7/sec	10.87
Load Login Page	250	1007.688	474.44528310	0.0	2109.0	0	4.1/sec	13.84
TOTAL	50750	4224.11450246	3106.7239197	0.0	16906.0	0	54.5/sec	247.99

Figure 5-7 Statistical aggregate report.

The results of the performance (load) run are depicted in Table 5-7.

Table 5-7 Results of the performance (load) run

Performance NFR	Expected result	Actual result	Status
Maximum number of concurrent users	250	250	✓
Peak throughput (TPS or hits/sec)	60	69	✓
Average throughput (TPS or hits/sec)	40	54.5	✓
Maximum refresh/response time (seconds)	5	16.91	x
Number of loan applications per day	1000	25000	✓

Xp In a scalable enterprise application, while the system resources' utilization increases with increase in user load, the response time should ideally remain constant.

As mentioned above, the test results are used to investigate performance bottlenecks. The bottlenecks could be due to any part of the application—the application code, database, Web server, application server, operating system or the network. The reasons for poor performance are several—poor programming practices including database queries or incorrect configurations of servers, operating system and network. Corrective action plans are laid out after performance testing to optimize the performance that typically involves the following activities:

- **Application code analysis.** Run-time behavior of application code is analyzed to investigate the performance bottleneck. It primarily includes profiling the application code base to determine time consuming methods, call sequences, thread deadlock information, list of live threads and thread states, etc., as explained in Chapter 4.
- **Database optimization.** Database schema and stored procedures are also profiled to determine recursive calls, physical reads, number of deadlocks, wait events, etc., to point out the performance bottlenecks as explained in Chapter 4. Database schema objects, like indexes and tables, are also tuned to optimize the database performance.
- **Infrastructure resources and capacity optimization.** Resources primarily include networks, servers—application, Web and database and operating systems. Network latency rate, collision rate, incoming/outgoing packet rate, etc., are the performance parameters to watch for a network. Memory and the processing capacities of various servers also play an important role in determining the performance of enterprise applications. Memory profiling is achieved through dynamic analysis of the application code base, which provides information particularly on object instance counts on the heap, size of the objects, call tree, etc. This helps in determining the memory leaks and setting the right heap size. CPU profiling is performed, which also involves dynamic code analysis, to identify most time-consuming methods, call sequences, etc., which help in identifying the computation related performance bottlenecks in enterprise applications. Some of the concepts were discussed in Chapter 4.

In In case of Java based enterprise applications, JVM profiling is also performed to identify the internal processes of the JVM, most importantly the garbage collection process.

5.4.2 Penetration Testing

Penetration testing is the black-box testing of an enterprise application from the security perspective. This testing not only helps in fortifying the application against the various types of online threats but

also ensures the application's compliance to the industry regulations. This is one of the important types of testing from the perspective of building the enterprise brand and the customer confidence. It is mandatory for B2C applications and community sites involving large number of users.

Penetration testing is a part of the overall application security engineering life cycle, as shown in Figure 5-8. It starts with the elicitation and validation of security NFRs in the inception phase of raising enterprise applications. Security test strategy and planning is laid out in the architecture and design phase. Threat modeling is also performed in this phase. This exercise is performed to determine the security objectives, threats and vulnerabilities of enterprise applications. The output of threat modeling helps in determining the design decisions for application security. The application code base is analyzed in the construction phase for security vulnerabilities. Penetration testing happens as part of system testing in the testing phase. The testing results in unearthing the security vulnerabilities in the application, which need to be fixed before the production rollout of the application. These results also help in ensuring legal compliance of the application from the data security perspective.

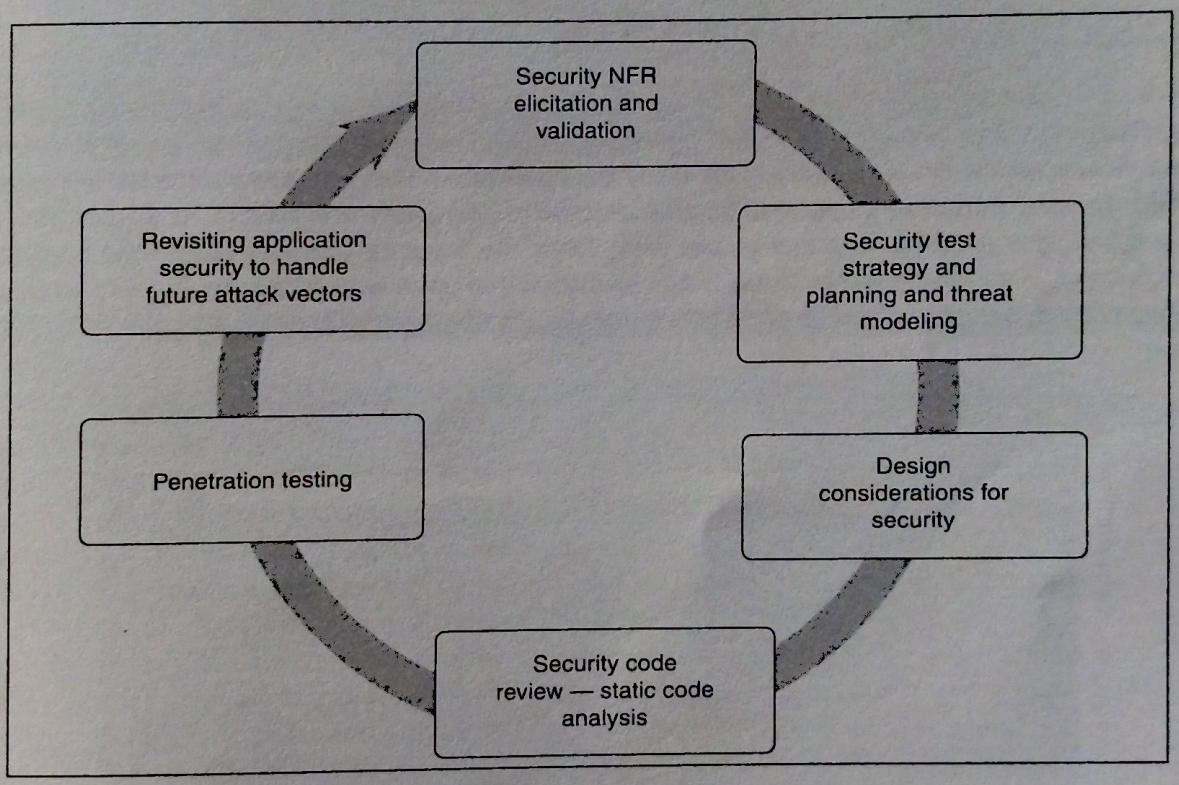


Figure 5-8 Application security engineering life cycle.

Application security engineering is a continuous endeavor to guard enterprise applications from current and future attack vectors and helps in fortifying the application against different forms of attack. Penetration testing is typically a mix of manual and automated testing. Some of the security vulnerabilities can be unearthed by manual testing while others are easier to figure out by automated testing using penetration testing tools.

Tt Paros Proxy and OWASP WebScarab are a few of the freely available and popular tools for Web application security assesment. Fortify 360 is one of the commercial tools available for performing penetration testing.

Security requirements for LoMS dictate taking care of the OWASP top-10 security vulnerabilities. A subset of penetration test cases based on this is shown in Table 5-8.

Table 5-8 Penetration test cases

Test case	Expected result	Result
Input the following text in the username field of the login screen. `Scott' # <iframe src="http://www.google.com" width="500" height="500"></frame></b><b>`	System should gracefully alert the user regarding the bad or malicious input	Pass/Fail
Crawl the Web site using tools like WebScarab.	The tool shouldn't be able to generate the Web resources tree unless the user performing this activity is authenticated to do so.	Pass/Fail

As illustrated in test case 1 (Table 5-8), the field username is populated with the malicious input containing HTML tags. The string `"Scott' #"` is used to inject SQL and the remaining part of the malicious input is a simple JavaScript in order to inject XSS in the application. This scenario is depicted in Figure 5-9.

The intended outcome of providing valid username and password field is to display the user's home page with a greeting message to the user. Since the input to the username field contains the malicious input, it is interpreted by the browser as directed to open an unintended content (in this case, http://www.google.com) within an iframe. This scenario is depicted in Figure 5-10.

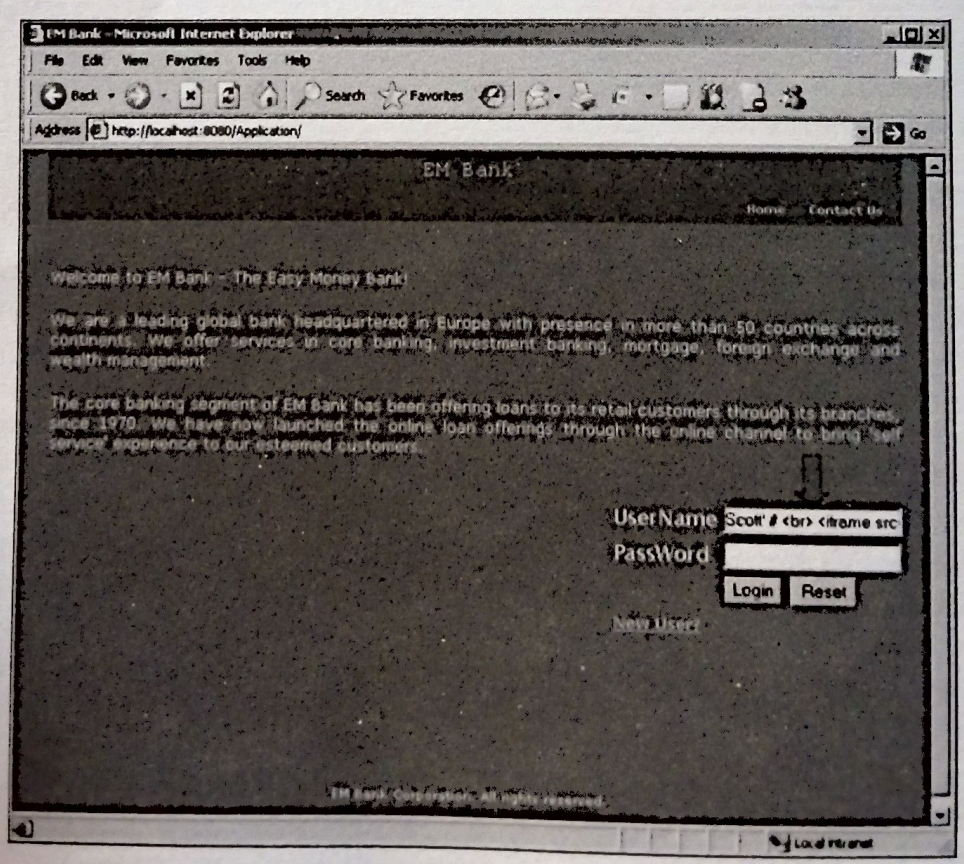


Figure 5-9 SQL and XSS injection demonstration.

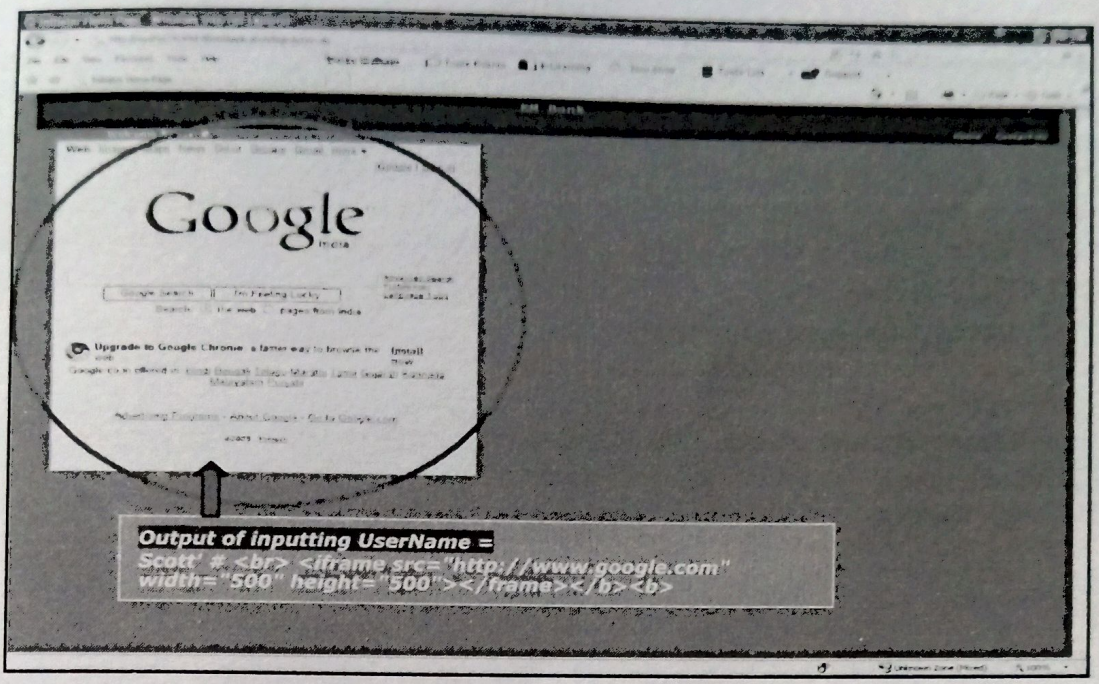


Figure 5-10 Application security vulnerability—unearthed.

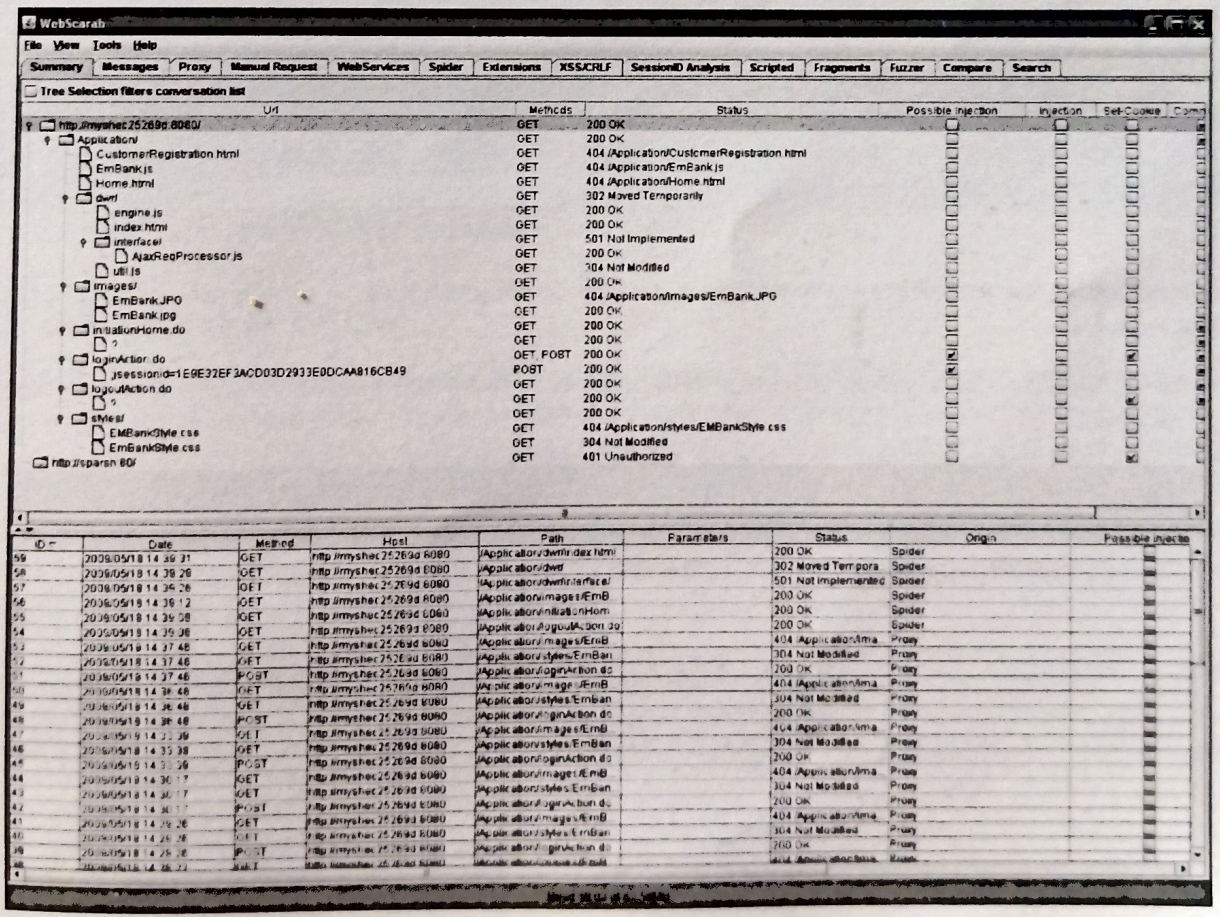


Figure 5-11 Web layer resource tree—outcome of crawling.

As stated in test case 2, the WebScarab tool is used to crawl over the Web layer resources after authentication as a user who is restricted from accessing them. The Web layer resource tree is generated by the tool, as depicted in Figure 5-11, which means that the Web layer resources are not secure. In such a case, any un-authenticated user or a user with inadequate privileges is able to download static resources from the site by typing their address in the browser's address bar.

Various coding best practices are exercised to fix security vulnerabilities. To avoid XSS injection, measures like implementing a security filter for a servlet and escaping HTML characters are used. To avoid SQL injection, usage of prepared statements is advised. The security vulnerabilities and the measures taken to curb them were discussed in Chapter 4.

Xp

Penetration testing tools should be selected based on factors like how accurately they report security vulnerabilities, the extent to which they can be customized and configured, availability of the tool documentation and other necessary support for tool.

5.4.3 Usability Testing

Usability testing is a form of black-box testing of the enterprise application to validate the usability requirements typically mentioned as part of the nonfunctional requirements. This type of testing is primarily useful to ensure that the enterprise application is easy to understand and use. Usability testing is especially important for B2C applications. This is normally performed either by the end users themselves or alternatively by business analysts who represent end users.

Xp

Often usability is misinterpreted as performance of the system. Performance is tested separately as part of performance testing. Together, usability and performance can contribute significantly to end user productivity.

Let us look at the usability requirements stated for LoMS in Chapter 2, which are reproduced below for easy reference:

- LoMS application requires customer and prospects to be self-guided through the loan initiation process.
- For bank representatives, the system should facilitate easy navigation combined with an easy workflow system.
- Since EM Bank is a global bank, the LoMS application needs to be internationalized.
- Context-sensitive help should be provided to a user.

In

Demographics of users needs to be considered in testing the accessibility features of the system. For example, if a significant of user population is above 40 years of age, then it may be useful to design the user interface with larger fonts.

Usability testing has test cases based on the user behavior and nature of use. Considering the requirements stated above, a part of the usability test plan prepared is as shown in Table 5-9.

Table 5-9 Usability test cases

Test case	Expected result	Result
Loan initiation. Test whether the user is self-guided through the whole loan initiation process	Simple and easy-to-use loan initiation process with ease of navigation to complete the initiation process. Already available customer information for loan processing should be pre-populated	Pass/Fail
Loan approval. System should have a simple work flow from initiation to approval and disbursement of loan	Workflow should be completed in a maximum of three steps	Pass/Fail
Globalization. Multi-lingual support	Site to be multi-lingual and all messages and forms are displayed in the correct local language	Pass/Fail
Help. System should provide an adequate help facility for both customers and bank representatives	System should have form level help, tips, process help available based on context.	Pass/Fail
Personalization. System should display the information relevant to the customer	Personalized information displayed to the customer	Pass/Fail

The success of an enterprise application depends on its acceptance by the end users which, in turn, depends primarily on the usability characteristics. Hence, usability testing becomes one of the important steps in system testing. Significance of usability testing increases when the system exposure to the outside world increases. For example, the system would be tested for usability more rigorously in a B2C environment compared to B2B or in-house applications.

5.4.4 Globalization Testing

Enterprises are reaching out to users across the globe, which mandates applications that can be localized per the user needs to ensure customer friendliness. *Globalization testing* is a manual testing to validate whether an enterprise application meets the internationalization requirements. It also validates whether the application can be localized based on a given locale without any architectural or design change. Globalization (G11N) is the term typically used for a combination of internationalization (I18N) and localization (L10N), and is a key aspect of the usability of an application. This testing is typically performed by test engineers and business analysts, who are aware of the language of choice, by running the application in different locales.

Globalization testing involves testing of various things related to data and content of the enterprise application. Language translations, writing direction, currency symbols, units of measures, date and time format, time zones, number formatting, fonts, sizes, colors, images, collation and ordering are the predominant things which are typically validated as part of this testing. Based on the LoMS globalization requirements, a sample of the globalization test cases is depicted in Table 5-10.

Table 5-10 Globalization test cases

Test case	Expected result	Result
Login as a bank representative using locale different from the default one, and sort the customers on customer first name	The sort order should be correct based on the locale selected	Pass/Fail
Login as a customer and select the locale different from the default one, and check the date and currency format	The date and currency format should be correct based on the locale selected	Pass/Fail
Login as a customer and select the locale different from the default one, and check the language translation on customer page	The language of the form fields and other form content should be correctly translated based on the locale selected	Pass/Fail

Xp

In practice, the rollout of a globalized application typically happens in several phases based on the successful completion of the globalization test suites for each locale. A single universal rollout of the application across all the required locales may be difficult to achieve due to considerations such as availability of language experts for all the locales.

5.4.5 Interface Testing

Interface testing validates the incoming and outgoing interfaces of the application under test with respect to all other applications that it interacts with. It is also known as *intersystem testing*, which validates the coexistence of the application under test with other enterprise applications.

Objectives of interface testing are dependent on several factors such as nature of communication of interfaces, request-response handling of the system, data formats exchanged through the interfaces, types of validations performed and nature of errors and exceptions that could arise. In synchronous interfaces, the response time is a major consideration as against in asynchronous interfaces.

Xp

During interface testing, especially synchronous ones, the system interfaces should not be just validated from the exchanged content or data perspective. Optimum performance need to be validated by performing performance test on the participating system interface for the end to end scenarios.

The entry criteria for interface testing are the readiness and correct documentation of all the participating systems in the interface testing. The documentation should properly document the data passed between the systems. Based on the LoMS interface requirements, a sample interface test case is shown in Table 5-11.

Interface testing involves a mix of manual and automated techniques and is typically a collaborative effort by multiple teams responsible for different systems whose interfaces are integrated with the

Table 5-11 Interface test cases

Test case	Expected result	Result
Login as customer and check the pre-approved loan amount at the time of loan initiation	The pre-approved loan amount will be made available to the customer at the time of loan initiation by credit rating system	Pass/Fail

in-bound and/or out-bound interfaces of the system under test. As presented in Chapter 3, there are various integration mechanisms and technologies to integrate enterprise applications. Each one of them has its own testing mechanisms.

Interoperability and platform independence are the need of almost all enterprise applications of modern times. To facilitate these, interfaces based on Web services constitute a considerable share of system interfaces of enterprise applications. Testing system interfaces developed using Web services is different in several ways, as it involves validating security, interoperability, platform independence, etc. Lack of a user interface also poses a few challenges in testing interfaces developed using Web services.

Tt

SoapUI is one of the open source Web services testing tool to test SOAP and REST based Web services. It provides features like Web service inspection, Web service invocation, load testing, compliance testing, Web service simulation and mocking among other features.

5.5 User Acceptance Testing

User acceptance testing is a black-box testing of an enterprise application performed by the end users of the system, primarily from the perspective of system's stated functionalities and usability. This is the last stage of testing before the application rollout, and is required to certify the readiness of enterprise application to go live in production. The team to certify the "acceptance" of the enterprise application typically represents a cross-section of the end users of the application. This testing is mostly manual and is carried out in the staging environment.

The entry criteria for user acceptance testing are successful completion of unit, integration and system testing. The bugs identified during these testing should either be fixed or documented as "known issues". The objectives of a typical user acceptance testing are to validate:

- The stated functionalities in the system requirement specification document
- Part of nonfunctional requirements like usability and response time
- The access control of the modules of the application
- The complete business processes
- The business data from the outgoing and incoming interfaces
- Batch processes, if any
- The report format and contents

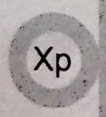
Xp

The acceptance criteria for the enterprise application should be worked out with the end user in the early life cycle phase of the application development.

User acceptance testing plan is prepared by taking inputs from the system requirement specification document. A part of the user acceptance testing plan for LoMS is presented in Table 5-12.

Table 5-12 User acceptance test cases

Test case	Expected result	Result
Login as an existing customer and check the pre-approved loan amount	Pre-approved loan amount should be displayed based on credit rating of the customer	Pass/Fail
Login as a prospect and try to initiate loan	Prospect should not be able to initiate the loan	Pass/Fail
Login as an existing customer and check the fonts, colors, toolbars, tooltips, text wrap, static content on the customer home page	The customer home page should be correct in content and consistent with the overall presentation of the enterprise application	Pass/Fail
Login as a bank representative and upload the signed loan application	The contents and format of signed loan application should be appropriate and as per the EM Bank's guidelines	Pass/Fail



The members of UAT team should be trained using manual and training aids to get acquainted with the features of enterprise application under test, before the commencement of user acceptance test.

A controlled and organized user acceptance test plan and test cases help in keeping testing focus on the important features of the enterprise application and gain end user confidence. A successful user acceptance test sets the stage for the acceptance of the application by the end user community. By this time, the enterprise application is all set to be rolled out in the production environment.

5.6 Rolling Out Enterprise Applications

Rolling out enterprise applications means making the application live on the production environment for the end user. This is the time when the enterprise applications are handed over to the enterprise application maintenance team. Release engineers facilitate the rolling out procedure of enterprise applications. There are a few high-level strategies to be considered for rolling out enterprise applications:

- **Brand new rollout.** This rollout strategy is for the first roll out of a brand new application. This rollout is possibly the easiest to manage among all other rollout strategies.
- **Parallel switch.** The parallel switch rollout strategy is to bring up a new application environment in parallel. The new environment is verified, and application traffic is switched from the old to the new environment. The old environment is kept on standby for a predetermined period. This rollout strategy has nil to minimal downtime.
- **Rolling rollout.** The rolling rollout strategy is applicable in a clustered and highly managed environment. In this strategy, a node of the cluster is taken offline, upgraded, verified, brought

to online status, and then the next node follows suite. This is not applicable to all types of systems, as it poses challenges in terms of transaction integrity and data continuity of the enterprise applications.

- **Phased rollout.** Phased rollout is a slower variant of the rolling rollout strategy. The rollout is phased out on the basis of particular region or time.
- **Up-down-up rollout.** This rollout strategy starts with bringing everything down. The system is upgraded, verified and brought up again. This strategy is the most time consuming and requires a lot of upfront planning. This is the safest rollout strategy for large-scale deployments.

Whatever be the selected strategy to rollout enterprise applications, the following are the key prerequisites for a successful application rollout:

- The rollout plan that should consist of sequence of rollout related events, rollout schedules, key personnel along with their contact information and the escalation mechanism
- Communication plan for rollout that describes who gets involved at what stage, handshake protocol for handover from one group to another and escalation protocol
- Certification plan to ensure that everything is working which is necessary for rollout activities; release notes, scripts, configuration files, installable packages, etc., should all be in place
- Plan B or the rollback plan which is required if things do not go well as per the rollout plan; the plan should also contain the impact in money terms, brand image and the impact on customer for every hour of delay during the rollout
- Identify the best time and duration for launching the rollout of enterprise applications
- Depending on the criticality of the rollout, a dress rehearsal can be done before roll out, possibly in the test or staging environment. Dress rehearsal is similar to a smoke test. *Smoke testing* is a cursory check of the crucial pieces of the enterprise application without getting into the finer details of them

Once the rollout strategy is finalized and the team is ready to roll out the enterprise application, the actual rollout of application takes place. Although steps to rollout the application may vary, depending on type of deployment, a typical rollout goes through the following steps:

1. Make sure all the communication systems like emails, instant messengers, pagers, cell phones, war room phones are working.
2. Identify and verify the binaries, configuration files, scripts and artifacts to be used in the application deployment. Typically a quality analyst should have all these tagged and labeled.

Xp

Do not check out code from version control systems for the production rollout. Everything should be prebuilt, verified and readily available on the target systems prior to starting the rollout activity.

3. Run backups and setup recovery points.
4. Turn OFF monitoring, alerting and alarm notification systems. Increase queue depths and cache ratios, if you expect traffic to pile up for subsequent processing after the system is back online.

Tt

BMC Patrol, Tivoli monitors, Queue threshold monitors, pager notification systems, etc., are a few examples of monitoring, alerting and alarm notification systems.