Shivani®

Educating People®

Complete Book®

For

Engineering Students

July 2020

As Per New
Scheme & Syllabus
(AICTE Flexible Curricula)

New Edition

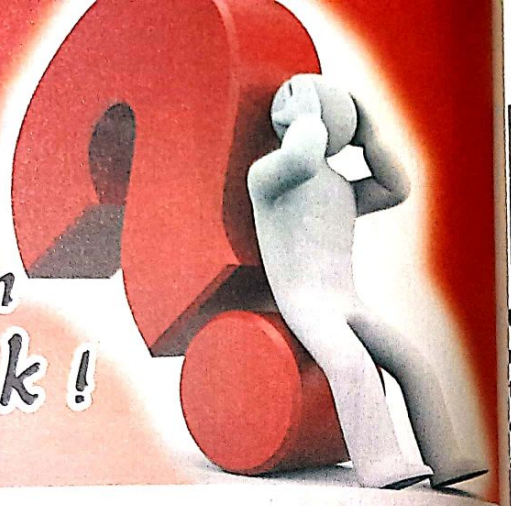
VII Semester

R.G.P.V. Examination Papers Completely Solved

Software Architectures

Smart Accurate Comprehensive

About Preparation of this Book !



The Present Edition of this book SOFTWARE ARCHITECTURES Stands out distinctly from others on account of the following salient features.

- This book Covers Complete New Syllabus as Prescribed by R.G.P.V. Bhopal.
- This book is according to New Scheme & Syllabus of Examinations.
- This book is thoroughly revised with a view to making it more student friendly.
- This book covers each and every topic in lucid and simple language with questions and answer form with easy solutions.
- This book has been presented on Teach Yourself technique without assuming any prior knowledge of the subject.
- This book has been presented with essentially elementary approach along with some Special tips on important topics and Diagrams.
- This book includes Complete Topics organised in to increasing degree of complexity, Nos of Numerical Problems with easy solution.
- All Questions set at examinations of R.G.P.V. Bhopal are included Chapter-wise with Full Solutions/Answers.
- We have referred to a number of books on SOFTWARE ARCHITECTURES before writing of this book. Still we would whole heartely accept suggestions for improvement offered by the reader. We hope this book will meet all the requirements of the readers and come up to their expectations.

Syllabus

SOFTWARE ARCHITECTURES

Unit - 1 : Overview of software development methodology and software quality model, different models of software development and their issues. Introduction to software architecture, evolution of software architecture, software components and connectors, common software architecture frameworks, architecture business cycle, architectural patterns, reference model.

Unit - 2 : Software architecture models : structural models, framework models, dynamic models, process models. Architectures styles : dataflow architecture, pipes and filters architecture, call and return architecture, data-centered architecture, layered architecture, agent based architecture, micro-services architecture, reactive architecture, representational state transfer architecture etc.

Unit - 3 : Software architecture implementation technologies : Software Architecture Description Languages (ADLs), struts, Hibernate, Node JS, Angular JS, J2EE – JSP, Servlets, EJBs; middleware: JDBC, JNDI, JMS, RMI and CORBA etc. Role of UML in software architecture.

Unit - 4 : Software architecture analysis and design : requirements for architecture and the life-cycle view of architecture design and analysis methods, architecture based economic analysis : Cost Benefit Analysis Method (CBAM), Architecture Tradeoff Analysis Method (ATAM). Active Reviews for Intermediate Design (ARID), Attribute Driven Design Method (ADD), architecture reuse, Domain-specific software architecture.

Unit - 5 : Software architecture documentation : principles of sound documentation, refinement, context diagrams, variability, software interfaces. Documenting the behavior of software elements and software systems, documentation package using a seven-part template.

Price : Rs. 100.00 (Rs. One Hundred Only)

Edition : 2020

Contents

SOFTWARE ARCHITECTURES

UNIT 1

OVERVIEW OF SOFTWARE DEVELOPMENT AND ARCHITECTURE

OVERVIEW OF SOFTWARE DEVELOPMENT METHODOLOGY AND SOFTWARE QUALITY MODEL, DIFFERENT MODELS OF SOFTWARE DEVELOPMENT AND THEIR ISSUES

Q.1. Discuss the software development methodology.

Ans. Architecture development and its component, structural modeling, are members of a family of methodologies used in a defined, repeatable, improvable software development process. A methodology should spell out general steps to follow. It should be specific enough to give guidance but be general enough to apply to most software situations. It should not be taken as a step-by-step way to develop entire systems; these recipes for how to get the work done simply do not exist. Management needs to realize that a directive to "use Schlaer & Mellor" or "use a structural model" has about the same content as a directive to "use an oscilloscope".

The systems approach to software development concentrates on the total system over its whole lifecycle. It addresses quality characteristics, methods, and standards, and provides a roadmap that integrates them into the whole. Fig. 1.1 illustrates this concept. These components address all the considerations needed for software development. The lifecycle describes the phases in which software development takes place. Quality characteristics define the attributes that the software must exhibit in order to reach the software goals. The methods are the procedures employed in software development. The standards are used to guide and evaluate the software development process.

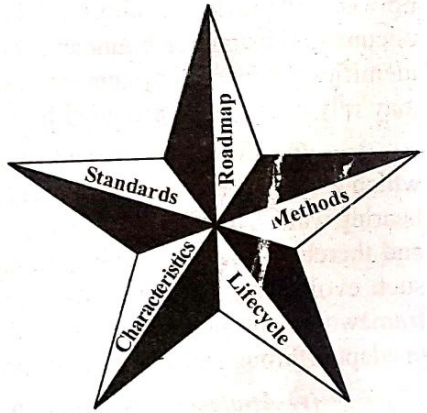


Fig. 1.1 Components of a Software Development Methodology

The software standards component of the methodology provides for consistency in the software development process. Each of the standards addresses the style and layout of the code. These standards are used both for a guide as well as a review tool. The following elements expound the standards that a defined methodology might commonly include. They can be treated as a minimal requirements set for a software design methodology. The generalized elements include (a) structural model, (b) data structure model, (c) coding standard, and (d) verification standard. It is imperative that each standard be followed exactly and enforced during each design review. Then at the end of the program these guides will be the most important documents in the maintenance of the software system.

Q.2. How can software architecture play an important role in software development ?

Ans. Software architecture can play an important role in at least six aspects of software development as follows –

(i) Understanding – Software architecture simplifies our ability to comprehend large systems by presenting them at a level of abstraction at which a system's high-level design can be easily understood. Moreover, at its best, architectural description exposes the high-level constraints on system design, as well as the rationale for making specific architectural choices.

(ii) Reuse – Architectural design supports reuse of both components and also frameworks into which components can be integrated. Domain-specific software architectures, frameworks, platforms and architectural patterns are various enablers for reuse, together with libraries of plugins, add-ins and apps.

(iii) Construction – An architectural description provides a partial blueprint for development by indicating the major components and dependencies between them. For example, a layered view of an architecture typically documents abstraction boundaries between parts of a system's implementation, identifies the internal system interfaces, and constrains what parts of a system may rely on services provided by other parts.

(iv) Evolution – Architectural design can expose the dimensions along which a system is expected to evolve. By making explicit a system's "load-bearing walls", maintainers can better understand the ramifications of changes, and thereby more accurately estimate costs of modifications. In many cases such evolution and variability constraints are manifested in product lines, frameworks and platforms, which dictate how the system can be instantiated or adapted through the addition of application-specific features and components.

(v) Analysis – Architectural descriptions provide opportunities for analysis, including system consistency checking, conformance to constraints imposed by an architectural style, satisfaction of quality attributes, and domain-specific analyses for architectures built in specific styles.

(vi) Management – For many companies the design of a viable software architecture is a key milestone in an industrial software development process. Critical evaluation of an architecture typically leads to a much clearer understanding of requirements, implementation strategies, and potential risks, reducing the amount of rework required to address problems later in a system's lifecycle.

Q.3. Write short note about software quality models.

Ans. Software quality models can be valuable tools for software engineering of embedded system, because some software-enhancement techniques are so expensive or time consuming that it is not practical to apply them to all modules. Targeting such enhancement techniques is an effective way to reduce the likelihood of faults discovered in the field.

A software quality model is developed using measurements and fault data from a past release. The calibrated model is then applied to modules currently under development. Such models yield predictions on a module by module basis.

Q.4. Explain the waterfall model. ***(R.G.P.V., June 2011)***

Ans. The waterfall model or the classic life cycle is sometimes called the linear sequential model. It suggests a systematic approach to software development that begins at the system level and progresses through analysis, design, coding, testing, and support. The principal stages of the model as shown in fig. 1.2 are explained as follows –

(i) Requirements Analysis and Definition – The system's services, constraints and goals are established by consultation with system users. They are then defined in detail and serve as a system specification.

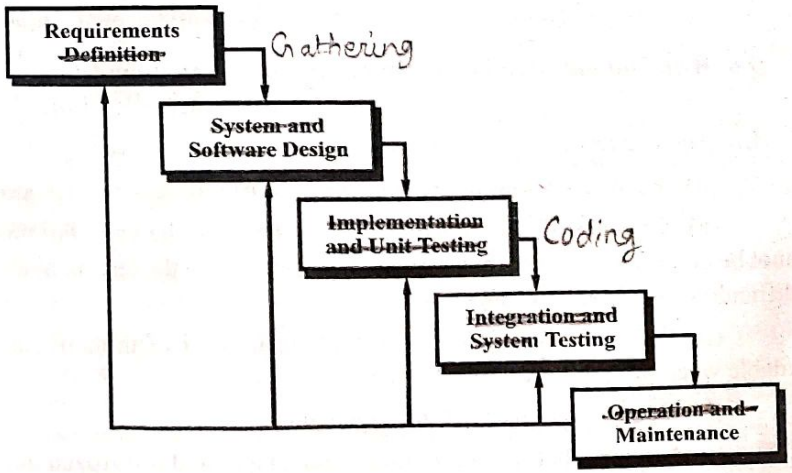


Fig. 1.2 Waterfall Model

(ii) System and Software Design – The systems design process partitions the requirements to either hardware or software systems. It establishes an overall system architecture. Software design involves identifying and describing the fundamental software system abstractions and their relationships.

(iii) Implementation and Unit Testing – During this stage the software design is realized as a set of programs or program units. Unit testing involves verifying that each unit meets its specifications.

(iv) Integration and System Testing – The individual program units or programs are integrated and tested as a complete system to ensure that the software requirements have been met. After testing, the software system is delivered to the customer.

(v) Operation and Maintenance – Normally this is the longest life-cycle phase. The system is installed and put into practical use. Maintenance involves correcting errors which were not discovered in earlier stages of the life cycle, improving the implementation of system units and enhancing the system's services as new requirements are discovered.

Q.5. What are the advantages of the waterfall model ?

Ans. There are following advantages of the waterfall model –

(i) Relatively simple to understand.

(ii) Each phase of development proceeds sequentially.

(iii) Allows managerial control where a schedule with deadlines is set for each stage of development.

(iv) Helps in controlling schedules, budgets and documentation.

Q.6. Write out the reasons for the failure of waterfall model.
(R.G.P.V., June 2016)

Ans. The reasons for the failure of waterfall model are given below –

(i) Requirements need to be specified before the development proceeds.

(ii) Changes of requirements in later phases of the waterfall model cannot be done. This implies that once an application is in the testing phase, it is difficult to incorporate changes.

(iii) No user involvement and working version of the software is available when the software is developed.

(iv) Does not involve risk management.

(v) Assumes that the requirements are stable and are frozen across the project span.

Q.7. What is prototype model ? Under what circumstances it is beneficial to construct a prototype model ? Does the construction of a prototype model always increase the overall cost of software development ?
(R.G.P.V., June 2003, Dec. 2015)

Ans. As shown in fig. 1.3, the prototype model begins with requirements gathering. Developer and customer meet and define the objectives of the software, identify the requirements known and outline areas where further definition is mandatory. A "quick design" then occurs. The quick design focuses on a representation of those aspects of the software that will be visible to the customer/user (e.g., input approaches and output formats). The quick design leads to the construction of a prototype. The prototype is then evaluated by the customer and used to refine requirements for the software to be developed. The prototype serves as a mechanism for identifying software requirements. If a working prototype is built, the developer attempts to use existing program fragments or applies tools that enable working programs to be generated quickly.

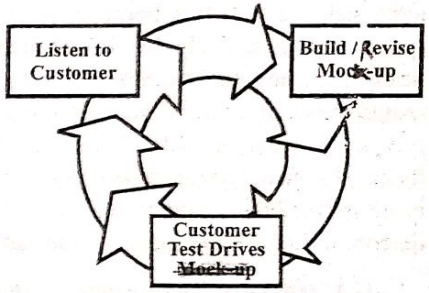


Fig. 1.3 Prototype Model

Although having some problems in its implementation, prototyping can be an effective model for software engineering. For having its effectiveness, one should define the rules in the beginning i.e., both the customer and the developer must agree that the prototype is built as a mechanism for defining requirements. Afterwards it is discarded and the actual required software can be engineered having an eye toward quality and maintainability.

When your customer has a legitimate need but is clueless about the details, develop a prototype as a first step.

Though it is not cost effective, yet it is useful as it helps to build a software whose complete requirements are not specified or is clueless. It is the prototype that helps to identify software requirements with an eye towards quality and maintainability.

For prototyping for the purposes of requirement analysis to be feasible its cost must be kept low. Consequently, only those features are included in the prototype that will have a valuable return from the user experience. Exception handling, recovery and conformance to some standards and formats are typically not included in prototypes. In prototyping, as the prototype is to be discarded, there is no point in implementing those parts of the requirements that are already well understood. Hence, the focus of the development is to include those features that are not properly understood.

Prototyping is often not used, as it is feared that development costs may become large. However, in some situations, the cost of software development without prototyping may be more than with prototyping. There are two major reasons for this. First, the experience of developing the prototype might reduce the cost of the later phases when the actual software development is done. Secondly, in many projects the requirements are constantly changing, particularly when development takes a long time. We saw earlier that changes in requirements at a later stage of development substantially increase the cost of the project. By elongating the requirements analysis phase, the requirements are "frozen" at a later time, by which time they are likely to be more developed and consequently, more stable. In addition because the client and users get experience with the system, it is more likely that the requirements specified after the prototype will be closer to the actual requirements. This again will lead to fewer changes in the requirements at a later time. Hence, the costs incurred due to changes in the requirements may be substantially reduced by prototyping. Hence, the cost of the development after the prototype can be substantially less than the cost without prototyping. Prototyping is well suited for projects requirements are hard to determine and the confidence in the stated requirements is low.

Q.8. What are the advantages and disadvantages of prototype model ?

Ans. The various advantages and disadvantages associated with the prototype model are as follows –

Advantages –

(i) Provides a working model to the user early in the process, enabling early assessment and increasing user confidence.

(ii) The developer gains experience and insight by developing a prototype, thereby resulting in better implementation of requirements.

(iii) The prototyping model serves to classify requirements, hence reducing ambiguity and improving communication between the developer and the user.

(iv) There is a great involvement of users in software development. Hence, the requirements of the users are met to the greatest extent.

(v) Helps in reducing risks associated with the project.

Disadvantages –

(i) If the user is not satisfied with the developed prototype, then a new prototype is developed. This process goes on until a satisfactory prototype evolves. Thus, this model is time-consuming and expensive.

(ii) The developer loses focus of the real purpose of prototype and comprises on the quality of product.

(iii) Prototyping can lead to false expectations. It often creates a situation where the user believes that the development of the system is finished when it is not.

(iv) The primary goal of prototyping is rapid development. Thus, the design of the system may suffer as it is built in a series of layers without considering integration of all the other components.

Q.9. Explain RAD model. Write different drawbacks of RAD model. ***(R.G.P.V., June 2015)***

Or

Write a short note on RAD model. ***(R.G.P.V., June 2005, Dec. 2009)***

Ans. Rapid application development is an incremental software development process model that has extremely short development cycle. It is a high speed version of the linear sequential model in which rapid development is achieved by using component based construction. If requirements are well understood and project scope is constrained, the RAD process enables a development team to create a fully functional system within 60 to 90 days.

The phases of RAD approach are (see fig. 1.4) –

(i) Business modeling (ii) Data modeling

(iii) Process modeling (iv) Application generation

(v) Testing and turnover.

(i) Business Modeling – The information flow among business functions is modeled in a way that answers some questions as – what information is required ? What information is generated ? Who generates it ? etc.

(ii) Data Modeling – The information flow defined as part of business modeling phase is refined into data objects to support business.

(iii) Process Modeling – The data objects defined in the data modeling phase are transformed to achieve the information flow necessary to implement a business function.

(iv) Application Generation – RAD assumes the use of fourth generation techniques to facilitate construction of software. Rather than creating software using conventional third generation programming languages the RAD process works to reuse existing program components (when possible) or create reusable components (when necessary). In all cases, automated tools are used to facilitate construction of the software.

(v) Testing and Turnover – Since the RAD process emphasized reuse, many of the program components have already been tested. This reduces overall testing time.

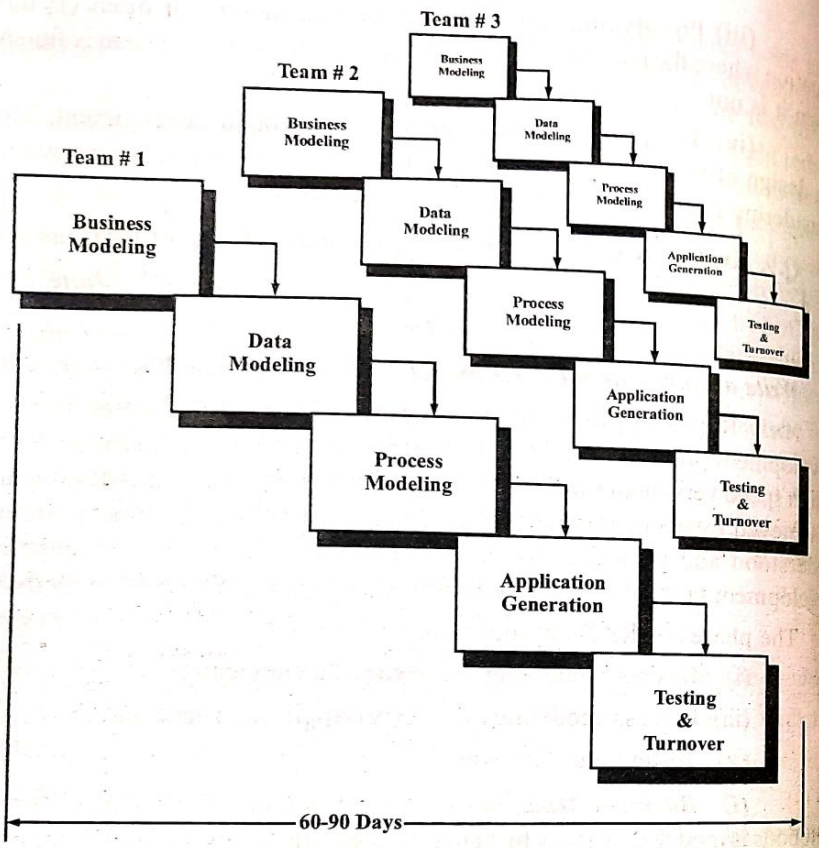


Fig. 1.4 The RAD Model

Advantages –

(i) Deliverables are easier to transfer as high-level abstractions, scripts, and intermediate codes are used.

(ii) Provides greater flexibility as redesign is done according to the developer.

(iii) Results in reduction of manual coding due to code generators and code reuse.

(iv) Encourages user involvement.

(v) Possibility of lesser defects due to prototyping in nature.

Disadvantages –

(i) Useful for only larger projects.

(ii) RAD projects fail if there is no commitment by the developers or the users to get the software completed on time.

(iii) Not appropriate when technical risks are high. This occurs when the new application utilizes new technology or when new software requires a high degree of interoperability with existing system.

(iv) As the interests of users and developers can diverge from single iteration to the next, requirements may not converge in RAD model.

Q.10. Explain increamental model in detail. ***(R.G.P.V., June 2016)***

Ans. The incremental model combines elements of the linear sequential model with the iterative philosophy of prototyping. As shown in fig. 1.5, the incremental model applies linear sequences in a staggered fashion as calendar time progresses. Each linear sequence produces a deliverable "increment" of the software. When an incremental model is used, the first increment is often a core product. That is basic requirements are addresssed, but many supplementary features remain undelivered. The core product is used by the customer. As a result of use and/or evaluation, a plan is developed for the next increment. The plan addresses the modification of the core product to better meet the needs of the customer and the delivery of additional features and functionality. This process is repeated following the delivery of each increment, until the complete product is produced. For example, word-processing software developed using the incremental paradigm might deliver basic file management, editing and document production functions in the first increment, more sophisticated editing and document production capabilities in the second increment, spelling and grammar checking in the third increment and advanced page layout capability in the fourth increment. The process flow for any increment can incorporate the prototyping paradigm.

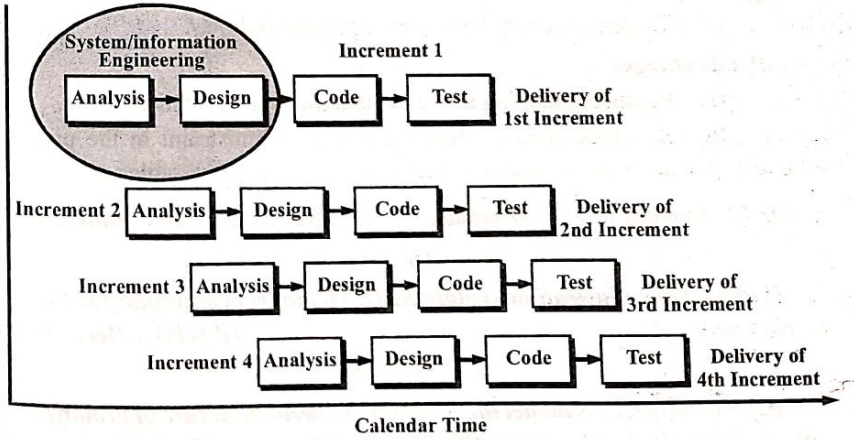


Fig. 1.5

The incremental process model, like prototyping and other evolutionary approaches, is iterative in nature. But unlike prototyping, the incremental model focuses on the delivery of an operational product with each increment. Early increments are stripped down versions of the final product, but they do provide capability that serves the user and also provide a platform for evaluation by the user. Incremental development is particularly useful when staffing is unavailable for a complete implementation by the business deadline that has been established for the project. Early increments can be implemented with fewer people. If the core product is well received, then additional staff can be added to implement the next increment. In addition, increments can be planned to manage technical risks.

Q.11. What are the advantages and disadvantages of the incremental model ?

Ans. There are following advantages and disadvantages of the incremental model –

Advantages –

(i) Avoids the problems resulting in risk-driven approach in the software.

(ii) Understanding of the problem increases through successive refinements.

(iii) Performs cost-benefit analysis before enhancing software with capabilities.

(iv) Incrementally grows in effective solution after each multiple iteration.

(v) Does not involve a high complexity rate.

(vi) Early feedback is generated, because implementation occurs rapidly for a small sub-set of the software.

(vii) There is a low risk of overall project failure.

Disadvantages –

(i) Requires planning at the management and technical level.

(ii) Becomes invalid when there is time constraint in the project schedule or when the users cannot accept the phased deliverables.

Q.12. Explain the spiral model. ***(R.G.P.V., Dec. 2002, June 2011)***

Or

With suitable illustration explain SPIRAL model evolutionary software development. ***(R.G.P.V., Dec. 2010)***

Or

Explain the process model that couples the iterative nature of prototyping with the controlled and systematic aspects of the waterfall model. ***(R.G.P.V., Dec. 2017)***

Ans. Boehm proposed a recent model for software development process known as the ***spiral model***. It is an evolutionary software process model that couples the iterative nature of prototyping with the controlled and systematic aspects of the waterfall model. All the activities in this model can be organized as a spiral which has many cycles, as shown in fig. 1.6.

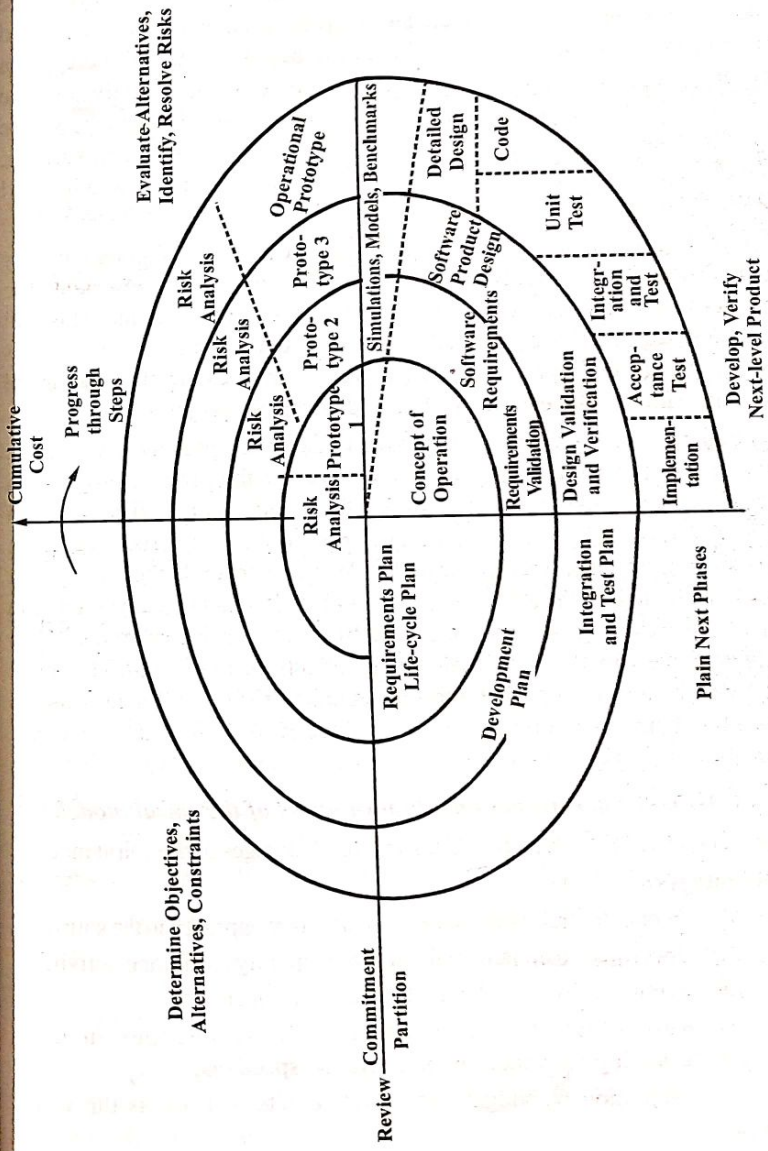


Fig. 1.6 The Spiral Model

In this model, the radial dimension represents the cumulative cost incurre in accomplishing the steps done so far and the angular dimension represen the progress made in completing each cycle of the spiral. Each cycle in th model begins with the identification of objectives for that cycle, the differe alternatives that are possible for achieving the objective and constraints tha are exist. It is the upper left or first quadrant of the cycle. In the cycle, th next step is to evaluate these different alternatives based on the objectives an constraints. In this step, focus of evaluation is based on the risk perceptio for the project. Risks reflect, the chances that some of the objectives of th project may not be met. To develop strategies is the next step that resolve th uncertainties and risks and it may involve activities such as benchmarkin simulation and prototyping. Next, the software is developed, keeping in min the risks. Then finally the next stage is planned.

The risk-driven nature of the spiral model allows it to accommodate an mixture of a specification-oriented, prototype-oriented, simulation-oriented o some other type of approach. The most important feature of the model is tha each cycle of the spiral is completed by a review that covers all the produc developed during that cycle including plans for the next cycle. The spira model works for development as well as enhancement projects.

The spiral model is a realistic approach to the development of large-scal systems and software. Because software evolves as the process progresse the developer and customer better understand and react to risks at eac evolutionary level. The spiral model uses prototyping as a risk reductio mechanism but, more important, enables the developer to apply the prototypin approach at any stage in the evolution of the product. It maintains the systemati stepwise approach suggested by the classic life cycle but incorporates it into an iterative framework that more realistically reflects the real world. The spira model demands a direct consideration of technical risks at all stages of the project and, if properly applied, should reduce risks before they become problematic.

Q.13. Write the advantages and disadvantages of the spiral model.

Ans. There are following advantages and disadvantages of the spiral model-

Advantages –

(i) Avoids the problems resulting in risk-driven approach in the software.

(ii) Specifies a mechanism for software quality assurance activities.

(iii) Is utilized by complex and dynamic projects.

(iv) Re-evaluation after each step allows changes in user perspectives, technology advances or financial perspectives.

(v) Estimation of budget and schedule gets realistic as the work progresses.

(vi) The spiral model is a realistic approach to the development of large scale systems and software.

(vii) This model reduces risk.

Disadvantages –

(i) Assessment of project risks and its resolution is not an easy task.

(ii) Difficult to estimate budget and schedule in the beginning, as some of the analysis is not done until the design of the software is developed.

INTRODUCTION TO SOFTWARE ARCHITECTURE, EVOLUTION OF SOFTWARE ARCHITECTURE, SOFTWARE COMPONENTS AND CONNECTORS, COMMON SOFTWARE ARCHITECTURE FRAMEWORKS

Q.14. What do you understand by software architecture.

Ans. Software architecture represents the overall software structure and the ways in which that structure offers conceptual system integrity for a system. Generally, architecture is the hierarchical program components.

Bass, Clements, and Kazman define software architecture in the following way –

The software architecture of a program or computing system is the structure or structures of the system, which comprise software components, the externally visible properties of those components, and the relationships among them.

One aim of software design process is to derive an architectural rendering of a system which provides a framework from which more detailed design activities are performed.

Following are the set of properties that should be specified as part of an architectural design –

(i) Structural Properties – These are the properties which define the system components and the way those components are packaged and interact.

(ii) Extra-functional Properties – These properties of an architectural design address how the design architecture achieves requirements for performance, reliability, adaptability, capacity, security, and other system characteristics.

(iii) Families of Related Systems – The architectural design should draw upon repeatable patterns found in the design of families of related systems. In short, the design should be able to reuse architectural building blocks.

Q.15. What does a good software architecture look like ?

Ans. Some points of a good software architecture are as follows –

(i) A good architecture is rational. It should promote and suppor repeatable and improvable process for building out a specific member of product family.

(ii) A good architecture is affordable. It must be "efficient enoug in both time and memory. It must support large-scale cost and sched improvements in both the short term and the long term. And it must ha been defined, published, and demonstrated to work in order to reduce ris

(iii) A good architecture takes into account the complete domain the problem. It must address visibility, interprocessor communications, tim and memory requirements, frame balancing and processor balancing, an testing and debugging.

(iv) A good architecture is consistent and enforces an interfac contract between loosely coupled component models. It should permit subsystems to be developed independent of the source of the inputs and th destination of the outputs. It should allow new implementations of systems be integrated into existing specifications.

(v) A good architecture encourages early development. In the cas of data voids, systems for which data is missing can be stubbed, and th interface specification defines what must be known later about the system

(vi) A good architecture promotes system understanding. It mus "look like" the problem space in some significant sense. It must be clear, an it must clearly meet both user and end customer requirements. Its quality an style should match what are considered sound systems and softwar engineering principles.

(vii) A good architecture is a good citizen. It should not violat company or customer standards. It should be broadly accepted or acceptable in the community. It should be available in the public domain rather than being bound to a proprietary hardware or software system. And it must take advantage of military and international standards like the Ada programming language and ISO communications protocols.

Q.16. Enumerate the important properties of software architecture.

Ans. Software architecture is the high-level structure of a software system. The important properties of software architecture are as follows –

(i) It is at a high-enough level of abstraction that the system can be viewed as a whole.

(ii) The structure must support the functionality required of the system. Thus, the dynamic behaviour of the system must be taken into account when designing the architecture.

(iii) The structure or architecture must conform to the system qualities.

(iv) At the architectural level, all implementation details are hidden.

Q.17. What are the important elements of software architecture ? Explain.

Ans. The important elements of software architecture are as follows –

(i) Meta Architecture – The architectural vision, style, principles, key communication and control mechanisms and concepts that guide the team of architects in the creation of the architecture.

(ii) Architectural Views – Just as building architecture are best envisioned in terms of a number of complementary views or models, so too are software architectures, and these include structural views, behavioural views and execution views.

(a) Structural Views – These help document and communicate the architecture in terms of the components and their relationships and are useful in assessing architectural qualities like extensibility.

(b) Behavioural Views – These views are especially useful in assessing run-time qualities such as performance and security. These views also useful in thinking through how the components interact to accomplish their assigned responsibilities and evaluating the impact of what-if scenarios on the architecture.

(c) Execution Views – These help in evaluating physical distribution options and documenting and communicating decisions.

(iii) Architectural Patterns – Structural patterns such as layers and client/server, and mechanisms such as brokers and bridges.

(iv) Architecture Design Principles – The architectural design principles involve abstraction, postponing decisions, separation of concerns and simplicity.

(v) System decomposition principles

(vi) Good interface design.

Q.18. What are the major benefits of software architecture ?

Ans. The major benefits of software architecture are as follows –

(i) Development – It is important to be able to recognize common paradigms so that high-level relationships among systems can be better understood and so that new systems can be built as variants of old systems.

An architectural representation is often essential to the analysis and description of the high-level properties of a complex system.

(ii) Maintenance – Documenting a system's structure and properties in a rigorous way has obvious advantage to maintenance.

In addition, retaining the designer's intentions about system organization should help maintainers preserve the systems design integrity.

(iii) Optimization – Software architecture supports optimization of components w.r.t. e.g. performance, fault tolerance and security concerns.

(iv) Communication – Communication among stakeholders based on an explicit description of high-level abstractions of the system under development.

(v) Early Design Decisions – These influenced by driving quality attributes.

Q.19. Describe the objectives of software architecture.

Ans. The objectives of software architecture are as follows –

(i) Customer Concern – This concern involves schedule and budget estimation, feasibility and risk assessment, progress tracking and requirement traceability.

(ii) User Concern – This concern includes consistency with requirements and usage scenarios, future requirement growth accommodation, performance, reliability and interoperability.

(iii) Architect Concern – This concern includes requirements traceability, support of trade off analyses, completeness and consistency of architecture.

(iv) Developer Concern – This concern includes sufficient detail for design, reference for selecting components and maintain interoperability with existing systems.

(v) Maintainer Concern – This concern includes guidance on software modification and architecture evolution, maintain interoperability with existing systems.

Q.20. Write the role and responsibilities of a software architect.

Ans. The software architect role is new enough that there is considerable debate about what constitutes the role. A simplistic view of the role is that software architects create software architectures, and their responsibilities encompass all that is involved in doing so.

Few major roles of software architect includes the following –

(i) Articulating the architectural vision.

(ii) Conceptualising and experimenting with alternative architectural approaches.

(iii) Creating models and component and interface specification documents.

(iv) Validating the architecture against requirements and assumptions.

Q.21. Explain the evaluation of software architecture.

Ans. Architecture evaluations can be performed in one or more stages of a software development process. They can be used to compare and identify strengths and weaknesses in different architecture alternatives during the early design stages. They can also be used for evaluation of existing systems before future maintenance or enhancement of the system as well as for identifying architectural drift and erosion. Software architecture evaluation methods can be divided into four main categories. Methods in the categories can be used independently but also be combined to evaluate different aspects of a software architecture, if needed.

1) Experience based evaluations are based on the previous experience and domain knowledge of developers or consultants. People who have encountered the requirements and domain of the software system before can based on the previous experience say if a software architecture will be good enough.

2) Simulation-based evaluations rely on a high-level implementation of some or all of the components in the software architecture. The simulation can then be used to evaluate quality requirements such as performance and correctness of the architecture. Simulation can also be combined with prototyping, thus prototypes of an architecture can be executed in the intended context of the completed system.

3) Mathematical modelling uses mathematical proofs and methods for evaluating mainly operational quality requirements such as performance and behaviour of the components in the architecture. Mathematical modelling is similar to simulation and can be combined with simulation to more accurately estimate performance of components in a system.

4) Scenario-based architecture evaluation tries to evaluate a particular quality attribute by creating a scenario profile which forces a very concrete description of the quality requirement. The scenarios from the profile are then used to go through the software architecture and the consequences are documented.

Q.22. Define the terms components and connectors.

Ans. **Components** – It represent the primary computational elements and data stores of a system. Intuitively, they correspond to the boxes in box-and-line descriptions of software architectures. Typical examples of components include such things as clients, servers, filters, objects, blackboards, and databases. In most ADLs components may have multiple interfaces, each interface defining a point of interaction between a component and its environment.

Connectors – It represent interactions among components. Computationally speaking, connectors mediate the communication and coordination activities among components. That is, they provide the "glue"

for architectural designs, and intuitively, they correspond to the lines in box-and-line descriptions. Examples include simple forms of interaction, such as pipes, procedure call, and event broadcast. But connectors may also represent more complex interactions, such as a client-server protocol or a SQL link between a database and an application. Connectors also have interfaces that define the roles played by the various participants in the interaction represented by the connector.

Systems represent configurations (graphs) of components and connectors. In modern ADLs a key property of system descriptions is that the overall topology of a system is defined independently from the components and connectors that make up the system. (This is in contrast to most programming language module systems where dependencies are wired into components via import clauses). Systems may also be hierarchical – components and connectors may represent subsystems that have "internal" architectures.

Q.23. How are components and connectors are related to software architecture ? Justify your answer.

Ans. Software architecture is commonly defined in terms of components and connectors. Components are identified and assigned responsibilities that client components interact with through "contracted" interfaces. Component interconnections specify communication and control mechanisms, and support all component interactions needed to accomplish system behaviour.

Q.24. Describe the common software architecture frameworks.

Ans. Architecture structure can be divided into three parts as described below –

(i) Module Structure – Here the elements are modules, which are units of implementation modules represent a code-based way of considering the system. They are assigned areas of functional responsibility. There is less emphasis on how the resulting software manifests itself at runtime.

Module-based structures include the following –

(a) Decomposition – The units are modules related to each other by the "is a submodule of" relation, showing how larger modules are decomposed into smaller ones recursively until they are small enough to be easily understood.

(b) Class – The module unit in the structure are called classes. The class structure allows us to reason about re-use and the incremental addition of functionality.

(c) Uses – The units are related by the uses relation. One unit uses another if the correctness of the first requires the presence of a correct version (as opposed to a stub) of the second.

(d) Layered – Layers are often designed as abstractions (virtual machines) that hide implementation specifies below from the layers above, engendering portability.

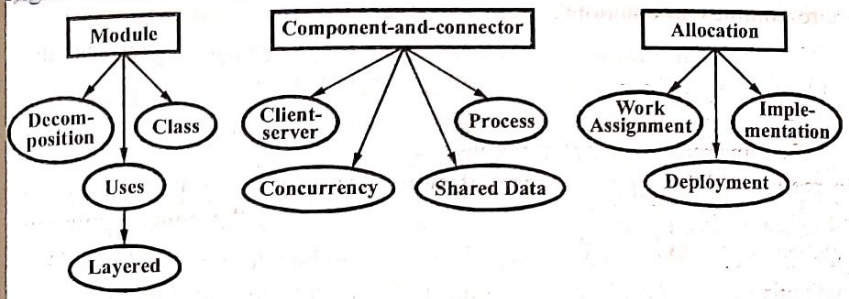


Fig. 1.7 Common Software Architecture Structures

(ii) Component and Connector Structures – Here the elements are runtimes (which are the principal units of computation) and connector (which are the communication vehicles among components). These structures include the following –

(a) Process or Communicating Processes – The units here are processes or threads that are connected with each other by communication, synchronization, and/or exclusion operations.

(b) Concurrency – The concurrency structure is used early in design to identify the requirements for managing the issues associated with concurrent execution.

(c) Shared Data or Repository – This structure comprises components and connectors that create, store, and access persistent data.

(d) Client-server – This is useful for separation of concerns (supporting modifiability), for physical distribution, and for load balancing (supporting runtime performance).

(iii) Allocation – Allocation structures show the relationship between the software elements and the elements in one or more external environments in which the software is created and executed.

These structures include the following –

(a) Deployment – This view allows an engineer to reason about performance, data integrity, availability, and security.

(b) Implementation – This is critical for the management of development activities and builds processes.

(c) Work Assignment – This structure assigns responsibility for implementing and integrating the modules to the appropriate development teams.

ARCHITECTURE BUSINESS CYCLE, ARCHITECTURE PATTERNS, REFERENCE MODEL

Q.25. Explain the architecture business cycle (ABC) with suitable diagram.

Ans. The model of the architecture business cycle (ABC) is based on the assumption that "software architecture is the result of technical, business and social influences". The resulting architecture "in turn affects the technical, business and social environments". The key elements of the cycle are the forces influencing the architecture, the requirements that result from these forces, the architect and his experience, the architecture and the system (or systems in a product line architecture). The architecture business cycle also shows how these key elements influence each other, seen in fig. 1.8.

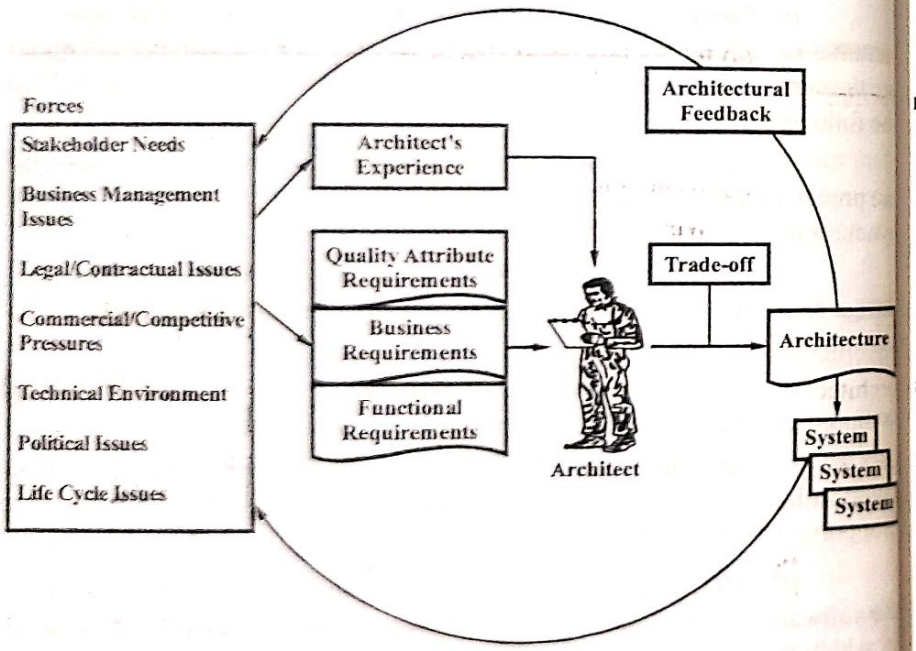


Fig. 1.8 The Architecture Business Cycle

In a later report the originators clarified the purpose; "...the architecture business cycle was envisioned as a means to depict the influences on a software architect and to show how architectures can eventually influence the very things that originally shaped them".

The influences of the original cycle have been updated by the original authors in and are subsequently called forces in. This study is based on the latest of these updated architecture business cycles, since the seven categories

of forces, seen in fig. 1.8, shaping the architecture was easier to relate to the interview responses.

The main idea of the cycle, that the architecture provides feedback in turn affecting one or more of the original influences or forces, have remained the same through all evolutions of the architecture business cycle. The cycle is often used as a theoretical framework, but it is hard to find empirical studies involving the actual stakeholders and not only as an observation of an architecture business cycle from a distance.

Q.26. Discuss the software process and the architecture business cycle.

Ans. Software process is the term given to the organization, ritualization, and management of software development activities.

The various activities involved in creating software architecture are –

(i) Creating the Business Case for the System –

(a) It is an important step in creating and constraining any future requirements.

(b) How much should the product cost ?

(c) What is its targeted market ?

(d) What is its targeted time to market ?

(e) Will it need to interface with other systems ?

(f) Are there system limitations that it must work within ?

(g) These are all the questions that must involve the system's architects.

(h) They cannot be decided solely by an architect, but if an architect is not consulted in the creation of the business case, it may be impossible to achieve the business goals.

(ii) Understanding the Requirements –

(a) There are a variety of techniques for eliciting requirements from the stakeholders.

For example, object oriented analysis uses scenarios, or "use cases" to embody requirements. Safety-critical systems use more rigorous approaches, such as finite-state-machine models or formal specification languages.

(b) Another technique that helps us understand requirements is the creation of prototypes.

(c) Regardless of the technique used to elicit the requirements, the desired qualities of the system to be constructed determine the shape of its structure.

(iii) Creating or Selecting the Architecture – In the landmark book Mythical Man-Month, Fred Brooks argues forcefully and eloquently that conceptual integrity is the key to sound system design and that conceptual integrity can only be had by a small number of minds coming together to design the system's architecture.

(iv) Documenting and Communicating the Architecture –

(a) For the architecture to be effective as the backbone of the project's design, it must be communicated clearly and unambiguously to all of the stakeholders.

(b) Developers must understand the work assignments it requires of them, testers must understand the task structure it imposes on them, management must understand the scheduling implications it suggests, and so forth.

(v) Analyzing or Evaluating the Architecture –

(a) Choosing among multiple competing designs in a rational way is one of the architect's greatest challenges.

(b) Evaluating an architecture for the qualities that it supports is essential to ensuring that the system constructed from that architecture satisfies its stakeholders needs.

(c) Use scenario-based techniques or architecture tradeoff analysis method (ATAM) or cost benefit analysis method (CBAM).

(vi) Implementing the System based on the Architecture –

(a) This activity is concerned with keeping the developers faithful to the structures and interaction protocols constrained by the architecture.

(b) Having an explicit and well-communicated architecture is the first step toward ensuring architectural conformance.

(vii) Ensuring that the Implementation Conforms to the Architecture –

(a) Finally, when an architecture is created and used, it goes into a maintenance phase.

(b) Constant vigilance is required to ensure that the actual architecture and its representation remain to each other during this phase.

Q.27. Explain the working of architecture business cycle.

Ans. Relationships among business goals, product requirements, architect experience, architectures and fielded systems form a cycle with feedback loops that a business can manage. A business manages this cycle to handle growth, to expand its enterprise area, and to take advantage of previous investments in architecture and system building.

Fig. 1.9 shows the feedback loops. Some of the feedback comes from the architecture itself, and some comes from the system built from it.

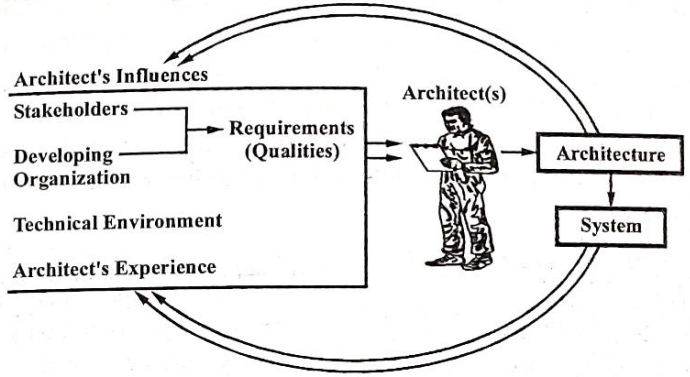


Fig. 1.9 The Architecture Business Cycle

The architecture affects the structure of the developing organization. An architecture prescribes a structure for a system it particularly prescribes the units of software that must be implemented and integrated to form the system. Teams are formed for individual software units; and the development, test, and integration activities around the units. Likewise, schedules and budgets allocate resources in chunks corresponding to the units. Teams become embedded in the organization's structure. This is feedback from the architecture to the developing organization.

The architecture can affect the goals of the developing organization. A successful system built from it can enable a company to establish a foothold in a particular market area. The architecture can provide opportunities for the efficient production and deployment of the similar systems, and the organization may adjust its goals to take advantage of its newfound expertise to plumb the market. This is feedback from the system to the developing organization and the systems it builds.

The architecture can affect customer requirements for the next system by giving the customer the opportunity to receive a system in a more reliable, timely and economical manner than if the subsequent system were to be built from scratch.

The process of system building will affect the architect's experience with subsequent systems by adding to the corporate experience base.

A few systems will influence and actually change the software engineering culture. i.e., the technical environment in which system builders operate and learn.

Q.28. What is architecture patterns ? Explain.

Ans. The compositions have been found useful over time, and over many different domains, and so they have been documented and disseminated. These compositions of architectural elements, called architectural patterns, provide packaged strategies for solving some of the problems facing a system.

An architectural pattern delineates the element types and their forms of interaction used in solving the problem. Patterns can be characterized according to the type of architectural elements they use. For example, a common module type pattern is this –

(i) Layered Pattern – When the uses relation among software elements is strictly unidirectional, a system of layers emerges. A layer is a coherent set of related functionality. In a strictly layered structure, a layer can only use the services of the layer immediately below it. Many variations of this pattern, lessening the structural restriction, occur in practice. Layers are often designed as abstractions (virtual machines) that hide implementation specifics below from the layers above, engendering portability.

Common component-and-connector type patterns are these –

(ii) Shared-data (or Repository) Pattern – This pattern comprises components and connectors that create, store, and access persistent data. The repository usually takes the form of a (commercial) database. The connectors are protocols for managing the data, such as SQL.

(iii) Client-server Pattern – The components are the clients and the servers, and the connectors are protocols and messages they share among each other to carry out the system's work.

Common allocation patterns include the following –

(iv) Multi-tier Pattern – Multi-tier pattern, which describes how to distribute and allocate the components of a system in distinct subsets of hardware and software, connected by some communication medium. This pattern specializes the generic deployment (software-to-hardware allocation) structure.

(v) Competence Center and Platform – These are patterns that specialize a software system's work assignment structure. In competence center, work is allocated to sites depending on the technical or domain expertise located at a site. For example, user-interface design is done at a site where usability engineering experts are located. In platform, one site is tasked with developing reusable core assets of a software product line and other sites develop applications that use the core assets.

Q.29. What do you mean by layers pattern ? Also describe their benefits and issues.

Ans. Refer to Q.28 (i).

Some examples of this pattern are, networking protocols, in the Java virtual machine, the application in Java consists of instructions for the Java virtual machine; the JVM uses services from the operating system underneath.

Benefits – This pattern has the following benefits –

(i) A lower layer can be used by different higher layers. The TCP layer from TCP/IP connections, for instance, can be reused without changes by various applications, such as telnet or FTP.

(ii) Layers make standardisation easier, clearly defined and commonly accepted levels of abstraction make it possible to develop standardised tasks and interfaces.

(iii) Dependencies are kept local. When a layer shows the agreed interface to the layer above, and expects the agreed interface of the layer below, changes can be made within the layer without affecting other layers. This means a developer can test particular layers independently of other layers, and can develop them independently as well – this supports development by teams.

Issues in the Layers Pattern – The most stable abstractions are in the lower layer, a change in the behaviour of a layer has no effect on the layer below it. The opposite is true as well, a change in the behaviour of a lower layer has an effect on the layers above it, so this should be avoided.

Of course, changes in or additions to a layer without an effect on behaviour will not affect the layers above it. Layer services can therefore be implemented in different ways (think of the bridge pattern here, where a dynamic link is maintained between abstraction and implementation).

Layers can be developed independently. However, defining an abstract service interface is not an easy job. There may also be performance overhead due to repeated transformations of data. Furthermore, the lower layers may perform unnecessary work that is not required by the higher layers.

Q.30. What do you mean by client-server pattern ? Also describe their issues.

Ans. In the client-server pattern as shown in fig. 1.10, a server component provides services to multiple client components. A client component requests services from the server component. Servers are permanently active, listening for clients.

The requests are sent beyond process and machine boundaries. This means that some inter-process communication mechanism is required – clients and servers may reside on different machines, and thus in different processes. In fact, you can see the client-server pattern as a variant of the layered pattern, crossing process or machine boundaries – clients form the higher level and the server forms the lower level.

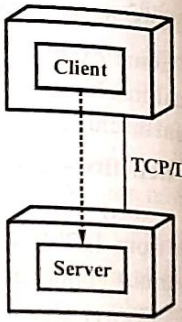


Fig. 1.10 Client-server Pattern

Examples of the client-server pattern are remote database access (client applications request services from a database server), remote file systems (client systems access files, provided by the server system, applications access local and remote files in a transparent manner) or web-based applications (browsers request data from a web server).

Issues in the Client-server Pattern – Requests are typically handled in separate threads on the server.

Inter-process communication causes overhead. Requests and result data often have to be transformed or marshalled because they have a different representation in client and server and because there is network traffic.

Distributed systems with many servers with the same function should be transparent for clients – there should be no need for clients to differentiate between servers. When you type in the URL for Google, for instance, you should not have to know the exact machine that is accessed (location transparency), the platform of the machine (platform transparency), the route your request travels and so on. Intermediate layers may be inserted for specific purposes – caching, security or load balancing, for instance.

Sometimes, callbacks are needed for event notification. This can also be seen as a transition to the Peer-to-Peer pattern.

Q.31. What do you mean by master slave pattern ? Also describe their issues.

Ans. The master-slave pattern as shown in fig. 1.11 supports fault tolerance and parallel computation. The master component distributes the work among identical slave components and computes a final result from the results the slaves return. Fig. 1.12 shows a sequence diagram of a master distributing work between slaves.

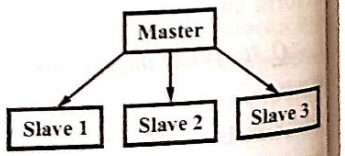


Fig. 1.11 Master-slave Pattern

The Master-slave pattern is applied, for instance, in process control, in embedded systems, in large-scale parallel computations and in fault-tolerant systems.

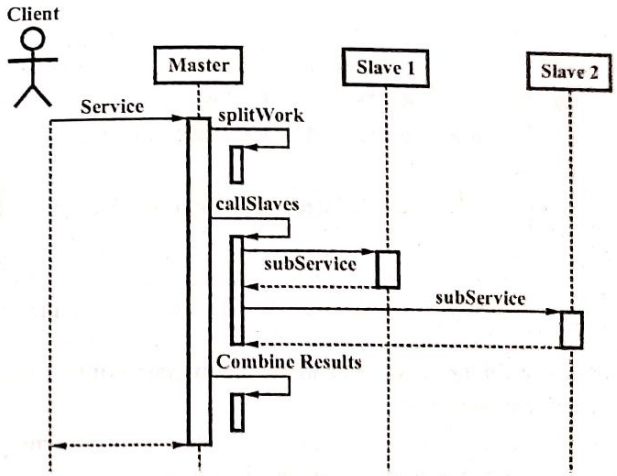


Fig. 1.12 Sequence Diagram for the Master-slave Pattern

Examples – One application area for the master-slave pattern is fault tolerance – the master delegates the job to be done to several slaves, receives their results and applies a strategy to decide which result to return to the client. One possible strategy is to choose the result from the first slave that terminates. Another strategy is to choose the result that the majority of slaves have computed. This is fail-proof as far as the slaves are concerned (the master can provide a valid result as long as not all slaves fail), but not with respect to the master. Failure of slaves may be detected by the master using time-outs. Failure of the master means the system as a whole fails. Another application area is parallel computing. A third application area is that of computational accuracy.

Issues in the Master-slave Pattern – The Master-slave pattern is an example of the divide-and-conquer principle. In this pattern, the aspect of coordination is separated from the actual work – concerns are separated. The slaves are isolated – there is no shared state. They operate in parallel.

The latency in the master-slave communication can be an issue, for instance in real-time systems – master and slaves live in different processes.

The pattern can only be applied to a problem that is decomposable.

Q.32 Explain the relation among the reference model, architectur patterns and reference architecture.

Ans. An architectural pattern is a description of element and relation type together with a set of constraints on how they may be used.

For example, client-server is a common architectural pattern. Client and server are two element types, and their coordination is described in terms of the protocol that the server uses to communicate with each of its clients.

A reference model is a division of functionality together with data flow between the pieces.

A reference model is a standard decomposition of a known problem into parts that cooperatively solve the problem.

A reference architecture is a reference model mapped onto software elements (that cooperatively implement the functionality defined in the reference model) and the data flows between them. Whereas a reference model divides the functionality, A reference architecture is the mapping of that functionality onto a system decomposition.

The relationships of reference models, architecture patterns, reference architectures and software architectures is shown in fig. 1.13.

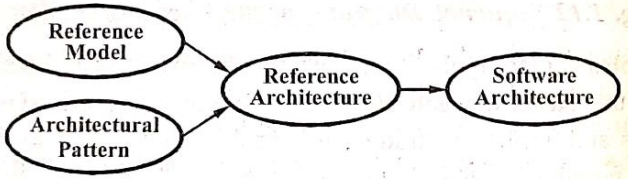


Fig. 1.13

These reference models, architectural patterns, and reference architectures are not architectures; they are useful concepts that capture elements of an architecture. Each is the outcome of early design decisions. The relationship among these design elements is shown in fig. 1.13. A software architect must design a system that provides concurrency, portability, modifiability, usability, security, and the like, and that reflects consideration of the tradeoffs among these needs.

⌘⌘

UNIT 2

SOFTWARE ARCHITECTURE MODELS & STYLES

SOFTWARE ARCHITECTURE MODELS – STRUCTURAL MODELS, FRAMEWORK MODELS, DYNAMIC MODELS, PROCESS MODELS

Q.1. Write short note on software architecture model.

Ans. There are five different types of models used to represent the architectural design. These are – ***structural models***, which represent architecture as a well organized collection of program components; ***framework models***, which enhance the design abstraction level by trying to identify repeated patterns found in similar applications; ***dynamic models***, which address the behavioural view of the program architecture, thus indicating how the structure or system configuration may change externally; ***process models***, which emphasize on the business design or the design of technical process that the system must accommodate; and finally ***functional models***, which can be used to represent the functional hierarchy of a system. Various ***architectural description languages (ADLs)*** have been developed to represent these models.

Q.2. Discuss the concept of structural model.

Ans. A structural model is the architectural map for a large software system or family of systems (domain). The structural model used in a domain represents the point of convergence for trade-offs between maintainability and performance, quality and efficiency. As such, different domains will likely have different structural models. The idea of a structural model evolved out of the Ada Simulator Validation Program (ASVP), which established the efficacy of Ada for real-time training simulation. The most important of these standards is the idea of a structural model standard. A structural model is new to the software process and falls directly out of a systems engineering process as it is applied to Ada software development.

The structural model is the framework through which components, attributes, and inter-relationships within the system are expressed. The structural model enforces a consistency in the software structure, thus aiding

understanding. The structural model does not need to be confused with structural analysis. Structural analysis is a process of software component definition and refinement. In contrast, the structural model becomes an important part of the architecture in which the results of the structural analysis are implemented.

An architecture, as we intend to use the term, consists of (i) a partitioning strategy and (ii) a coordination strategy. The partitioning strategy leads to dividing the entire system into discrete, non-overlapping parts or components. The coordination strategy leads to explicitly defined interfaces between those parts. These two strategies provide an engineering approach to bridging the gap between the system as a whole (as represented by its specification) and the design (the plan to build the product from primitive parts, such as computer instructions, metal struts, and switches). The structural model specifies –

(i) The kinds of entities that will exist in the design (How do you package?).

(ii) How the real world is mapped into the software entities (What's in a package?).

(iii) The communication between entities (How do packages communicate?).

Q.3. Explain the domain architecture for reuse in training system (DARTS) structural model.

Ans. Domain architecture for reuse in training system (DARTS) is the software architecture/system architecture we have applied to the air vehicle training systems (AVTS) domain. The AVTS domain is a family of air vehicle training devices that provides the simulation, stimulation, and/or emulation of all the components and systems for a real-time air vehicle training systems. This domain encompasses the systems necessary to provide training devices that a trainee uses to become familiar with the operator station configuration and/or flight characteristics of the application air vehicle, gain proficiency in executing normal procedures, recognizing malfunctions/abnormal indications and executing the corresponding standard/emergency procedures, and executing mission procedures. The devices within this domain are each made up of some subset of the domain segments or sub-domains.

DARTS includes both a body of systems engineering work (common content) and a structural model (a common form for components). We will not be discussing the former any further, except to note that a body of pre-existing systems engineering work has major possibilities for reuse which we are in the process of demonstrating under the STARS program.

The structural model for DARTS is shown in fig. 2.1. The DARTS structural model contains five major structural elements – the virtual network,

the module executive(s) the segment executives, the subsystem controllers, and the components.

(i) Virtual Network – The Virtual Network (VNET) is the means by which the other structural elements of the DARTS structural model communicate with one another.

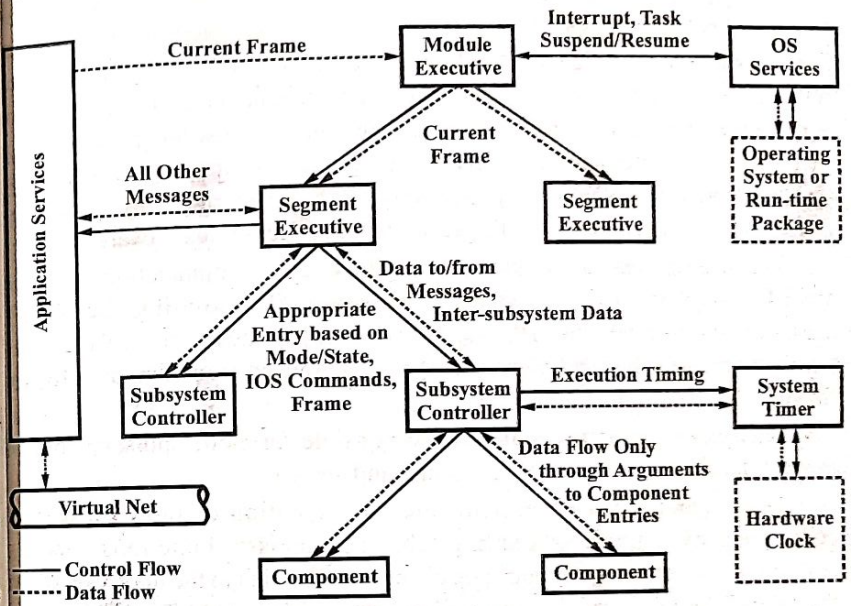


Fig. 2.1 Domain Architecture for Reuse in Training Systems (DARTS) Overview

(ii) Module Executive – A module is a computational system with associated resources such as storage devices, displays, network interfaces, and so on. However abstract we'd like our structural model to be, at the end of the day it has to run on a real computer. Multiple-CPU computers are generally modules, the rule being "do you have to communicate with the other CPU(s) as though they were in different boxes?" (e.g., over a network). The purpose of the module executive is to cause the lower-level design elements to execute.

There is one module executive for every module. All operating system and hardware dependent functions such as interrupt, task suspend and resume, and so on, are located here. A module executive running on brand X computers may look quite different from a module executive running on brand Y computers, though of course its functionality will be the same.

The module executive "causes" the segment executives to execute. This is kept deliberately ambiguous, since the right way of doing this on a given

program may be to call the segment executives as subprograms, or it may to schedule their execution as independent tasks. Because data flow betwe the module executive and segment executives is one-way and small (the clo tick message passes from the module executive to its segment), it does n stand in the way of implementing the right choice for a program.

(iii) Segment Executives – A segment is a major grouping functionality. Early in the Mod Sim program, it was determined that so functions and objects "go together" in the sense that (a) there are major d flows between them, and (b) there are order dependencies between the Subsystems and components which go together in this sense are grouped int segment. These functions and objects were gathered together into 12 segmen and this represents some of the pre-done systems engineering work that mak DARTS a robust candidate for the development of reusable software.

The segment executives are responsible for all communications over i VNET (apart from the clock tick message). By isolating the VNE communications functions in the segment executives, the lower-level eleme (subsystem controller and component) may be reused from similar softwa for other architectures.

The segment executives are also responsible for mode and state contr logic (total freeze, reposition, run mode, and so on).

The segment executives schedule the execution of their subsyste controllers by using a scheduling table mechanism. Functions such malfunction insertion and mode/state change are handled in the main executi thread in DARTS – a mode or state change message in DARTS is a messa like any other, though it is processed by the segment executives.

The segment executives in DARTS call upon the appropriate aperiod entries in each of the subsystem controllers, based on the receipt of t appropriate control messages through the VNET.

Segment executives execute a large amount of highly predictable cod If a segment is defined as the sender of a message in our interface specs, th it will identify itself to the VNET as a sender of that message, send th message, and so on.

(iv) Subsystem Controllers – Subsystems originally correspond to the functions allocated to segments in the Mod Sim architecture, which the request of the customer was a traditional functional decomposition. Sin that time, we have done additional work on abstracting the objects in t system in a more object-focused or object-abstracted methodology.

Subsystem controllers are implemented as in the air vehicle structur model (AVSM). In the AVSM data flows out of subsystems through a shar memory based export area, while in DARTS (largely because of t

correspondence between subsystems and interface messages) the segment executive provides the subsystem controller with data from messages and builds messages to send to the VNET.

The subsystem controller is at the middle of the DARTS hierarchy. Its entry points are standardized. Every subsystem controller has the same entry points. This does not mean that all entry points will be used in every subsystem controller.

(v) Components – As in the AVSM, the lowest level element identified in the architecture is called the component. Each of the components corresponds to an object in the OOA sense. In DARTS, as in the AVSM, all knowledge about the operation and state of components is contained within the components. And no knowledge about the external environment (simulation control commands, presence or absence of other components, computational environment) is contained within the components. Components compute the state of the objects they simulate in a purely abstract, and therefore reusable, manner. These rules, which are among the most attractive features of the AVSM and DARTS structural models, comprise "knowledge firewalls".

Just as in the AVSM, all data flow between components takes place through the subprogram calls for each of the entries in the components. As the SEI points out, this set of entries is both necessary and sufficient to permit the knowledge firewalls described above to operate.

Q.4. What are the advantages and disadvantages of DARTS ?

Ans. Some advantages of DARTS are as follows –

(i) The subsystem controllers and components are based on reusable templates. Every subsystem looks like every other subsystem and every component looks like every other component, in that they have the same subprogram entries and the same package structure.

(ii) Components are structured to be widely reusable. Since components have no knowledge of their environments, they should be reusable in the widest possible context.

(iii) Delivery is more predictable, since the components are all nameable and locatable very early in the program. Each of the subsystems and components can be tracked from a very early date in the program, so that reaction to delays and data voids have lower impact.

(iv) DARTS is designed to permit segments to be easily subcontracted. Companies with expertise in visual systems, electronic warfare, or weapons but which have little or no training system experience can compete to build an individual or set of appropriate segments. The ability of segments to be tested as stand-alone entities lowers both prime and subcontractor risk at acceptance.

(v) As requirements change, within a range of simulators, or follow-ons require more or less computational power, DARTS permits ne zero-effort addition or deletion of segments and of computational pow allocated to a segment. Segments may be moved from one module to anoth again with near-zero effort. When this change is anticipated, a hardwa architecture can be chosen for the affected segments that permits the simp plug-replacement of CPUs with less powerful or more powerful CPUs.

(vi) Interfaces between segments are strictly specified in compilab Ada. Adaptation of these reusable interfaces to the requirements of a specif program is accomplished by the decision model for the domain, and we ha demonstrated that it is easy to automate this process.

Disadvantages of DARTS are as follows –

(i) DARTS, like the AVSM, absolutely requires data flow contr which takes engineering hours. Our slogan has been, "If you want to contr data flow, you've got to control data flow". The generic, adaptable interfa specification provides a great deal of help in this control process, and utiliti associated with DARTS automatic much of the tedious coding work li message setup and connection.

(ii) DARTS allocates systems functional requirements to subsyster and permits object or functional decomposition of components. O-O purist may see this as a red flag.

(iii) DARTS is domain-specific. As such, organizations working different domains may not realize the full benefits of DARTS.

Q.5. Write short note on frameworks model.

Ans. Frameworks are a central concept of large scale object orient software development. They promise increased productivity, short development times, and higher quality of application.

A framework is a class model, together with an integration role type se and a builds-on class set. A framework covers one particular domain or significant aspect thereof. It is a coherent unit of reuse, both by use relationships and by extension through subclassing.

The integration role type set determines how the framework is to be use by use-relationship based clients. It contains those role types of the class model which have not been assigned to classes, and which must be defined by clie classes so that their instances can make use of objects from the framework runtime. An element of an integration role type set is called an integration ro type. A role model, which provides an integration role type, is called a integration role model.

The builds-on class set specifies the classes of frameworks the curre framework builds on. To build on another framework, the current framewo

assigns some or all of the role types from the other framework's integration role type set to its classes. The builds-on class set comprises those classes of other frameworks that define role types from role models, in which integration role types are involved that are used by the current framework.

Q.6. What do you understand by dynamic model ?

Ans. Dynamic modelling describes those aspect of the system that are concerned with time and sequencing of the operations. It is used to specify and implement the control aspect of the system. Dynamic model is represented graphically with the help of state diagrams. It is also known as state modelling. State model consist of multiple state diagrams, one for each class with temporal behaviour that is important to an application. State diagram relates with events and states. Events represents external functional activity and states represents values objects. The dynamic model is used to express and model the behaviour of the system over time. It includes support for activity diagrams, state diagrams, sequence diagrams and extensions including business process modelling.

(i) Sequence Diagrams – Sequence diagrams are used to display the interaction between users, screens, objects and entities within the system. It provides a sequential map of message passing between objects over time. Frequently these diagrams are placed under use cases in the model to illustrate the use case scenario-how a user will interact with the system and what happens internally to get the work done. Often, the objects are represented using special stereotyped icons, as in the example below. The object labelled login screen is shown using the user interface icon. The object labelled security manager is shown using the controller icon. The object labelled users is shown using the entity icon.

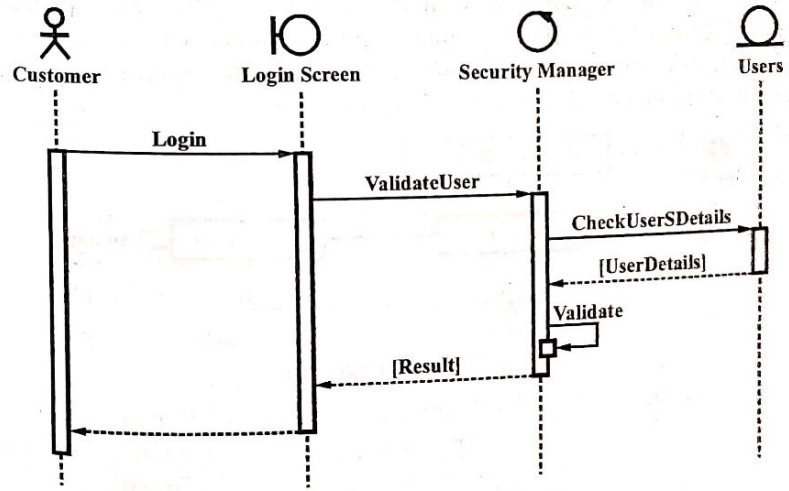


Fig. 2.2 Sequence Diagram

(ii) Activity Diagrams – An activity diagram is one method of specifying dynamic behaviour of a model. It depicts activities which are carried out by human or computer actors, and the transitions between activities including the conditions governing the moving to another activity. The model may also include synchronization points at which two or more activities may meet or diverge indicating the possibility of parallel processing.

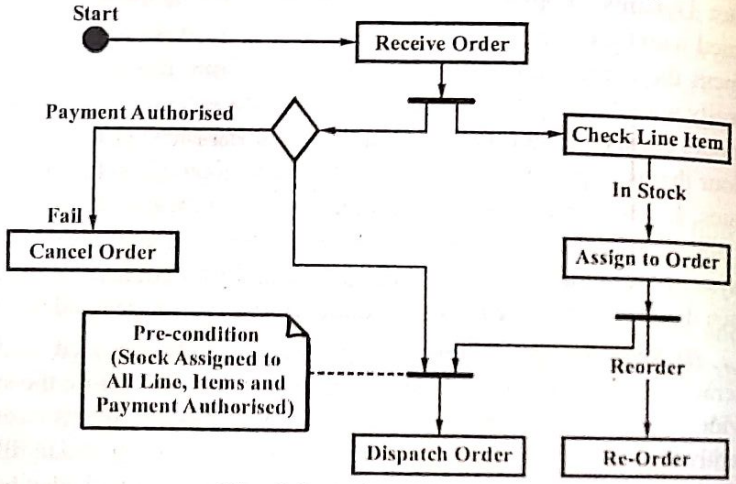


Fig. 2.3 Activity Diagram

Activity diagrams are used to show how different workflows in the system are constructed, how they start and the possibly many decision paths that can be taken from start to finish. They may also illustrate the where parallel processing may occur in the execution of some activities.

(iii) State Charts – State charts are used to detail the transitions or changes of state an object can go through in the system. They show how an object moves from one state to another and the rules that govern that change. State charts typically have a start and end condition.

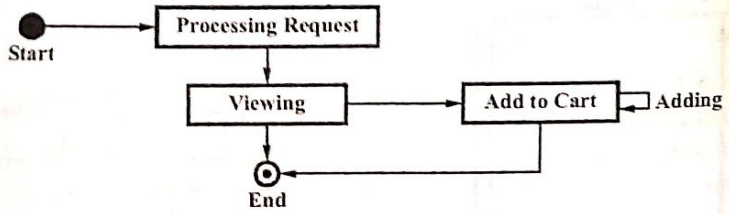


Fig. 2.4 State Diagram

(iv) Process Model – A process model is a UML extension of an activity diagram used to model a business process this diagram shows what goal the process has, the inputs, outputs, events and information that are involved in the process.

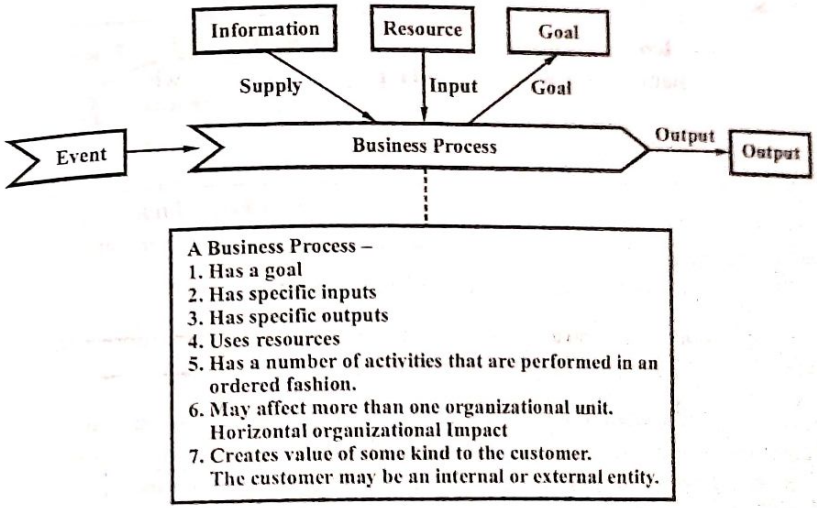


Fig. 2.5 Process Model

Q.7. Explain process model.

Ans. A software engineer or a team of engineers must incorporate a development strategy that include the process, methods, and tools layers and the generic phases to solve actual problems in an industry setting. This strategy is often called as a ***process model*** or a ***software engineering paradigm***.

That is, a software process model is an abstraction of a software process. A process model relies on the nature of the project and application, the methods and tools to be used, and the controls and deliverables that are required.

Therefore, it is essential to define process model for each software project. IEEE defines a process model as "a framework containing the processes, activities, and tasks involved in the development, operation and maintenance of a software product, spanning the life of the system from the definition of its requirements to the termination of its use".

The various process models are –

(i) Linear sequential model or waterfall model
(ii) Prototyping model
(iii) RAD model
(iv) Evolutionary process model
(v) Incremental model
(vi) Spiral model
(vii) Component-assembly model.

Q.8. Describe the 4+1 view model.

Ans. The 4+1 view model presented in was developed to rid the problem of software architecture representation. Five concurrent views are used, each view address a specific set of concerns of interest to the different stake-holders as shown in fig. 2.6.

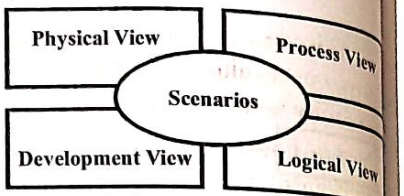


Fig. 2.6 Each View Address Specific Concerns

Software architecture = {Elements, Form, Rationale}

is applied independently. Each view is described using its own representation, in a so called blueprint. The design method is scenario-driven.

(i) Physical View – The elements of the physical view are easily identified in the logical, process and development views and are concerned with the mapping of these elements onto hardware, e.g. networks, processes, tasks and objects. In this view, quality requirements like availability, reliability (fault-tolerance), performance (throughput) and scalability can be addressed.

(ii) Process View – This view specifies the concurrency model used in the architecture. In this view, for example, performance, system availability, concurrency, distribution system integrity and fault-tolerance can be analyzed. The process view is described at several levels of abstractions, each addressing an individual concern.

In this view, the concept of a process is defined as a group of tasks that form an executable unit. Two kinds of tasks exist; major and minor. Major tasks are architectural elements, individually and uniquely addressable. Minor tasks, are locally introduced for implementation reasons, e.g. time-outs, buffering, etc. Processes represent the tactical level of architecture control. Processes can be replicated to deal with performance and availability requirements, etc.

For the process view use an expanded version of the Booch process view. Several styles are useful in the process view, e.g. pipes & filters client/server.

(iii) Development View – This view takes into account internal, or, intrinsic properties/requirements like reusability, ease of development, testability, and commonality. This view is the organization of the actual software modules in the software development environment. It is made up of program libraries or subsystems. The subsystems are organized in a hierarchy of layers. It is recommended to define 4-6 layers of subsystems in the development view. A subsystem may only depend on subsystems in the same or lower layers, to minimize dependencies.

This view supports allocation of requirements and work division among teams, cost evaluation, planning, progress monitoring, reasoning about reuse, portability and security.

The notation used is taken from the Booch method, i.e. modules/subsystems graphs. Module and subsystems diagrams that show import and export relations represent the architecture.

The development view is completely describable only after all the other views have been completed, i.e. all the software elements have been identified. However, rules for governing the development view can be stated early.

(iv) Logical View – This view denotes the partitions of the functional requirements onto the logical entities in the architecture. The logical view contains a set of key abstractions, taken mainly from the problem domain, expressed as objects and object classes.

If an object's internal behaviour must be defined, use state-transition diagrams or state charts.

The object-oriented style is recommended for the logical view. The notation used in the logical view is the Booch notation. However, the numerous adornments are not very useful at this level of design.

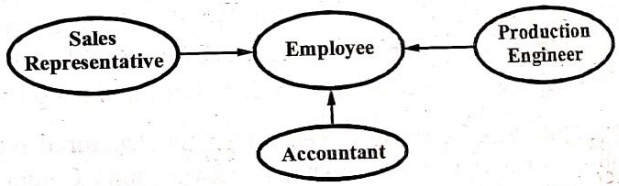


Fig. 2.7 Booch Notation Example in Logical View

(v) Scenarios – The fifth view (the +1) is the list of scenarios. Scenarios serve as abstractions of the most important requirements on the system. Scenarios play two critical roles, i.e. design driver, and validation/illustration. Scenarios are used to find key abstractions and conceptual entities for the different views, or to validate the architecture against the predicted usage.

The scenario view should be made up of a small subset of important scenarios. The scenarios should be selected based on criticality and risk.

Each scenario has an associated script, i.e. sequence of interactions between objects and between processes. Scripts are used for the validation of the other views and failure to define a script for a scenario discloses an insufficient architecture.

Scenarios are described using a notation similar to the logical view, with the modification of using connectors from the process view to show interactions and dependencies between elements.

ARCHITECTURES STYLES – DATA-FLOW ARCHITECTURE, PIPES AND FILTERS ARCHITECTURE, CALL AND RETURN ARCHITECTURE, DATA-CENTERED ARCHITECTURE, LAYERED ARCHITECTURE

Q.9. Discuss the architecture design process. ***(R.G.P.V., June 2004***

Ans. The architectural design process is considered as developing a basic structural framework for a system. It includes determining the major system components and their communications. The following are the three advantages of explicitly designing and documenting a software architecture –

(i) Stakeholder Communication – A high-level system presentation of system is the architecture. It may be used as a focus for discussion by a variety of different stakeholders.

(ii) System Analysis – To make the system architecture explicit at the initial stage of system development implies that some analysis may be performed.

(iii) Large-scale Reuse – The transfer of the architecture can be done across system having same requirements and hence can assist large-scale software reuse. There may be possibility of developing product-line architectures where the same architecture is used across several related systems.

The following activities are common to all architectural design processes –

(i) System Structuring – The system is structured into several principal sub-systems, which are independent software units. Communications between sub-systems are recognized.

(ii) Control Modeling – A general control relationships model is developed between the parts of the system.

(iii) Modular Decomposition – Each identified sub-system is decomposed into modules. The architect must decide on the module types and the types of their interconnections.

These activities are usually interleaved instead of conducting in sequence.

The result of the architectural design process is an architectural design document. It consists of several graphical representations of the system models along with related descriptive text. It should describe how the system is structured into subsystems and how each subsystem is structured into modules.

The different graphical models of the system present different perspectives on the architecture. The following architectural models may be developed –

(i) A static structural model that depicts the subsystems or components that are to be developed as separate units.

(ii) A dynamic process model that depicts how the system is organized into processes at run-time. This may be different from the static model.

(iii) An interface model that establishes the services provided by each subsystem through their public interface.

(iv) Relationship models that depict relationships like data flow between the subsystems.

Q.10. What do you mean by an architectural style ?

Ans. Architectural styles define a family of systems in terms of a pattern of structural organization. They also characterize a family of systems that are related by sharing structural and semantic properties. In essence, the purpose of using architectural styles is to develop a structure for all the components present in a system. If an existing architecture is to be reengineered, then imposition of an architectural style results in fundamental changes in the structure of the system. Also, this change includes reassignment of the functionality performed by the components.

Q.11. How do you assess an architectural style that has been derived ? ***(R.G.P.V., June 2017)***

Ans. The assessment of an architectural style that has been derived in the following ways –

Data – How data communication between components take place ? Is data flow continuous or discrete ? What is data transfer mode (i.e. either one-to-one or globally available) ? What is the role of data components, if exist ? How data and function components interact with each other ? Does the data component interact to other components actively or passively ?

Control – Within architecture, how control management take place ? Within control hierarchy (if exist), what is the role of components ? Within the system, how the control transfer take place between components ? How control sharing is done among components ? What is the topology that control defines ? Is there a synchronization of control ? Within a system, how interaction of control and data takes place ?

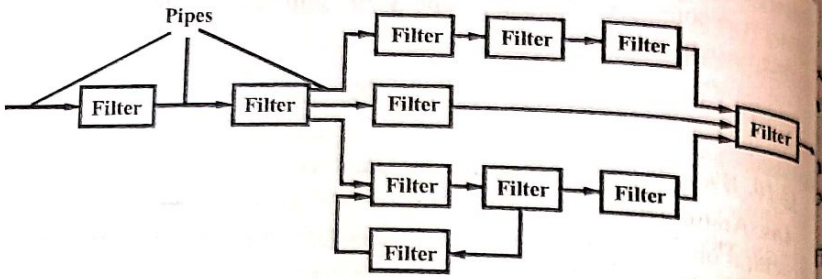
These questions gives the early assessment of architectural style.

Q.12. What do you mean by data-flow architecture ?

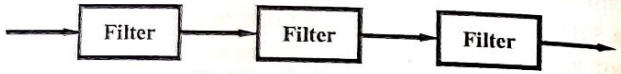
Ans. This architecture is used in case of the transformation of input data into output data through a series of computational or manipulative components. Fig. 2.8 shows these architectures.

In fig. 2.8 (a), a pipe and filter pattern has a set of components called ***filter***. These filters are connected by pipes that transmit data from one component to the next. Each filter is designed to accept data input of a certain form and produces

data output of a specified form, which goes to the next filter as its input. No
neighbouring filters require to know the working of each other.



(a) Pipes and Filters



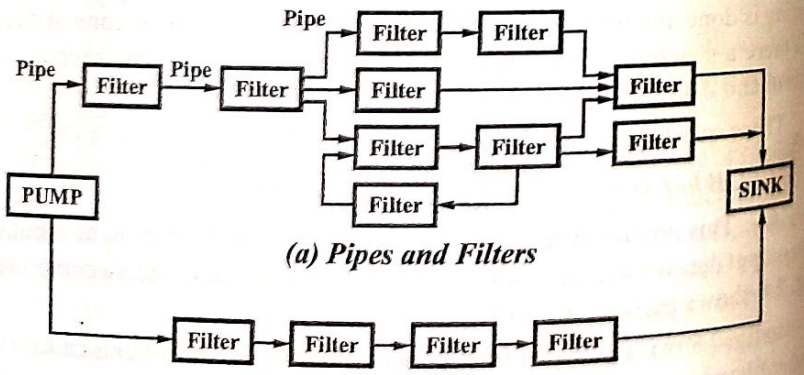
(b) Batch Sequential

Fig. 2.8 Data-flow Architectures

Fig. 2.8 (b) illustrates a batch sequential architecture, where the data flo
degenerates into a single line of transforms. This pattern takes a batch of da
and then a series of sequential components i.e., filters are applied to transform i

Q.13. Explain in detail about the pipes and filters with example.

Ans. In a pipe and filter style each component has a set of inputs and a s
of outputs. A component reads stream of data on its inputs and produces strea
of data on its outputs, delivering a complete instance of result in standar
order. This is accomplished by applying a local transformation to the inp
streams and computing incrementally so output begins before input i
consumed. Hence components are termed "filters". The connectors of th
style serve as conduits for the streams, transmitting outputs of one filter t
inputs of another. Pipes and filter style is shown in fig. 2.9.



(a) Pipes and Filters

(b) Batch Sequential

Fig. 2.9 Pipes and Filters

Pipes and filters style include the following –

(i) The filter transforms or filters the data it receives via the pipes
ith which it is connected. A filter can have any number of input pipes and
ny number of output pipes.

(ii) The pipe is the connector that passes data from one filter to the
ext. It is a directional stream of data, which is usually implemented by a data
uffer to store all data, until the next filter has time to process it.

(iii) The pump or producer is the data source. It can be a static text
ile, or a keyboard input device, continuously creating new data.

(iv) The sink or consumer is the data target. It can be another file, a
database, or a computer screen.

Some examples of pipe filter architecture are as follows –

Unix Programs – The output of one program can be linked to the input
of another program.

Compilers – The consecutive filters perform lexical analysis, parsing,
semantic analysis, and code generation.

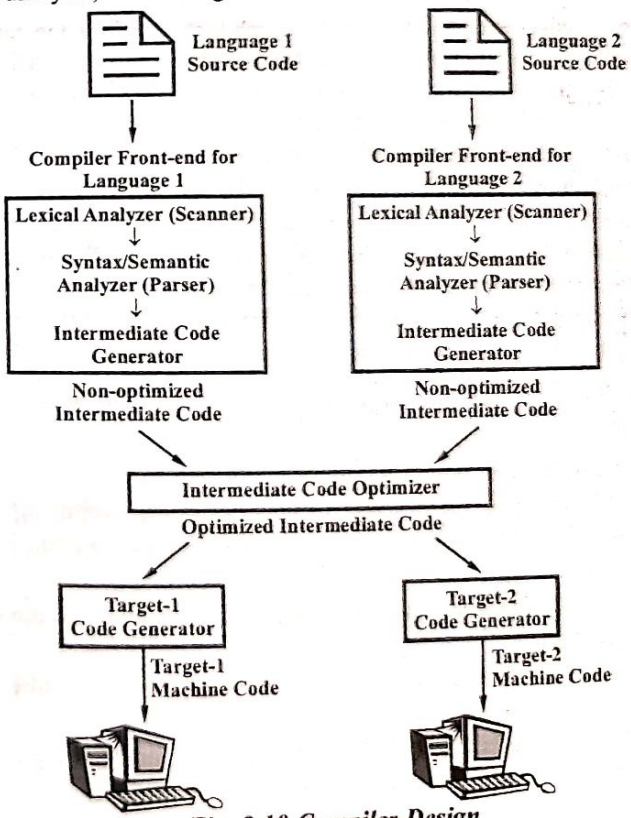


Fig. 2.10 Compiler Design

Compilers translate source programs in high-level languages to machine code of the underlying hardware. A compiler consists of three parts – the frontend, the intermediate code optimizer, and the backend.

The ***front end*** checks whether the program is correctly written in terms the programming language syntax and semantics. Here legal and illegal programs are recognized. Errors are reported, if any, in a useful way. Type checking is also performed by collecting type information. The frontend then generates an intermediate representation or IR of the source code for processing by the middle-end.

The ***intermediate code optimizer*** is where optimization takes place. Typical transformations for optimization are removal of useless or unreachable code, discovery and propagation of constant values, relocation of computation to a less frequently executed place (e.g., out of a loop), or specialization of computation based on the context. The middle-end generates another IR for the following backend. Most optimization efforts are focused on this part.

The ***back end*** is responsible for translating the IR from the middle-end into assembly code. The target instruction(s) are chosen for each IR instruction. Register allocation assigns processor registers for the program variables where possible. The backend utilizes the hardware by figuring out how to keep parallel execution units busy, filling delay slots, and so on. Although most algorithms for optimization are in NP, heuristic techniques are well-developed.

Some problems encountered in pipe and filter architecture includes that a filter needs to wait until it has received all data (e.g. a sort filter), its buffer may overflow, or it may deadlock. Also, if the pipes only allow a single data type (a character or byte) the filters will need to do some parsing. This complicates things and slows them down. If you create different pipes for different data types, you cannot link any pipe to any filter.

Q.14. What are the advantages and disadvantages of pipes and filters architectures.

Ans. Advantages of pipes and filters are as follows –

(i) They allow the designer to understand the overall input/output behaviour of a system as a simple composition of the behaviour of the individual filters.

(ii) They support reuse – Any two filters can be hooked together if they agree on data.

(iii) Systems are easy to maintain and enhance – New filters can be added to exciting systems.

(iv) They permit certain kinds of specialized analysis e.g. deadlock, throughput.

(v) They support concurrent execution.

Disadvantages of pipes and filters are as follows –

(i) They lead to a batch organization of processing.

(ii) Filters are independent even though they process data incrementally.

(iii) Not good at handling interactive applications.

(iv) When incremental display updates are required.

(v) They may be hampered by having to maintain correspondences between two separate but related streams.

(vi) Lowest common denominator on data transmission.

This can lead to both loss of performance and to increased complexity in writing the filters.

Q.15. Discuss the different classification of architectural styles with respect to software and discuss each style in detail. (R.G.P.V., June 2012)

Ans. The various architectural styles are as follows –

(i) Call and Return – This architectural style helps a software designer to get a program structure that is easier to modify and scale. The following are the substyles of this category –

(a) Main Program/Subprogram Architecture – It decomposes function into a control hierarchy where a main program calls various program components, which in turn may call still other components. This type of architecture is shown in fig. 2.11.

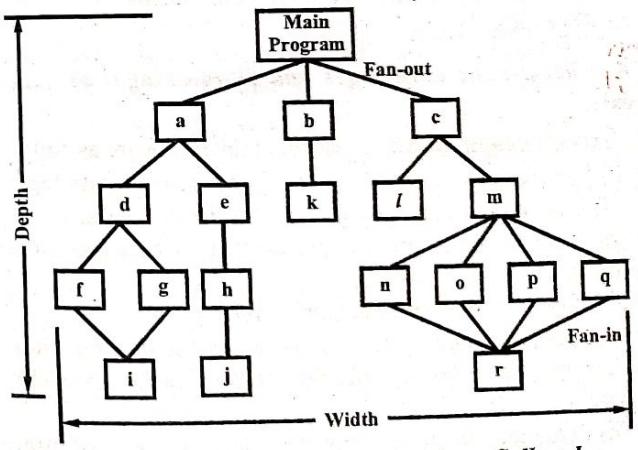


Fig. 2.11 Structure Terminology for a Call and Return Architectural Style

(b) Remote Procedure Call Architecture – In this, components of a main program or subprogram architecture are distributed over a network across several computers.

(ii) ***Data-centered*** – In this architectural style, a data store is in the center and is accessed frequently by other components which add, delete, update, or modify data within the store. Fig. 2.12 shows a typical data-centered style.

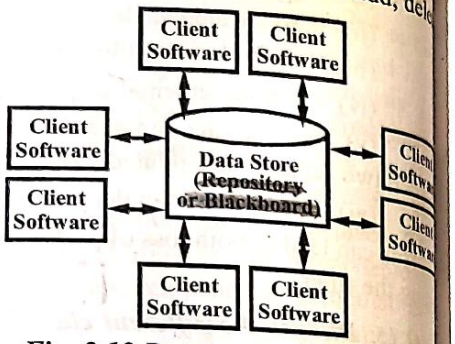


Fig. 2.12 Data-centered Architecture

Observe the fig. 2.12, where client software accesses a central repository (i.e., data store). The data store can be passive in some cases i.e., client software accesses the data independent of any change or the actions of other client software.

Data-centered architectures enhance integrability i.e., currently components can be changed and new client components can be added architecture, regardless of other clients. Also, the blackboard mechanism is to pass data among clients. The processes are executed independently by components.

(iii) ***Data-flow*** – Refer to Q.12.

(iv) ***Object-oriented*** – The system components encapsulate data the required operations for data manipulation. Communication and coordinati is established via message passing between components.

(v) ***Layered*** – Refer to Q.17.

Q.16. What are the advantages and disadvantages of data cente architecture.

Ans. The advantages of data centered architecture are as follows –

(i) It centralizes the data logic or Web service access logic.

(ii) It provides a substitution point for the unit tests.

(iii) It provides a flexible architecture that can be adapted as the over design of the application evolves.

Disadvantages of data centered architecture –

(i) The associated systems must agree on the repository mod inevitably compromising on the specific needs of each, adversely affecti performance.

(ii) Evolution may be difficult when large volumes of information generated according to an agreed model, and translating this to a newer mod may be very expensive, difficult or impossible.

(iii) Activities such as backup, security, access control and recove may be different for different associated systems, resulting in additional a unnecessary overheads.

Q.17. Explain in detail about the layered architecture.

Ans. Fig. 2.13 shows the basic structure of a layered architecture.

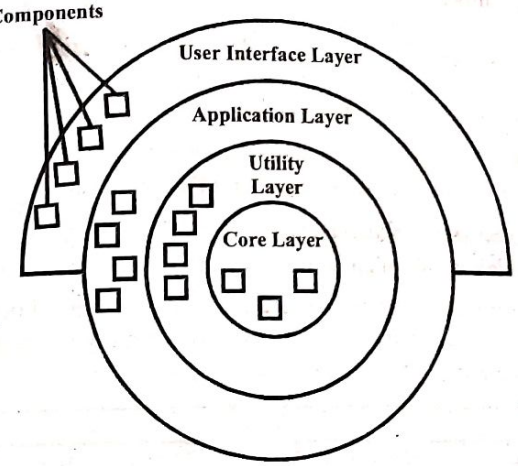


Fig. 2.13 Layered Architecture

This architecture consists of a number of different layers. Each layer performs operations that progressively become closer to the machine instruction set. At the outermost layer, called ***user interface layer***, compo-nents service user interface operations. At the innermost layer, known as ***core layer***, components perform operating system interfacing. The two intermediate layers i.e., ***application layer*** and ***utility layer***, offer application software functions and utility services, respectively.

Layered architecture is an architectural style that organizes the software hierarchically, each layer in the hierarchy providing service to the layer above it and serving as a client to the layer below it. A layer can be loosely defined as a set of (sub) systems with the same degree of generality.

The layers architectural pattern helps to structure applications that can be decomposed into groups of subtasks in which each group of subtask is at a particular level of abstraction.

The layered architectural style has been described as an inverted pyramid of reuse where each layer aggregates the responsibilities and abstractions of the layer directly beneath it.

The common principles for designs that use the layered architectural style include –

(i) Abstraction – This style provides the functionality, while abstracting the roles and responsibilities of individual layers and their interrelation.

(ii) Encapsulation – The layer boundaries are not exposed to the features such as data types, methods, implementation details, achieving the required encapsulation.

(iii) Specify the Services – Each layer has to specify a distinct functionality. The behaviour and flow of data within the layer boundaries are also to be specified in clear terms.

(iv) Reusable – There exists no dependencies between the lower layers and the upper ones; we can reuse them in other scenarios.

(v) Coupling between Layers – Provide coupling, abstractly, between layers to establish communication among them.

Fig. 2.14 shows the layered architecture of the android operating system. It is a software stack of different layers where each layer is a group of several different components.

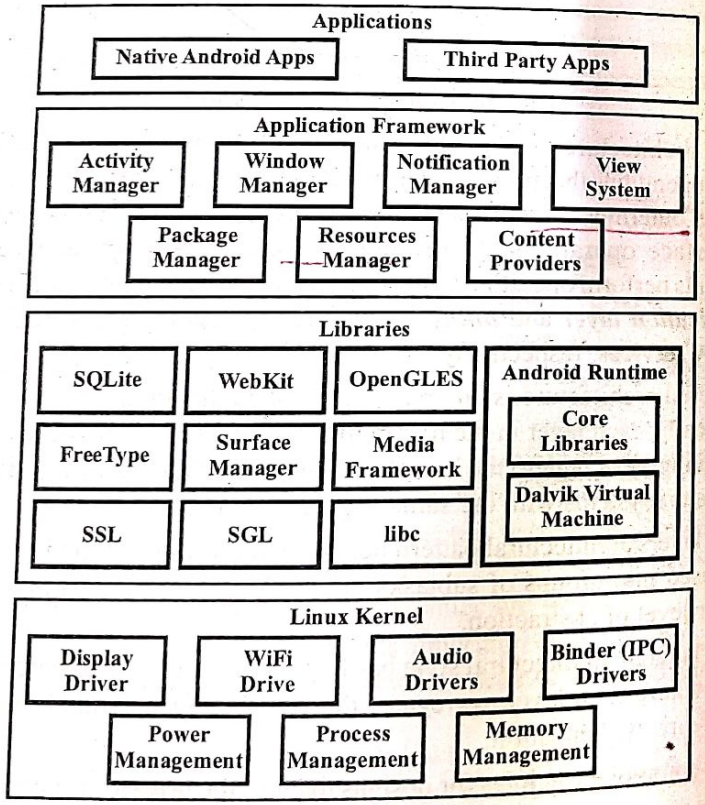


Fig. 2.14 Android Layered Architecture

The basic layer is the Linux kernel on which the whole of android OS is built on (version Linux 2.6 kernel). This acts as an abstraction between the hardware and the software layers.

The next layer is the native libraries which enables the device to handle different types of data. These are written in C or C++ and are specific for particular hardware.

Android runtime layer consists of the dalvik virtual machine and the core java libraries.

The dalvik virtual machine is a type of Java virtual machine used in android devices to run the applications and is optimized for low processing power and low memory. The DVM allows multiple instances of virtual machine to be created simultaneously providing, security and memory management, isolation and threading support.

Application framework is the layer above the android runtime and this the block where the applications directly interacts with and this block includes activity manager, window manager notification manager, package manager, resource manager and content providers.

The topmost layer of the android architecture is the application layer that supports writing applications in two modes; native android apps and in the third party apps thereby providing an endless opportunity to the developers.

Q.18. What are the advantages and disadvantages of layered architecture.

Ans. Advantages of layered architecture are as follows –

(i) They support designs based on increasing levels abstraction.

(ii) Allows implementers to partition a complex problem into a sequence of incremental steps.

(iii) They support enhancement.

(iv) They support reuse.

Disadvantages of layered architecture are as follows –

(i) Not easily all systems can be structures in a layered fashion.

(ii) Performance may require closer coupling between logically high-level functions and their lower-level implementations.

(iii) Difficulty to mapping existing protocols into the ISO framework as many of those protocols bridge several layers.

(iv) Layer bridging – functions is one layer may talk to other than its immediate neighbour.

Q.19. Explain in detail about the client server architecture.

Ans. The client server model is a computing model that acts as a distributed application which partitions tasks or workloads between the providers of a resource or service, called servers and service requesters called clients. The

client/server characteristic describes the relationship of cooperating programs in an application. The server component provides a function or service to one or many clients, which initiate requests for such services.

There are two types of client server architectures –

(i) 2-tier Architectures – In this architecture, client directly interacts with the server. This type of architecture may have some security holes and performance problems. Internet explorer and the web server works on two tier architecture. Here security problems are resolved using secure socket layer (SSL).

(ii) 3-tier Architectures – In this architecture, one more software sits in between client and server. This middle software is called middleware. Middleware are used to perform all the security checks and load balancing in case of heavy load. A middleware takes all requests from the client and after doing required authentication it passes that request to the server. Then server does required processing and sends response back to the middleware and finally middleware passes this response back to the client. If you want to implement a 3-tier architecture then you can keep any middle ware like Web logic or WebSphere software in between your Web server and Web browsers.

Some disadvantages of client server architecture are –

Dependence – The client-server network model relies on a functioning and available centralized server. If the centralized server is removed from the system or goes down due to problems, the entire network cannot function.

Expense – The central server computer must be powerful enough to maintain and share resources with the other computers on the network. This entails a substantial cost.

Congestion – Centralized servers must handle the majority of the network traffic, as all queries for resources are directed toward the server. This can cause network congestion on the network and slow down response times for each computer available.

Maintenance – Client-server networks often require a staff with at least a single network administrator to manage and maintain the equipment and the network. Other network operating systems, such as peer-to-peer network systems, do not require a network administrator to maintain machines, as this work is distributed among individual clients and their related machines.

Q.20. Give example of client server architecture.

Ans. TCP-IP is perfect example of client server architecture. The TCP-IP client server is shown in fig. 2.15.

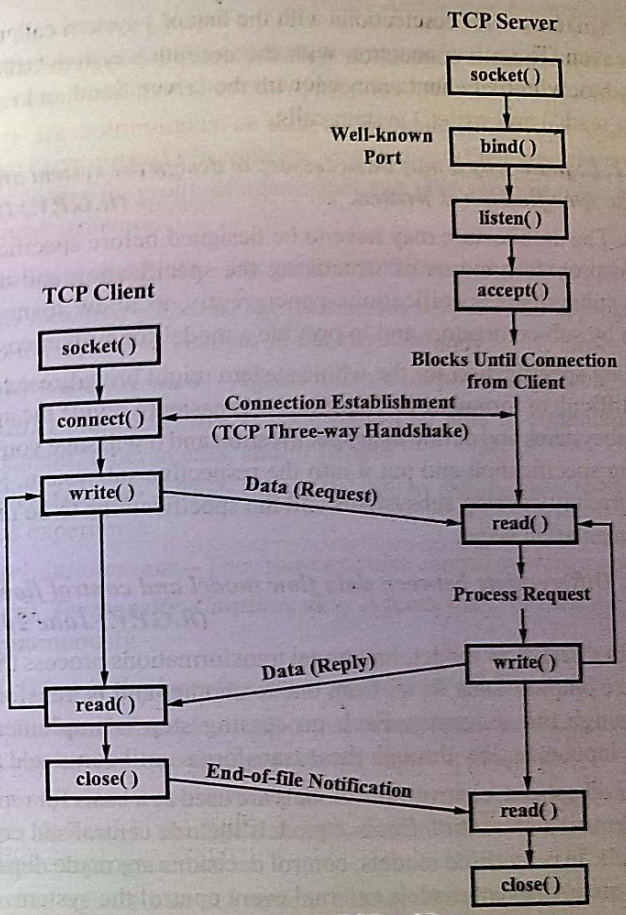


Fig. 2.15 TCP – IP

The steps involved in establishing a socket on the client side are as follows–

(i) Create a socket with the socket( ) system call.

(ii) Connect the socket to the address of the server using the connect( ) system call.

(iii) Send and receive data. There are a number of ways to do this, but the simplest is to use the read( ) and write( ) system calls.

The steps involved in establishing a socket on the server side are as follows –

(i) Create a socket with the socket( ) system call.

(ii) Bind the socket to an address using the bind( ) system call. For a server socket on the Internet, an address consists of a port number on the host machine.

(iii) Listen for connections with the listen( ) system call.

(iv) Accept a connection with the accept( ) system call. This call typically blocks until a client connects with the server. Send and receive data using the read( ) and write( ) system calls.

Q.21. Explain why it may be necessary to design the system architecture before the specification is written. ***(R.G.P.V., Dec. 2015)***

Ans. The architecture may have to be designed before specifications are written to provide a means of structuring the specification and developing different subsystems specifications concurrently, to allow manufacture of hardware by sub-contractors and to provide a model for system costing.

Writing specification for the whole system might bring great complexity and it is difficult to formulate it. Therefore, it is easier to divide the system into simpler subsystems and define their specification and it will save you the hassle of defining specification and put it into the respective subsystem. Hence, we can concurrently develop subsystems and the specifications to be readily into the implementation stage.

Q.22. Differentiate between data flow model and control flow model. ***(R.G.P.V., June 2008, 2013)***

Ans. In a data-flow model, functional transformations process their inputs and produce outputs. Data flows from one to another and is transformed as it moves through the sequence. Each processing step is implemented as a transform. Input data flow through these transforms until converted to output.

On the other hand, control-flow models are used as a basis for representing the transformation of control. Control models include centralised control and event models. In centralised models, control decisions are made depending on the system state, in event models external event control the system.

AGENT BASED ARCHITECTURE, MICRO SERVICES ARCHITECTURE, REACTIVE ARCHITECTURE, REPRESENTATIONAL STATE TRANSFER ARCHITECTURE ETC.

Q.23. Write short note on agent based system.

Ans. Agent systems especially the multi-agent systems are part of the wide research area of distributed artificial intelligence (DAI). DAI can be subdivided into distributed problem solving (DPS) and multi-agent systems (MAS). DPS deals mainly with information management issues such as task decomposition and solution synthesis. While, MAS deals with breaking-down the problem and assigning the sub problems to the problem solving agents with the best abilities to solve the particular sub-problem. Although each agent has its own goal and interests, the assigning of the sub-problems is done in such away as to solve the global problem in the most appropriate and efficient manner.

Agents are autonomous or semi-autonomous hardware or software systems that carry out tasks in complex, continuously changing environments. A MAS consists of a group of agents that can take specific roles within an organizational structure.

Capabilities of Agents – The basic capabilities of agents are as follows–

(i) Reactive – That is, agents must react timely and appropriately to unplanned events and to changes in the environment.

(ii) Goal Oriented – Agents act in a purposeful manner.

(iii) Communicative – They should be able to communicate with the environment, other agents, and/or people.

(iv) Adaptive – They should be able to change their behaviour due to previous experience.

(v) Autonomous – They must exercise control over their own actions.

(vi) Temporally Continuous – Agents must posses the ability to running continuously.

Q.24. What is the learning agent architecture ? Explain.

Ans. The idea behind learning is that perceptions should be used not only to act but also to improve the agent ability to act in the future.

Basic software agents have no learning; they act according to the perceptions defined in the agent design. Therefore, for new perceptions the agent must be reprogrammed.

Learning agents can at runtime change their behaviour according to changes in the environment. In this type of agents, perceptions should be used not only to act but also to improve the agent ability to act in the future.

A learning agent has four basic components (see fig. 2.16) performance, critic, learning and problem generator.

The performance component is what we have previously considered to be a basic agent – perceives and acts on the environment.

The learning component is responsible for making the agent behaviour improvements. It uses feedback from the critic on how the agent is doing and determines how the performance component should be modified to do better in the future. The critic tells the learning component about the success of the agent according to a fixed performance standard. The critic is necessary because the percepts themselves provide no indication of the agent success.

The last component of the learning agent is the problem generator which is responsible for suggesting actions that will lead to new and informative

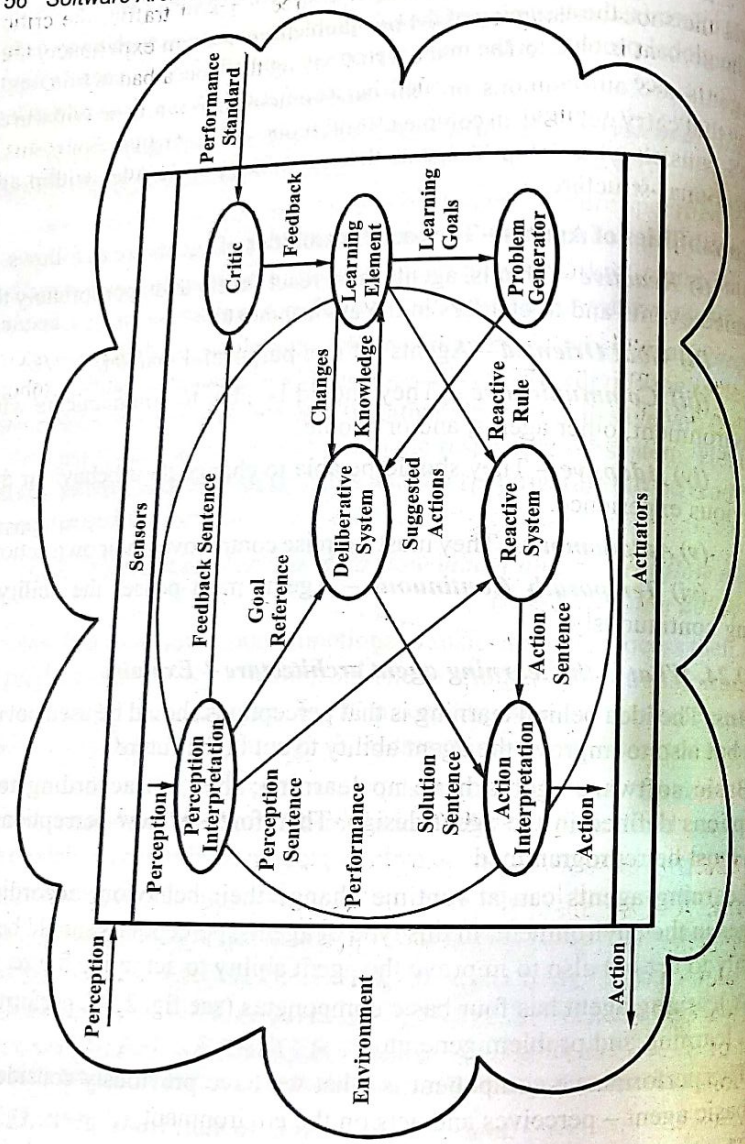


Fig. 2.16 The Structure of a Learning Agent

experiences. The point is that if the performance element had its way, it would keep doing the actions that are best, given what it knows. The problem generator main goal is to suggest these exploratory actions. This is what scientists do when they carry out experiments.

An example of the functioning of a learning agent architecture is automated taxi. The performance element consists of whatever collection of knowledge and procedures the taxi has for selecting its driving actions. The critic observes the world and passes information along to the learning element. For example,

after the taxi makes a quick left turn across three lanes of traffic, the critic observes the shocking language used by others drivers. From experience, the learning element is able to formulate a rule saying this was a bad action, and the performance element is modified by installation of the new rule. The problem generator might identify certain areas of behaviour in need of improvement and suggest experiments, such as trying out the brakes on different road surfaces under different conditions.

Q.25. Briefly explain the single-agent systems.

Ans. Single-agent systems are based on the centralized process model. In these systems, there is a single agent which makes all the decisions, leaving all the other agents to act as remote slaves. Therefore, single agent systems may have a number of entities such as transducers, actuators and/or robots. However, all entities send their perceptions to, and receive their actions from the same central processor.

The environment of a single-agent system may have other agents. However, these agents act as actuators or sensors because they do not posses goals of their own. The single-agent system is shown in fig. 2.17.

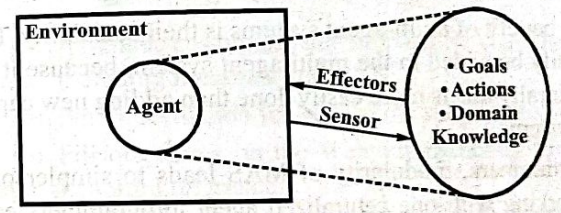


Fig. 2.17 General Single-agent Framework

Q.26. Describe the multi-agent systems.

Ans. A multi-agent system is a loosely coupled network of problem-solving agents that work together to solve problems that none of them could solve alone. The main difference between multi-agent systems and single-agent systems is that in multi-agent systems several agents exist, and they are aware of each other's goals and actions. Besides being aware of each other's intentions and behaviour, in a fully general multi-agent system, agents also communicate with one another, either to help an individual agent achieve its goal, or in a rare case, prevent it.

Multi-agent systems are composed of several autonomous entities which have the following general characteristics –

(i) Each agent has incomplete capabilities to solve the problem.

(ii) There is no global control.

(iii) Data is decentralized.

(iv) Computation is asynchronous.

Fig. 2.18 shows a multi-agent system with multiple agents, some w communication capabilities and others without communication capabilities.

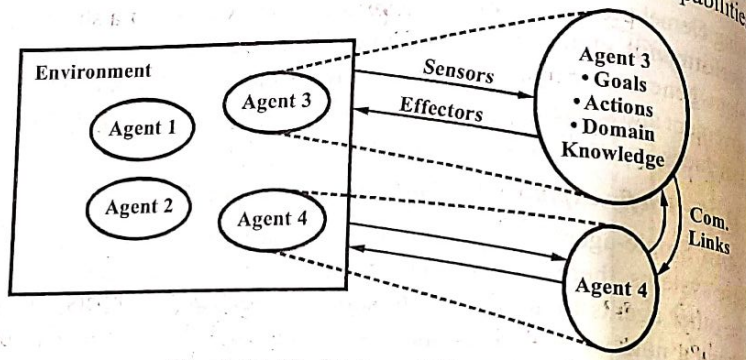


Fig. 2.18 Multi-Agent Framework

Q.27. What are the advantages of multi-agent system (MAS) ?

Ans. Multi-agent system (MAS) can be used for both distributed an centralized systems. For example, multiple agents can be used to speed u systems by providing means for parallel programming.

Another benefit of multi-agent systems is their scalability. That is, a ne agent can easily be added to the multi-agent system, because it is inherentl modular. Generally this is more easily done than adding new capabilities in monolithic systems.

For programmers, modularity of MAS leads to simpler programming Instead of working with one centralized agent, programmers easily identif subtasks and assign control of these subtasks to different agent. This also solve the problem of sharing time of one centralized agent between separate tasks.

Q.28. What is an intelligent software agent ? What are the types o intelligent software agents ? ***(R.G.P.V., June 2012)***

Or

Explain software agents. ***(R.G.P.V., May 2010)***

Ans. In computer science, a ***software agent*** is a piece of software tha acts for a user or other program in a relationship of ***agency.*** Such action o behalf of implies the authority to decide which (and if) action is appropriate. The idea is that agents are not strictly invoked for a task, but activate themselves.

Haag suggests that there are only four essential types of intelligent software agents –

(i) Buyer Agents (Shopping Bots) – Buyer agents travel around a network (i.e. the internet) retrieving information about goods and services. These agents, also known as 'shopping bots', work very efficiently for

commodity products such as CDs, books, electronic components and other one-size-fits-all products. ***Amazon.com*** is a good example of a shopping bot. The website will offer you a list of books that you might like to buy on the basis of what you're buying now and what you have bought in the past.

Another example is used on ***eBay.*** At the bottom of the page there is a list of similar products that other customers who did the same search looked at. This is because it is assumed the user tastes are relatively similar and they will be interested in the same products. This technology is known as ***collaborative filtering.***

(ii) User Agents (Personal Agents) – User agents, or personal agents, are intelligent agents that take action on your behalf. In this category belong those intelligent agents that already perform, or will shortly perform, the following tasks –

(a) Check your e-mail, sort it according to the user's order of preference, and alert you when important emails arrive.

(b) Play computer games as your opponent or patrol game areas for you.

(c) Assemble customized news reports for you. There are several versions of these, including ***newshub*** and CNN.

(d) Find information for you on the subject of your choice.

(e) Fill out forms on the Web automatically for you, storing your information for future reference.

(f) Scan Web pages looking for and highlighting text that constitutes the important part of the information there.

(g) Discuss topics with you ranging from your deepest fears to sports.

(h) Facilitate with online job search duties by scanning known job boards and sending the resume to opportunities who meet the desired criteria.

(i) Profile synchronization across heterogeneous social networks.

(iii) Monitoring and Surveillance (Predictive) Agents – Monitoring and surveillance agents are used to observe and report on equipment, usually computer systems. The agents may keep track of company inventory levels, observe competitors' prices and relay them back to the company, watch stock manipulation by ***insider trading*** and rumors, etc.

For example, NASA's Jet Propulsion Laboratory has an agent that monitors inventory, planning, and scheduling equipment ordering to keep costs down, as well as food storage facilities. These agents usually monitor complex computer networks that can keep track of the configuration of each computer connected to the network.

(iv) Data Mining Agents – This agent uses information technolo to find trends and patterns in an abundance of information from many differe sources. The user can sort through this information in order to find whate information they are seeking.

A data mining agent operates in a data warehouse discovering informatio A data warehouse brings together information from lots of different source Data mining is the process of looking through the data warehouse to fi information that you can use to take action, such as ways to increase sales keep customers who are considering defecting.

Q.29. What do you understand by microservices architecture ?

Ans. A microservices architecture is a method of developing applicatio as a network of independently deployable, modular services in which each servi runs a unique process and communicates through a well-defined, lightweig mechanism to serve a business goal. Think of it like a honeycomb. Each cell a honeycomb is independent from all the others and may be used for a differe purpose. Each single cell is not very useful, but when combined with each othe a strong and flexible network is created that supports many uses.

The vision of a service oriented architecture (SOA) was first sketched o in the 1980s as a way to unify monolithic islands of automation into a functiona goal. It's no coincidence that the concept of SOA arrived at the same time the internet made large-scale peer-to-peer networking possible.

Applications based upon services need other services to manage them The concept of middleware is based upon this idea. Middleware coordinate groups of services to ensure that data flows smoothly between them and th services are available when needed. The enterprise service bus (ESB) is anothe way to coordinate services. This is an integration architecture that uses common communications layer (the bus) to orchestrate a variety of point-to- point connections between providers and consumers. For example, an ESB may call upon separate services for a shopping cart, a credit approval proces and a customer account to present a unified checkout window to an onlin shopper. In most cases, a common data storage layer is shared by all services

The difference between a microservices approach and an ESB is th microservices architectures have no single coordinating layer. Each servic communicates independently with the others. This enables applications to b assembled quickly and flexibly by teams working in separate functions. The can each be written in different languages, which gives developers flexibilit to match the language to the task. Each service can be developed using th programming language most appropriate to the task. The goal in microservice development is to deconstruct the application into the smallest possible part so that services can be shared and combined easily with other applications.

Q.30. What are the general characteristics of microservices architecture?

Ans. General characteristics of microservices architecture are as follows–

(i) Applications are developed as a suite of small services, each running as an independent process in its own logical machine (or Linux container)

(ii) Services are built around capabilities – single responsibility principle.

(iii) Each microservice has a separate codebase and is owned by a separate team.

(iv) One can independently replace/upgrade/scale/deploy services.

(v) Standard lightweight communication is used, often REST calls over HTTP.

(vi) Potentially heterogeneous environments are supported.

Q.31. List out the merits and demerits of microservice architecture.

Ans. There are following merits and demerits of the microservice architecture –

Merits – Merits of microservice architecture are as follows –

(i) Faster and simpler deployment and rollback with smaller services. Taking advantage of the divide and conquer paradigm in software delivery and maintenance.

(ii) Ability to horizontally scale out individual services. Not sharing the same deployment platform with other services allows each service to be scaled out as needed.

(iii) Selecting the right tool, language and technology per service, without having to conform to a homogeneous environment being dictated by shared infrastructure.

(iv) Potential for fault isolation at microservice level by shielding services from common infrastructure failure due to the fault of one service. Where a system is designed to withstand the failure of some microservices, the result is higher availability for the system.

(v) Goes hand in hand with continuous delivery and integration.

(vi) Promotes DevOps culture with higher service self-containment and less common infrastructure maintenance.

(vii) More autonomous teams lead to faster/better development.

(viii) Facilitates A/B testing and canary deployment of services.

(ix) Traditional divide and conquer benefits.

Demerits – The downsides of MSA are direct results of higher service distribution. There is also a higher cost to having less common infrastructure. Demerits may be enumerated as follows –

(i) Network reliability is always a concern.

(ii) Less tooling/IDE support given the distributed nature.
(iii) Tracing, monitoring and addressing cascading failures are complex.
(iv) QA particularly integration testing can be difficult.
(v) Debugging is always more difficult for distributed systems.
(vi) Higher complexity – higher fixed cost and overhead.
(vii) Heterogenous environments are difficult and costly to maintain.

Q.32. Give some examples of microservices.

Ans. There are several robust frameworks that developers can use to build microservices-based applications. Frameworks incorporate libraries of essential services that developers need to build microservices-based applications. Here are some examples –

(i) Spring Boot is a highly regarded framework for dependency injection which is a technique for building highly decoupled systems. It's known for simplicity, flexibility and support for distributed programming techniques like inversion of control and aspect-oriented programming. It also permits developers to choose between multiple web servers like Tomcat, Jetty and Undertow.

(ii) Jersey is a RESTful Web Services framework for development of RESTful webservices in Java. RESTful refers to a popular development technique in which messages between microservices are handled with the web-standard HTTP protocol. Known for its ease-of-use, Jersey provides support for JAX-RS, a Java application program interface (API) specification for the development of Web services.

(iii) Swagger is a framework of development using APIs. It enables consistent descriptions of APIs that machines can read and that can serve as documentation. Swagger also automatically generate client libraries for API in a wide variety of languages.

Q.33. Discuss various technologies and development techniques used in microservices.

Ans. The microservices approach to application development has been enabled by a number of new technology and development techniques –

(i) High-speed, low-latency networks enable sophisticated distributed applications to be constructed of services from many providers. For example, an application can call a secure document signing service or fraud detection service in milliseconds in order to close a sale.

(ii) Containers are lightweight virtual machines that contain only the infrastructure elements necessary to perform a service. They can be launched and shut down quickly with minimal management overhead. Each service can be encapsulated in its own container and stored in a library.

(iii) RESTful APIs are defined by whatis.com as application program interfaces (API) that use HTTP requests to GET, PUT, POST and DELETE data. They're a low-bandwidth way for services to communicate with each other using a standard set of commands. This makes them well-suited to loosely coupled applications on the Internet. The population of services exposed as APIs is exploding. ProgrammableWeb lists more than 17,000 APIs, up from just 300 a decade ago. Not all microservices are RESTful, but all are message-driven.

(iv) Distributed databases have multiple services handling data requests. Each service works on a subset of the data and coordinates results with other services via an orchestration platform. This allows for highly scalable applications to be built at low cost using commodity servers.

(v) DevOps is an agile programming technique that emphasizes modularity, frequent releases and a constant feedback cycle.

(vi) DataOps combines the concepts provided by DevOps and also layers in the management and availability of data models. This enables the ability to quickly productionize intelligent machine learning models without the old approach of throwing the model over the wall with fingers crossed that someone else will figure out how to put it into production.

Q.34. Explain in detail the reactive architecture.

Ans. Reactive agent architecture is based on the direct mapping of situation to action. It is different from the logic based architecture where no central symbolic world model and complex symbolic reasoning are used. Agent responses to changes in the environment in a stimulus-response based. The reactive architecture is realized through a set of sensors and effectors, where perceptual input is mapped to the effectors to changes in the environment. Brook's subsumption architecture is known as the best pure reactive architecture. This architecture was developed by Brook who has critiqued on many of the drawbacks in logic based architecture. Fig. 2.19 illustrates an example of reactive architecture. The fig. 2.19 shows that each of the percept situation is mapped into an action which specifically responses to the percept situation.

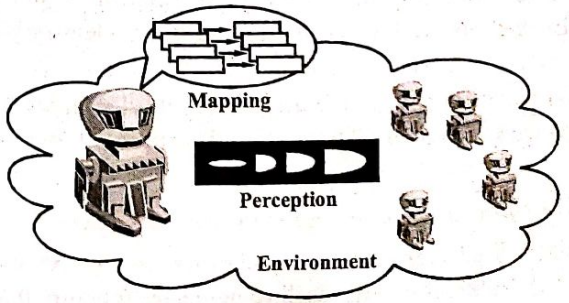


Fig. 2.19 Reactive Architecture

The key idea of subsumption architecture is that intelligent behaviour can be generated without explicit representations and abstract reasoning with symbolic AI technique. Intelligence is an emergent property of certain complex systems. Subsumption architecture is implemented in finite state machines with different layers connected to sensors that perceive the environment changes and map the action to be performed. A set of task-accomplishing behaviour are used in the decision making process. Each of the behaviour can be thought of as an individual function which maps changes in the environment with an action. Multiple behaviours that can be fired simultaneously is another characteristic of subsumption architecture. The subsumption architecture hierarchical structure represents different behaviours. The lowest layer in the hierarchy has the highest priority. Higher layer represent more abstract behaviour than the lower layer in the hierarchy. Complex behaviour is achieved through the combination of these behaviours. Fig. 2.20 shows action selection in the layered architecture. In this layered architecture, the lower the layer the higher the priority. The lower layer will be the primitive behaviour and higher layer will represent a more abstract behaviour.

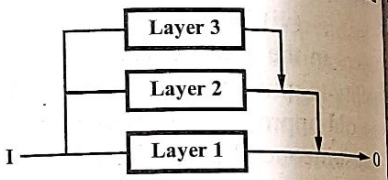


Fig. 2.20 Action Selections in Layered Architecture

Q.35. What are the advantages and disadvantages of reactive architecture?

Ans. The advantages of reactive architecture is that it is less complicated to design and implement than logic-based architecture. An agent's behaviour is computationally tractable. The robustness of reactive architecture against failure is another advantage. Complex behaviours can be achieved from the interaction of simple ones.

The disadvantages of reactive architecture include –

(i) Insufficient information about agent's current state to determine an activation action due to modeling of environment available.

(ii) The processing of the local information limits the planning capabilities in long term or bigger picture and hence, learning is difficult to be achieved.

(iii) Emergent behaviour which is not yet fully understood making it even more intricate to engineer. Therefore, it is difficult to build task-specific agents.

Q.36. What do you mean by logic based architecture ?

Ans. Logic-based architecture also known as the symbolic-based or deliberative architecture is one the earliest agent architecture that rests on the physical-symbol systems hypothesis.

This classical architecture is based on the traditional artificial symbolic approach by representing and modeling the environment and the agent behaviour with symbolic representation. Thus, the agent behaviour is based on the manipulation of the symbolic representation.

Agent's role in this classical architecture may also be considered as theorem provers. The syntactical manipulation of the symbolic representation is the process of logical deduction or theorem proving. As an instance of theorem proving, the agent specifications outlines how the agent behaves, how the goals are generated and what action the agent can take to satisfy these goals. An example of logic-based architecture formalism is as follows –

(i) Assume that the environment is described by sentences in L and the knowledge base that contains all the information regarding the environment KB = P(L) where P(L) is the set of possible environments.

(ii) For each moment of the time t, an agent's internal state is represented by $KB = (KB_1, KB_2, KB_3, ..., KB_n)$ where $KB_i \in KB$.

(iii) The possible environment states are represented by $S = \{s_1, s_2, ...\}$.

(iv) An agent's reasoning mechanism is modeled by a set of deduction rules, p which are the rules of inference.

(v) An agent perception functions as see : S -> P.

(vi) The agent's internal state is updated by a perception function where next : KB × P -> KB.

(vii) Thus, agent can choose an action from a set $A = \{a_1, a_2, ...\}$, action : KB -> A which is defined in terms of deduction rules. The outcome of an agent's actions is drawn via the function do where do : A × S -> S.

(viii) The decision making process is modeled through the rules of inference p, if a do : A can be derived, the A is returned as an action to be best performed, else if do : A cannot be derived, a special null action is returned.

Q.37. Describe the problem of logic based architecture.

Ans. The simplicity and elegance of logical semantics of the logic based architecture is attractive, there are several problems associated with this approach. Firstly, the transduction problem implies the problem of translating modeling into symbolic representation. It is difficult to translate and model the environment's information into symbolic representation accurately for computation process especially complex environment. Secondly, it is also difficult to represent information in a symbolic form that is suitable for the agents to reason with and in a time constrained environment. Finally, the transformation of percepts input may not be accurate enough to describe the environment itself due to certain faults such as sensor error, reasoning error and

etc. It is very difficult or sometimes impossible to put down all the rules for
situation that will be encountered by the agent in a complex environment
the deduction process is based on set of inference rules. The assumption
calculative rationality where the world does not change in a significant
while the agent is deliberating is not realistic. Assume that on time t_1,
tries to reason an optimal action for that particular time. However, the reason
result may only be available at time t_2 where the environment has already
so much so that the optimal action for time t_1 may not be an optimal action
time t_2. Thus, due to the computational complexity of theorem proving
this approach, it is not appropriate for time constrained domain.

Q.38. Write short note on representational state transfer architecture

Ans. Representational state transfer (REST) is an abstraction of
architecture of the World Wide Web; more precisely, REST is an architecture
style consisting of a coordinated set of architectural constraints applied
components, connectors, and data elements, within a distributed hypermedia
system. REST ignores the details of component implementation and protocol
syntax in order to focus on the roles of components, the constraints upon their
interaction with other components and their interpretation of significant data
elements.

The term representational state transfer was introduced and defined
2000 by Roy Fielding in his doctoral dissertation at UC Irvine. REST has been
applied to describe desired web architecture, to identify existing problems,
compare alternative solutions and to ensure that protocol extensions would
not violate the core constraints that make the web successful. Fielding developed
REST in collaboration with his colleagues during the same period he worked
on HTTP 1.1 and Uniform Resource Identifiers (URI).

The REST architectural style is also applied to the development of web
services. One can characterize web services as "RESTful" if they conform
the constraints described in the architectural constraints section. See the applic
to web services section if you are only interested in the application of REST
web APIs.

REST, on the other hand, is a client-server-based architectural style
is structured around a small set of create, read, update, delete (CRUD)
operations (called POST, GET, PUT, DELETE respectively in the REST world)
and a single addressing scheme (based on a URI, or uniform resource identifier).
REST imposes few constraints on an architecture – SOAP offers completeness.
REST offers simplicity.

REST is about state and state transfer and views the web (and the services
that service-oriented systems can string together) as a huge network of
information that is accessible by a single URI based addressing scheme. The

is no notion of type and hence no type checking in REST – it is up to the applications to get the semantics of interaction right.

Because REST interfaces are so simple and general, any HTTP client can talk to any HTTP server, using the REST operations (POST, GET, PUT, DELETE) with no further configuration. That buys you syntactic interoperability, but of course there must be organization-level agreement about what these programs actually do and what information they exchange. That is, semantic interoperability is not guaranteed between services just because both have REST interfaces.

REST, on top of HTTP, is meant to be self-descriptive and in the best case is a stateless protocol. Consider the following example, in REST, of a phone book service that allows someone to look up a person, given some unique identifier for that person –

http://www.XYZdirectory.com/phonebook/UserInfo/99999

Q.39. Explain about the RESTful web services.

Ans. The web application which follows the REST architecture we call as RESTful web service. RESTful web services uses GET, PUT, POST and DELETE http methods to retrieve, create, update and delete the resources.

The RESTful web services architecture is shown in fig. 2.21.

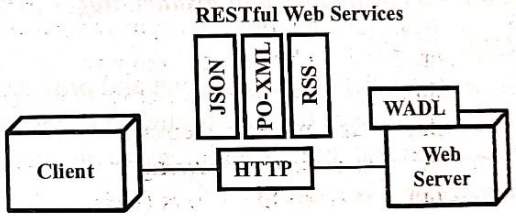


Fig. 2.21 Architecture of RESTFul Web Services, and The Communication between Client and Server

REST (representational state transfer) as the name implies, it has to do with client and server relationship and how state is stored. REST architecture is based on the client/server architecture style. Thus, the requests and responses are built based on the transfering process of the resources. All resources are identified by unique uniform resource identifier (URI), which typically represents a document that captures the state of the resource. Generally, the REST style architecture is much lighter compared to SOAP. It does not require formats like headers to be included in the message, like it is required in SOAP architecture. In the other hand it parses JSON – a human readable language designed to allow data exchange and making it easier to parse and use by the computer. It is estimated to be at around one hundred times faster than XML.

```
{
    "firstname" : "John"
    "lastname" : "Smith"
}
```

Fig. 2.22 A Simple JSON Document

There are several principles that designing RESTFul web service req
Addressability is a REST principle where the datasets are modeled to op
as URI marked resources. Statelessness is another principle that the des
of a REST service will have to follow. This means that every transaction
be independent and must not be related to any previous transaction, as a
data required to perform and process the request are contained on that req
thus, the server will not have to maintain client session data. Uniform inter
requires that an interface is uniform and standard used to access the resou
i.e. using fixed set of HTTP methods. If the service designer holds to
principles, than it is almost guaranteed that the REST application wi
simple and lightweight.

REST is becoming the go to for system interaction which includes
usage of RESTFul web services mostly the way cloud providers expose
services. In the present days, we can easily conclude that most of the
projects are based on RESTFul architecture, in order to create and pro
professional services. Not only the tech giants like Facebook, Google or Tw
use REST these days. This, because thanks to the REST architecture, e
application is able to scale horizontally in the easiest possible way.

Q.40. What are the advantages and disadvantages of REST ?

Ans. Advantages of REST –

(i) REST uses smaller message format and provides cost efficie
over time and better performance because of the JSON messages with wh
makes the communication and there is no intensive processing required.

(ii) Learning curve is reduced.

(iii) It supports stateless communication.

(iv) It's simple to learn and implement.

(v) Efficiently uses HTTP verbs.

(vi) Light bandwidth since its passes message is JSON (JavaSc
Object Notation) format.

(vii) It can use multiple other formats.

(viii) For security it uses HTTP standards.

(ix) REST can be consumed by any client.

(x) It makes data available as resource.

Disadvantages of REST –

(i) It's not suitable for large amount of data.

(ii) Comparative SOAP it does not cover all varieties of web ser
standards like security, transactions etc.

(iii) REST is not reliable.

(iv) REST requests (especially GET) are not suitable for large amount of data.

(v) Latency in request processing times and bandwidth usage.

(vi) REST APIs end up depending on headers for state (such as to route subsequent requests to the same back-end server that handled the previous update, or for authentication). Use of headers is clumsy and ties the API to http as a transport.

Q.41. What do you understand by SOAP ?

Ans. Simple object access protocol (SOAP) is a messaging protocol that allow applications to communicate using HTTP and XML. It represents a fundamentally stateless, one-way message exchange paradigm between nodes. By combining one-way exchanges with features provided by the underlying transport protocol and/or application specific information, SOAP can be used to create more complex interactions such as request/response, request/multiple response, etc.

The process of invoking web services is very important; therefore the SOAP protocol is established to exchange message between service providers and consumers. It is a structured XML message format for exchanging data in a distributed environment. It uses an underlying transport protocol (HTTP, SMTP etc.) through binding. There are two version of SOAP – SOAP version 1.1 and SOAP version 1.2 which has brought some new benefits – It is cleaner, faster, it has better web integration and more it is versatile.

There are three main types of SOAP nodes –

(i) SOAP Sender – Generates and transmits a SOAP message.

(ii) SOAP Receiver – Receives and processes the SOAP message and it also may generate SOAP response, message or fault as a result, and

(iii) SOAP Intermediary (Forwarding or Active) – It is both, a SOAP receiver and a SOAP sender. It receives and processes the SOAP header blocks targeted at it and resends the SOAP message towards and SOAP receiver. This process is illustrated in the fig. 2.23.

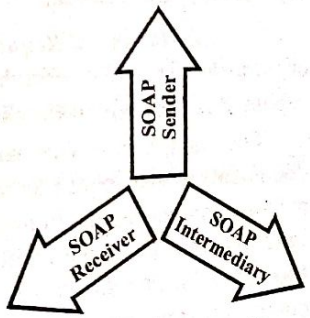


Fig. 2.23 SOAP Nodes

The SOAP message has a structure, which is characterized with two SOAP-specific sub-elements within the overall SOAP envelope (env:Envelope), namely a SOAP Header (env:Header) and a SOAP Body (env:Body).

SOAP is a lightweight independent protocol. It is independent a lightweight because it does not matter what OS or what platform is the serv used from – it responds in the same way in any platform or OS. All this possible because of XML and HTTP protocols.

There are two types of SOAP messaging requests – Remote proced call (RPC) and Document request. Each of them are treated in the follow subsections.

(i) Remote Procedure Call – A remote procedure call repres execution of a procedure in another remote address, usually on another compu in the same network, which is previously coded and it is called as a norm procedure local call. Thus, the programmer will only have to develop the co once, and it does not matter if the call is performed in local or remote circumstanc

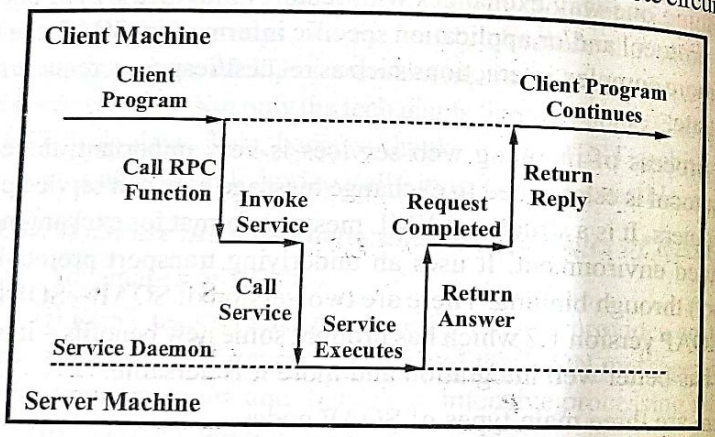


Fig. 2.24 RPC Lifecycle

This procedure represents a client-server model interaction, which implemented through a request/response methodology. These requests a responses are formatted in XML usually, this communication is synchronou which means that when a request is sent, the app is blocked until the respon is processed and returned.

(ii) Document Requests – While transmitting information from t client to server or vice versa through document requests, the XML docum is passed in the body of the SOAP message instead of as parameter.

For example, a service named purchase order expects a document (XM document) as the input message. When the request is sent through SOA message, requesting the PurchaseOrder operation, it must contain a purcha order document as input in the SOAP message. The requests is processed soon as it reaches the server, and when processing is done, another XM document is returned as response, which might contain any kind of informati related to that purchase.

Q.42. What are the advantages and disadvantages of SOAP ?

Ans. The advantages of SOAP are as follows –

(i) Its platform and language independent.

(ii) Uses XML to send and receive messages.

(iii) Its vendor neutral.

(iv) Utilizes WS-* efficiently along with security.

(v) Its firewall friendliness.

(vi) Universally accepted i.e., cost is not too high for implementation.

(vii) It also supports asynchronous messaging.

(viii) It makes data available as services.

(ix) WSDL fully describes SOAP.

The disadvantages of SOAP are as follows –

(i) Too much reliance on HTTP

(ii) It's not stateless

(iii) At times it's slow too because of XML generation. Also the bandwidth gets heavier due to its format for message generation.

Q.43. Compare the SOAP and REST.

Ans. The comparison between SOAP and REST is shown in table 2.1.

Table 2.1 Comparison between SOAP and REST

S.No.	*SOAP*	*REST*
(i)	Changing services in SOAP web provisioning often means a complicated code change on the client side.	Changing services in REST web provisioning not requires any change in client side code.
(ii)	SOAP has heavy payload as compared to REST.	REST is definitely lightweight as it is meant for lightweight data transfer over a most commonly known interface, - the URI.
(iii)	It requires binary attachment parsing.	It supports all data types directly.
(iv)	SOAP is not a wireless infrastructure friendly.	REST is a wireless infrastructure friendly.
(v)	SOAP web services always return XML data.	While REST web services provide flexibility in regards to the type of data returned.

(vi)	It consumes more bandwidth because a SOAP response could require more than 10 times as many bytes as compared to REST.	It consumes less bandwidth beca use it's response is lightweight.
(vii)	SOAP request uses POST and require a complex XML request to be created which makes response-caching difficult.	Restful APIs can be consumed using simple GET requests, inter mediate proxy servers/reverse- proxies can cache their respons very easily.
(viii)	SOAP uses HTTP based APIs refer to APIs that are exposed as one or more HTTP URIs and typical responses are in XML/ JSON. Response schemas are custom per object.	REST on the other hand adds an element of using standardized URIs, and also giving importance to the HTTP verb used (i.e. GET POST/PUT etc.)
(ix)	Language, platform, and trans- port agnostic.	Language and platform agnostic
(x)	Designed to handle distributed computing environments.	Assumes a point-to-point commu nication model – not for distributed computing environment where message may go through or more intermediaries.
(xi)	Harder to develop, requires tools. Is the prevailing standard for web services, and hence has better support from other standards (WSDL, WS) and tooling from vendors.	Much simpler to develop web services than SOAP. Lack of standards support for security, policy, reliable messaging, etc., so services that have more sophi sticated requirements are harder to develop.

UNIT 3

SOFTWARE ARCHITECTURE IMPLEMENTATION TECHNOLOGIES

SOFTWARE ARCHITECTURE DESCRIPTION LANGUAGES (ADLs), STRUTS, HIBERNATE, NODE JS, ANGULAR JS

Q.1. Write short note on software architecture implementation technologies.

Ans. Once the blueprint of software is complete then it is given to the development team for the implementation of product. The design notations are now converted into algorithms or pseudocodes compatible to the environment and platform of development. In this platform is referred as the selected operating system, programming constraints, programming languages, etc. The pseudocodes and algorithms are written for various modules and interfaces of these modules. Finally, now these pseudocodes are converted into programming language codes which can be compiled on a selected compiler. From this program, machine language codes and object codes are generated and executables are obtained. According to research, 40% of the total development cost of the software was consumed for software implementation (see fig. 3.1).

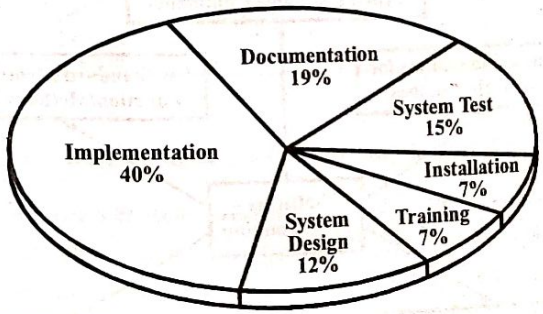


Fig. 3.1 Estimated Cost Distribution Over SDLC

Generally, the implementation phase is not executed in a single go for complex software. This process is distributed into various levels.

(i) Unit Implementation – After completing the design of soft and its components, these components are implemented in some selected h level programming languages. These modules/components are develo individually, independent of each other, in strict accordance with the de document, level language. This is called unit-implementation.

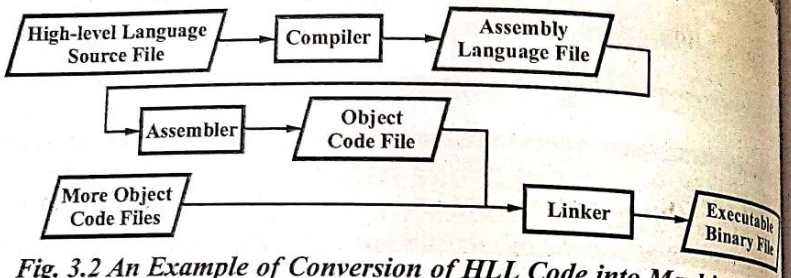


Fig. 3.2 An Example of Conversion of HLL Code into Machine Code

(ii) Unit Integration – Once the units are implemented, befo integration, they are tested under the unit testing phase of SDLC. Aft completing testing, these units are interfaced to make them interact with eac other to implement the tasks of the software collectively. Many a time, som interfacing modules are needed to be written for this. After that, interfacin modules are coded and compiled and executables are generated for them. A the units of the software are not integrated at once, this step is also executed i needed phases. Some units are integrated to form large modules, at first. Afte that, these modules are integrated into large components and finally th complete software is built. These integrations should be performed strictly i accordance with the design obtained at before.

(iii) Software Programming – Now programmer write a code fo these unit/module in HLL like C, C++, Java, PHP, .Net, etc. Moder

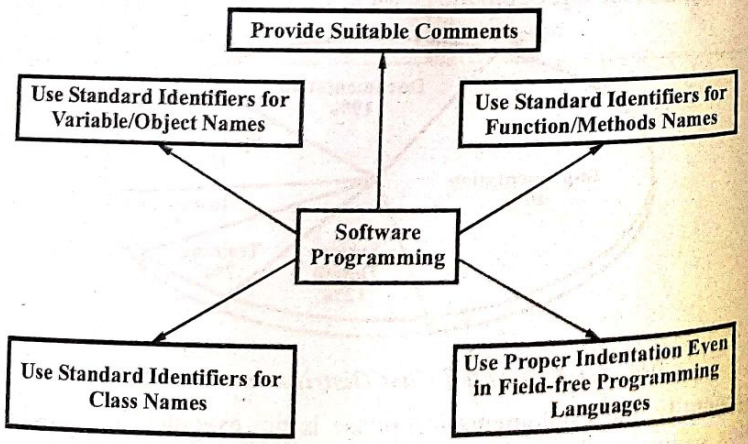


Fig. 3.3 Attributes of Good Programming

programming also supports a mixture of both HLL (high level language) and low level programming code like assembly or binary machine code.

(iv) Programming Paradigms – Each language has its own style of programming, there are called "programming paradigms". Each programming paradigms has its own view point for execution of software programming.

For Example –

(a) Logic Paradigm – In this paradigm, program is written as a set of predicates and truths, whose answers always come in the form of true or false only.

(b) Function Paradigm – In this paradigm, program is a collection of functions invoking each other and implementing the task of the system.

(c) Object Oriented Paradigm – In this paradigm, program are written as a collection of interfacing objects, each having its own details like identity, scope, state, behaviour, etc.

Every programming language support different paradigm like Java support object oriented paradigm.

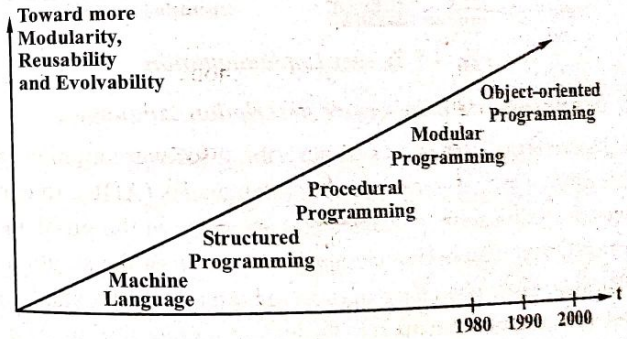


Fig. 3.4 Evolution of Programming Paradigms

(v) System Implementation – At the time of software design process, the software units are integrated and the complete software is thus created as a union of the components. This united software is finally integrated with its environmental constituents, including different hardware, communication lines, routers, third party software products and display panels (see fig. 3.5).

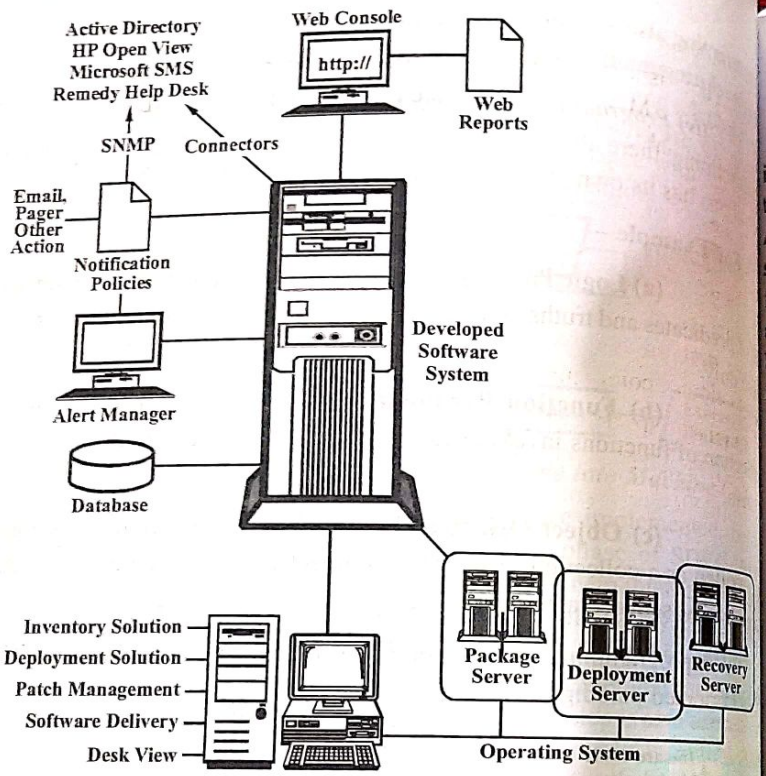


Fig. 3.5 System Implementation

Q.2. Define software architecture description languages.

Ans. More formal approaches to describing software architectures have emerged in form of architecture description languages (ADL). In comparison to requirement specification languages that are more in the problem domain, software architecture description languages are more in the solution domain. Most architecture description languages have both a formal textual syntax and a graphical representation that maps to the textual representation. ADLs should have, ability to represent components and connectors, abstraction and encapsulation, types and type checking, and an open interface for analysis tools. And in architecture description languages shall have component and communication abstraction, communication integrity, model support for dynamic architectures, causality and time support, and relativity or comparison.

At this point, over ten architecture description languages have been presented, e.g. Rapide, and Unicon. In eight ADLs are surveyed and compared on different features.

Q.3. What are the characteristics of ADL.

Ans. There is a large variety in ADLs developed by either academic or industrial groups. Many languages were not intended to be an ADL, but they turn out to be suitable for representing and analyzing an architecture. In principle ADLs differ from requirements languages, because ADLs are rooted in the solution space, whereas requirements describe problem spaces. They differ from programming languages, because ADLs do not bind architectural abstractions to specific point solutions. Modeling languages represent behaviours, where ADLs focus on representation of components. However, there are domain specific modeling languages (DSMLs) that focus on representation of components.

Minimal Requirements – The language must –

(i) Be suitable for communicating an architecture to all interested parties.

(ii) Support the tasks of architecture creation, refinement and validation.

(iii) Provide a basis for further implementation, so it must be able to add information to the ADL specification to enable the final system specification to be derived from the ADL.

(iv) Provide the ability to represent most of the common architectural styles.

(v) Support analytical capabilities or provide quick generating prototype implementations.

ADLs have in Common –

(i) Graphical syntax with often a textual form and a formally defined syntax and semantics.

(ii) Features for modeling distributed systems.

(iii) Little support for capturing design information, except through general purpose annotation mechanisms.

(iv) Ability to represent hierarchical levels of detail including the creation of substructures by instantiating templates.

ADLs Differ in their Ability to –

(i) Handle real-time constructs, such as deadlines and task priorities, at the architectural level.

(ii) Support the specification of different architectural styles. Few handle object oriented class inheritance or dynamic architectures.

(iii) Support the analysis of the architecture.

(iv) Handle different instantiations of the same architecture, in relation to product line architectures.

Q.4. What are the positive and negative elements of ADL ?

Ans. The positive and negative elements of ADL are as follows –

Positive Elements of ADL –

(i) ADLs are a formal way of representing architecture

(ii) ADLs are intended to be both human and machine readable

(iii) ADLs support describing a system at a higher level t previously possible

(iv) ADLs permit analysis and assessment of architectures, completeness, consistency, ambiguity, and performance.

(v) ADLs can support automatic generation of software systems.

Negative Elements of ADL –

(i) There is no universal agreement on what ADLs should repres particularly as regards the behaviour of the architecture.

(ii) Representations currently in use are relatively difficult to pa and are not supported by commercial tools.

(iii) Most ADLs tend to be very vertically optimized toward particular kind of analysis.

Q.5. Discuss on the common concepts of architecture.

Ans. The ADL community generally agrees that software architecture i set of components and the connections among them. But there are differe kind of architectures like –

Object Connection Architecture –

(i) Configuration consists of the interfaces and connections of object-oriented system.

(ii) Interfaces specify the features that must be provided by modul conforming to an interface.

(iii) Connections represented by interfaces together with call grap

(iv) Conformance usually enforced by the programming language

(a) Decomposition – associating interfaces with unique module

(b) Interface conformance – static checking of syntactic rules

(c) Communication integrity – visibility between modules.

Interface Connection Architecture –

(i) Expands the role of interfaces and connections.

(a) Interfaces specify both "required" and "provided" features

(b) Connections are defined between "required" features a "provided" features.

(ii) Consists of interfaces, connections and constraints.

(a) Constraints restrict behaviour of interfaces and connections in an architecture.

(b) Constraints in an architecture map to requirements for a system.

Most ADLs implement an interface connection architecture.

Q.6. What do you mean by Struts ? Explain.

Ans. The Apache Struts web framework is a free open-source solution for creating Java web applications. It uses the Model-View-Controller (MVC) design pattern. The pattern divides the software system into three basic parts which are model, view and controller. The controller is responsible for forwarding the request, and deals with interactions between model and view. The view is used to interface design for displaying the data, like JSP page. The model encapsulates business logic and data processing method, and can direct operate the data, such as access to the database. The model view controller diagram is shown in fig. 3.6.

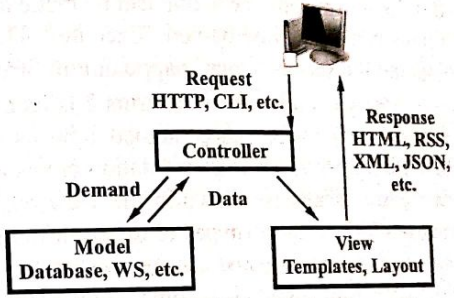


Fig. 3.6 MVC Pattern in General

There are two versions of Struts (1 and 2), but because of their large structural changes.

Q.7. Explain in detail about the Struts 2 with diagram.

Ans. The Struts 2 is the second generation product of Struts. It is the result of a merger between Struts 1 and another framework called WebWork. Struts 2 selects WebWork as the core, using interceptor mechanism to deal with the user's request, this design also make the controller of business logic completely divorced from Servlet API, so Struts 2 can be understood as the update product of WebWork.

The high-level design of Struts 2 follows the well-established model-view-controller design pattern. The MVC design pattern identifies three distinct concerns – model, view and controller. In Struts 2, these are implemented by the action, result, and Filter Dispatcher, respectively. The core components of Struts 2 is shown in fig. 3.7.

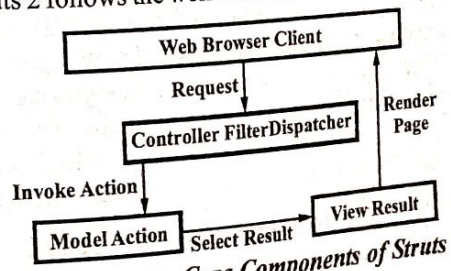


Fig. 3.7 Three Core Components of Struts 2 MVC

Controller-FilterDispatcher – It is the first component to be called w
a HTTP request has been sent by the browser client. The roll of the cont
is played by the Struts 2 FilterDispatcher. The FilterDispatcher is a se
filter (a part of the Java servlet API which can be transparently added to a
application to perform operations against the servlet request or servlet respo
and its purpose is to map each incoming request against an appropriate ac

Model-Action – The model is implemented by the Struts 2 ac
component. It is actually an ordinary Java class which often extends other cla
or implements interfaces belonging to the Struts 2 framework. It is both a p
to put business logic (directly or indirectly through calls to other objects) a
place to save data. (These two are often called the "state" of the applicatio

The developer implements the action classes and decides which feat
he wants to add depending on the requirements of the business logic. If
developer wants to, the action can be made as simple as any Java class wh
implements a method named "Execute". This simplicity comes with lot
"behind the scenes things" happening in Struts 2.

A very useful feature of Struts 2 is its error handling. There's actuall
whole framework for this contained in Struts 2. When it comes to errors rela
to certain form fields (like validation errors and type conversion errors) th
can automatically be shown in the view page, as long as the developer h
mapped different error types of different fields, (either in an XML file or i
method in the action) and is using the Struts 2 tag library. When the validat
fails the String "input" is automatically returned from the action. If the Str
is mapped against a certain page Struts 2 automatically calls that page too.

View-Result – The view is the presentation component of the MV
pattern. In fig. 3.7, we see that the result returns the page to the web brows
This page is the user interface that presents a representation of the applicatio
state to the user. These are commonly JSP pages, velocity templates, or som
other presentation-layer technology. While there are many choices for the vie
the role of the view is clear-cut – it translates the state of the application int
visual presentation with which the user can interact. For the view pages, Str
2 provides a big variety of tags. Three common types of tags are data tag
control-flow tags and User Interface (UI) tags. The data tags can be used f
creating instances of objects and putting them on the value stack among oth
things. Among the control-flow tags you can find things like an "if" tag whi
is useful for conditional expressions. The UI tags generate HTML marku
code. It can for instance generate a "select" tag including the underlying opti
"tag" from a Struts 2 UI tag with only one line.

Q.8. Write short note on interceptors.

Ans. A very important feature of Struts 2 is the interceptors. An intercept
is a reusable component which can be used for separating things like validat

and logging (things that do not really belong to the business logic) from the action code.

The interceptors are invoked before and after the action for every action that uses them. There's a standard stack of interceptors that's included in Struts 2 and it's commonly used by the actions. To use this standard stack the developer just has to extend one class (ActionSupport) in each action class.

Some common issues that are handled by interceptors are – validation, data transfer (that's how the form values are moved into the action), logging, injecting objects from the Servlet API (like HttpServletRequest) into the action through setters, uploading files and exception handling.

Q.9. How to work Struts ? Explain.

Ans. We can see the general processing of Struts 2 in fig. 3.8. First, the browser sends a request to the filter dispatcher which can map this request to an appropriate action. The interceptors that implement common concerns across actions are then called (in the before( ) method) in advance of invoking the action itself. Interceptors built into Struts 2 can perform core processing, like populating request parameters into action classes, performing validation, uploading files, and so on. We can also define custom interceptors as you want. The action class typically invokes the business layer and populates the

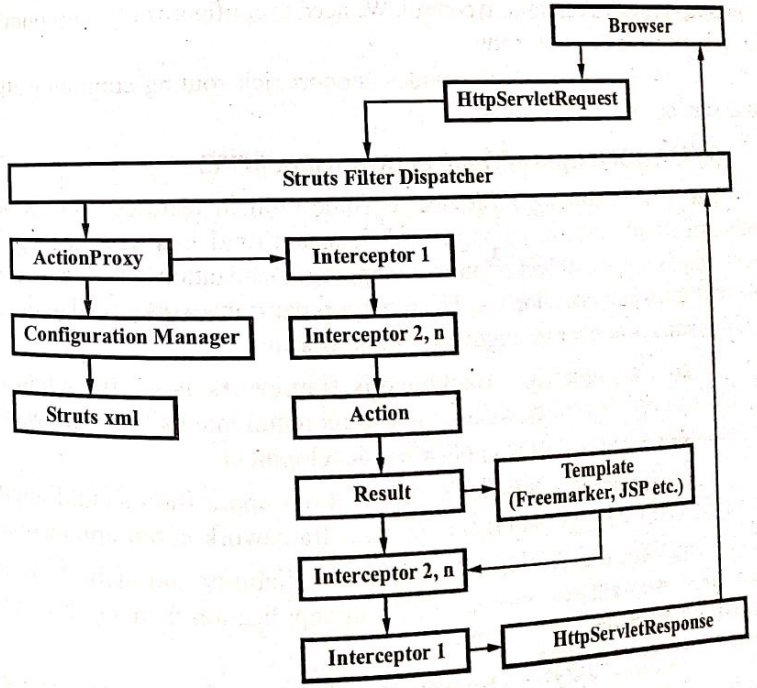


Fig. 3.8 Struts 2 Request Processing

model objects, which are instances variables of the action class. Next, the request is dispatched to the view layer (which is built on a technology like JavaServer Pages, FreeMarker, or Velocity), which renders the GUI. The interceptors are executed again (in reverse order, calling the after( ) method). Finally, the response returns through the filter dispatcher chain.

Ofcourse, Struts 2 framework has more than just its MVC components, and also has a few other important components which are interceptors, OGNL (Object-Graph Navigation Language), and the value stack. These components interact together to implement a cleaner MVC design.

Q.10. What are the advantages of MVC architecture ?

Ans. Advantages of MVC architecture are as follows –

(i) MVC architecture helps us to control the complexity of application by dividing it into three components i.e., model, view and controller.

(ii) MVC does not use server-based forms, that's why it is ideal for those developers who want full control over their application behaviour.

(iii) Test driven development approach is supported by MVC architecture.

(iv) MVC use front controller pattern. Front controller pattern handles the multiple incoming requests using single interface (controller). Front controller provides centralized control. We need to configure only one controller in web server instead of many.

(v) Front controller provides support rich routing communications to design our web application.

Q.11. Explain types of frameworks used in MVC.

Ans. MVC framework provide us some built in features such as form authentication, session management, transactional business logic, web application security, object relational mapping, localization, membership and roles and URL authorization etc. The most popular frameworks available today are backbone.js, ember.js; angular.js and knockout.js.

(i) Backbone.js – Backbone.js framework is useful when our application need flexibility, we have uncertain requirements. Also, we want to accommodate change during application development.

(ii) Ember.js – When we want that our application should interact with JSON API than we should use ember.js framework in our application.

(iii) Angular.js – If we want more reliability and stability in our application, we want extensive testing for our application then we should use angular.js framework.

(iv) Knockout.js – If we want to make a complex dynamic interface of application then knockout.js framework will be very useful for us.

Q.12. Describe the tool and technologies used with MVC.

Ans. There are many tools and technologies which can be used to developed web application with the help of MVC architecture. Depending upon the interest of developers, they can use any of the tools and technologies to develop web application. Here are some tools and technologies which can be used to develop web application using MVC architecture.

Tools – Visual studio is not just only a tool but a complete development environment which provide us facility to create different kinds of application. When we want to develop application using ASP.NET MVC framework then visual studio is very helpful for us.

(i) MYSQL Server – Relational database management server to maintain the database.

(ii) SQL Server – A database engine to maintain database just like MYSQL server.

(iii) MYSQL Workbench – A database design tool.

(iv) Net Beans – IDE (Integrated development environment) provide complete environment to develop different applications.

(v) Glassfish server – Java EE application server.

Technologies –

(i) HTML, CSS, JQUERY, AJAX for designing
(ii) Servlet and Java server pages (JSP) used with Net beans
(iii) EJB (Enterprise Java beans) technologies
(iv) JSTL (Java server pages standard tag libraries)
(v) JPA (Java persistence API)
(vi) JDBC (Java database connectivity)
(vii) ASP.NET MVC used with visual studio.

Q.13. What do you mean by Hibernate ? Explain.

Ans. A very common issue in java development today is how to map the object oriented (OO) Java code against a relational database. This is called object/relational mapping (ORM) and Hibernate is a framework made for providing good solutions to this problem. It's a so called persistence framework.

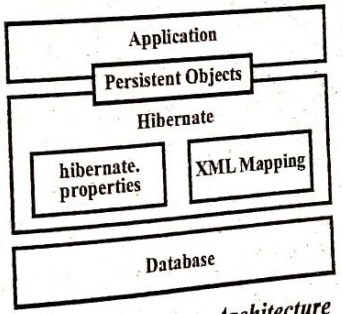


Fig. 3.9 Hibernate Architecture

Hibernate also uses the Java EE 5 APIs JDBC (Java DataBase Connectivity), JPA (Java Persistence API), JTA (Java Transaction API) and JNDI (Java Naming and Directory Interface).

Hibernate consists basically of four parts –

(i) Persistent objects

(ii) ORM (Object/Relational Mapping) mapping files

(iii) Interfaces called by Java applications to perform CRUD (Create – Read – Update – Delete) operations on the persistent classes.

(iv) Interfaces used for Hibernate configuration.

(i) Persistent Objects – A persistent object in Java is just a POJO (Plain Old Java Object) class, which means that it only has an empty property and constructor access methods ("getters" and "setters") for the properties of the class. It handles the data existing physically regardless of application execution. Generally, when developing an application using DBMS data, the business layer of the application handles the application data via SQL for the specific DBMS. But Hibernate-based applications can integrate application data and DBMS with persistent object as the main.

Persistent classes should not contain calls to objects in the Hibernate API. Instead this can be done by a DAO (Data Access Object) or directly (without any help classes) in a small application. The reason for this is that the persistent classes should be able to be used in "normal" Java applications as well as for persistence.

(ii) ORM Mapping Files – ORM mapping files in Hibernate is the xml file like *.hbm.xml. All the mapping information lies in these files, it responsible for mapping the persistent classes against the database tables. Hibernate creates SQL to be executed based on Hibernate Mapping XML. In this case the code of the persistent classes becomes "clean" and just like a POJO.

There are usually one Hibernate mapping file per persistent Java class. The mapping file maps the id property of the Java class against a primary key in the database, all the other properties against table fields and maps relations (of all types) to other persistent classes. Most of these mappings are easy to understand (like the "id" and "property" tags). The hardest part is mapping more complicated relationships (like unidirectional one-to-many which was used in this application) and composite ids.

(iii) Interfaces Called by Java Applications to Perform CRUD Operations on the Persistent Classes – Hibernate provide some interfaces which can easy perform these operations. Four important interfaces that Java developers need to use are SessionFactory, Session, Transaction and Query. The SessionFactory is used to create sessions, there is usually only one SessionFactory per application.

The session is the key object because it has methods for handling all the CRUD operations. It is an object performing a connection between Hibernate and DB connection, which maintains connection until the session is closed after opening a single DB connection on session development. As all the objects (persistent objects) loaded by Hibernate is related with session, the object changes are automatically reflected by session or handled with lazy loading. Session can also create and return a transaction or query object. It is not thread safe so make sure that only one session per thread exists.

The transaction can be used for separating different units of persistence operations, and it must be handled "manually". The hibernate query is used to easy handling of queries against the database. Through use of a query object the user is relieved from having to write SQL code and can instead just take the name of the class and the id value of an object.

(iv) Interfaces used for Hibernate Configuration – An instance of the configuration object must be created first when you build a SessionFactory. This object is responsible for the initialization and configuration of Hibernate, but it needs to know the values of certain paths and properties to the mapping files or annotated classes (if that approach is used). There are two different ways to set these paths and values.

The first way is to use an XML configuration file. This configuration file must called hibernate.cfg.xml and is on the classpath, so Hibernate can find it automatically. In this file you also can specify the mapping files that are used along with all the properties concerning the database connection.

The second way is to use a file which must be called hibernate.properties and must be on the classpath, but this can only be used for the database connection properties. The mapping files must still be specified in the hibernate.cfg.xml file (it's possible to have both).

Q.14. Define NodeJs.

Ans. NodeJs or just Node is the most important component of the MEAN stack. It provides the JavaScript development environment. It is built based on Google's V8 engine. Both Node and V8 are implemented in C and C++ for less memory consumption and faster performance. Node is based on asynchronous I/O eventing model designed for developing scalable network applications. It fires callbacks on events, and each client event generates its own callback. If no work is to be done, Node is sleeping. While Node works on a single thread, it can serve many clients. Almost no function in Node directly performs I/O, they are handled by higher-order functions. Node presents the event-loop as a runtime construct, but unlike some other technologies, node does not have a blocking start-the-event-loop call. It simply enters the loop and exist upon completion similar to browser JavaScript. Node also has different modules that help take advantage of a multiprocessor environment such as creating child processes, sharing sockets etc.

Q.15. Write short note on Express.js.

Ans. Express is a server side framework built in the NodeJs environment. It handles the client requests to the server and manages routing and HTTP methods such as GET, POST, PUT etc. Express configures Middlewares, which are basically functions that use the request, response objects and call the next middleware in the stack. It is the Middleware's responsibility to either end the request-response-cycle or pass the call next( ) to call the next middleware so the request is not left hanging.

An express application is created by calling the express( ) exported by express e.g. app = express( ). The app object is used to perform various operations and provide services by express. Express listens to a socket connection on a path or on a specified host and port number. Then using one of the METHOD( ) functions such as app. get( ) where app is the express application object and get( ) is the METHOD function, start the request-response-cycle of the appropriate middleware.

To configure middle wares the app. Route( ) returns an instance of a single route, which can be handles by HTTP methods and optionally middle wares. The app. render( ) is used to render HTML view files using a call back. Express uses template view engines to render views.

Q.16. What is Angular JS ?

Ans. Angular JS is framework manage by Google, it help build responsive sites. Angular JS use to make a smooth web performance. Angular JS is a toolset for building the framework most suited to your application development. It is fully extensible and works well with other libraries. Every feature can be modified or replaced to suit your unique development workflow and feature needs.

Angular JS is a JavaScript framework. It can be added to an HTML page with a <script> tag. Angular JS extends HTML attributes with directives, and binds data to HTML with expressions.

AngularJS extends HTML with new attributes.AngularJS is perfect for single page applications (SPAs).AngularJS is easy to learn. The idea turned out very well, and the project is now officially supported by Google.

AngularJS is a structural framework for dynamic web applications. It lets you use HTML as your template language and lets you extend HTML's syntax to express your application components clearly and succinctly. Its data binding and dependency injection eliminate much of the code you currently have to write. And it all happens within the browser, making it an ideal partner with any server technology. It was originally developed by Misko Hevery and Adam Abrons.

HTML is great for declaring static documents, but it falters when we try to use it for declaring dynamic views in web-applications. AngularJS lets you extend HTML vocabulary for your application. The resulting environment is extraordinarily expressive, readable, and quick to develop.

Q.17. Describe the important parts of Angular JS.

Ans. Fig. 3.10 shows the various parts of Angular JS as follows–

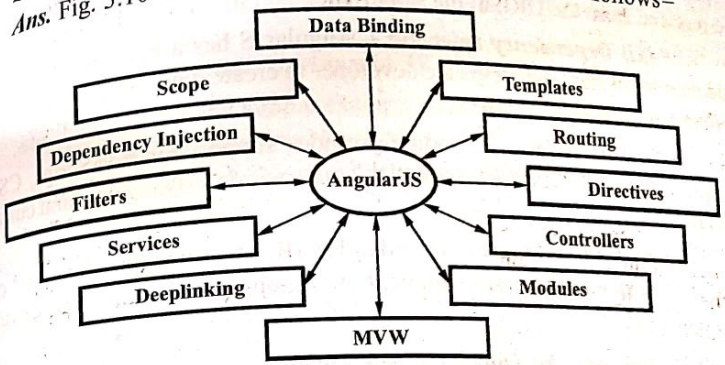


Fig. 3.10 Parts of Angular JS

(i) Data Binding – It is the automatic synchronization of data between model and view components.

(ii) Scope – These are objects that refer to the model. They act as a glue between controller and view.

(iii) Controller – These are JavaScript functions bound to a particular scope.

(iv) Services – AngularJS comes with several built-in services such as $http to make a XMLHttpRequests. These are singleton objects which are instantiated only once in app.

(v) Filters – These select a subset of items from an array and returns a new array.

(vi) Directives – Directives are markers on DOM elements such as elements, attributes, CSS, and more. These can be used to create custom HTML tags that serve as new, custom widgets. AngularJS has built-in directives such as ngBind, ngModel etc.

(vii) Templates – These are the rendered view with information from the controller and model. These can be a single file (such as index.html) or multiple views in one page using partials.

(viii) Routing – It is concept of switching views.

(ix) Model View Whatever – MVW is a design pattern for dividing an application into different parts called model, view, and controller, each with distinct responsibilities. AngularJS does not implement MVC in the traditional sense, but rather something closer to MVVM (Model-View-ViewModel). The angular JS team refers it humorously as model view whatever.

(x) Deep Linking – Deep linking allows you to encode the state application in the URL so that it can be bookmarked. The application can be restored from the URL to the same state.

(xi) Dependency Injection – AngularJS has a built-in dependency injection subsystem that helps the developer to create, understand, and test applications easily.

(xii) Modules – A module provides support for JavaScript, CSS transition and CSS3 keyframe animation hooks within existing core and custom directives.

Since ng-* attributes are not valid in HTML specifications, data-ng-* can also be used as a prefix. For example, both ng-app and data-ng-app are valid in AngularJS.

Q.18. Describe the Angular JS directives.

Ans. AngularJS directives allow the developer to specify custom and reusable HTML-like elements and attributes that define data bindings and the behaviour of presentation components. Some of the most commonly used directives are –

(i) ng-app – This directive starts and AngularJS application.

(ii) ng-bind – This directive binds the AngularJS application data to HTML tags.

(iii) ng-model – This directive binds the values of Angular JS application data to HTML input controls.

(iv) ng-model-options – Provides tuning for how model updates are done.

(v) ng-class – Lets class attributes be dynamically loaded.

(vi) ng-controller – Specifies a JavaScript controller class that evaluates HTML expressions.

(vii) ng-repeat – This directive repeats HTML elements for each item in a collection.

(viii) ng-show & ng-hide – Conditionally show or hide an element depending on the value of a Boolean expression. Show and hide is achieved by setting the CSS display style.

(ix) ng-switch – Conditionally instantiate one template from a set of choices, depending on the value of a selection expression.

(x) ng-view – The base directive responsible for handling routes that resolve JSON before rendering templates driven by specified controllers.

(xi) ng-if – Basic if statement directive that allow to show the following element if the conditions are true. When the condition is false, the

element is removed from the DOM. When true, a clone of the compiled element is re-inserted.

(xii) ng-aria – A module for accessibility support of common ARIA attributes.

Q.19. Give the example of Angular JS with source code.

Ans. Following is the simple example of Angular JS with source code –

```
<html ng-app="myNoteApp">
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/
1.4.8/angular.min.js">
</script>
<body>
<div ng-controller="myNoteCtrl">
<h2>MyNote</h2>
<p><textarea ng-
model="message" cols="40" rows="10"></textarea></p>
<p>
<button ng-click="save( )">Save</button>
<button ng-click="clear( )">Clear</button>
</p>
<p>Number of characters left:<span ng-bind="left( )">
</span>
</p>
</div>
```

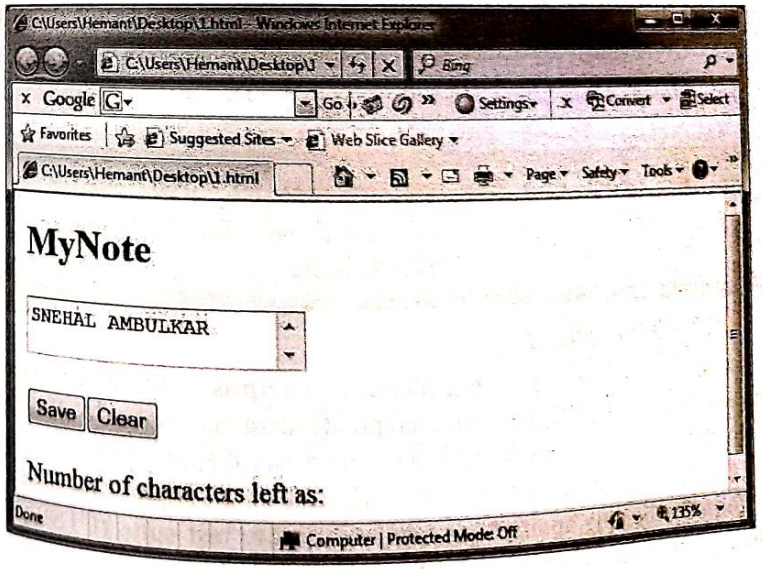


Fig. 3.11 Result

Q.20. What are the advantages of Angular JS?

Ans. The advantages of Angular JS are as follows –

(i) Angular JS provides capability to create single page applica... in a very clean and maintainable way.

(ii) Angular JS provides data binding capability to HTML. Thus ... gives user a rich and responsive experience.

(iii) Angular JS code is unit testable.

(iv) Angular JS uses dependency injection and make use of separa... of concerns.

(v) Angular JS provides reusable components.

(vi) With Angular JS, the developers can achieve more functiona... with short code.

(vii) In Angular JS, views are pure html pages, and controllers writ... in JavaScript do the business processing.

On the top of everything, Angular JS applications can run on all majo... browsers and smart phones, including android and iOS based phones/tablet...

Q.21. What are the general features of AngularJS ?

Ans. The most important general features of AngularJS are –

(i) AngularJS is a efficient framework that can create Rich Interne... Applications (RIA).

(ii) AngularJS provides developers an options to write client sid... applications using JavaScript in a clean model view controller (MVC) way.

(iii) Applications written in AngularJS are cross-browser compliant... AngularJS automatically handles JavaScript code suitable for each browser.

(iv) AngularJS is open source, completely free, and used by thousands... of developers around the world. It is licensed under the Apache license version... 2.0.

J2EE, JSP, SERVLETS, EJBs, MIDDLEWARE, JDBC, JNDI, JMS, RMI AND CORBA ETC. ROLE OF UML IN SOFTWARE ARCHITECTURE

Q.22. What is J2EE ?

Ans. J2EE stands for Java 2 platform, enterprise edition. The Java 2 ... platform, enterprise edition defines a simple standard that applies to all aspects ... of architecting and developing multi-tier server based applications.

It defines a standard architecture composed of an application model, a ... platform for hosting applications, a compatibility test suite (CTS) and a ... reference implementation in the J2EE specification.

The primary concern of J2EE is the platform specification – it describes the runtime environment for a J2EE application. This environment includes application components, containers, resource manager drivers, and databases. The elements of this environment communicate with a set of standard services that are also specified. The fig. 3.12 shows the J2EE server model and in which tiers the J2EE components reside.

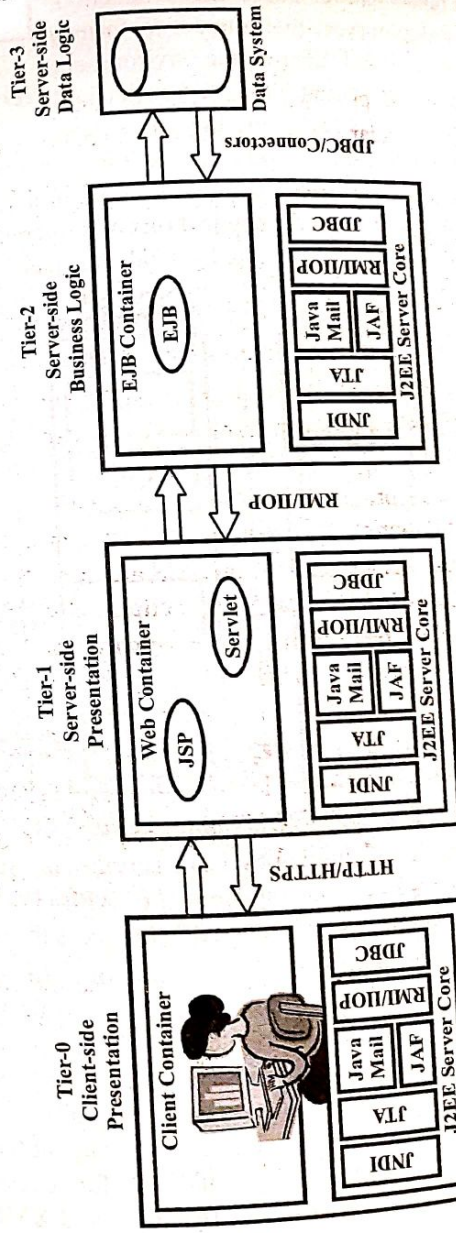


Fig. 3.12 J2EE Server Model

Q.23. What is J2EE technology ?

Ans. From a pure technological point of view, three areas can be recognized within the J2EE specifications as shown in fig. 3.13.

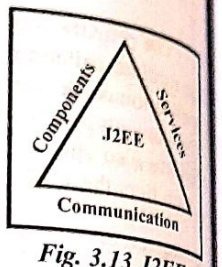


Fig. 3.13 J2EE Technologies at High Level

Component Technologies – Software components are binary units of independent production, acquisition, and deployment that interact to form a functioning system. The J2EE runtime environment defines the application client, Applet, Servlet and JavaServer Pages (Servlets/JSPs), and Enterprise Java Beans (EJB) as the four types of application components that a J2EE product must support. Each type of component is executed in a separated container. Fig. 3.14 shows an overview of the application components in their respective containers along with the logic they usually handle in an enterprise application.

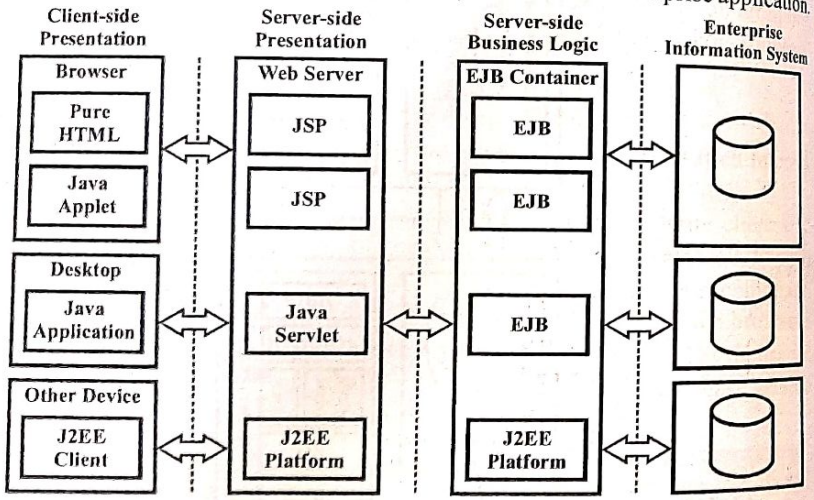


Fig. 3.14 J2EE Components

Services and Service Technologies – The services are functions that are accessible to the components via a standard set of APIs. For example, using the Java Naming Directory Interface (JNDI) APIs to access the naming services.

Communication Technologies – The essence of J2EE is the definition of a distributed, object-oriented infrastructure. Components need to communicate with each other in this distributed world. Therefore, the containers need to provide the appropriate communication mechanisms to make this happen. An example of the communications included in the J2EE standards are RMI/IIOP protocols for remote method calls, messaging technologies like JavaMail for programmatic access to e-mail and JMS for accessing messaging technologies, data format technologies like JAR files and XML.

Q.24. Define JSP. *(R.G.P.V., Dec. 2010)*

Ans. JSP stands for Java Server Pages. JSP is a standard Java extension used to simplify the creation and management of dynamic Web pages. The stages of JSP lifecycle are page translation, compilation, loading & initialization, request handling and destroying.

Q.25. Can I create XML pages using JSP technology ? *(R.G.P.V., June 2012)*

Ans. Yes, the JSP specification does support creation of XML documents. For simple XML generation, the XML tags may be included as static template portions of the JSP page. Dynamic generation of XML tags occurs through bean components or custom tag that generate XML output.

Q.26. What is Java Server Pages (JSP) technology ? How does JSP technology work ? *(R.G.P.V., June 2011, Dec. 2015)*

Or

What is JSP technology ? How does JSP technology works ? *(R.G.P.V., Dec. 2016)*

Ans. Java Server Pages (JSP) is Sun's solution for developing dynamic Web sites. JSPs allow to separate the dynamic content of a Web page from its static presentation content. Programming in servlets is very complex and requires additional files, such as Web.xml and Java class files, to generate the required Web pages. JSP offers an easier approach to build dynamic Web pages. JSP documents consist of HTML tags and special tags, known as JSP tags. HTML tags are used to create static page content and JSP tags are used to add dynamic content to Web pages. JSP pages are compiled into a Java servlet by a JSP translator. This Java servlet is then compiled and executed to generate an output for the browser (client).

```
<html>
<head>
<title> First Page </title>
</head>
<body>
<H3> Today is :
<%=new java.util.Date( ) %>
</H3>
</body>
</html>
```

Fig. 3.15 First JSP

Java server pages are saved with .jsp extension.

Fig. 3.15 shows a JSP example.

In JSP, Java codes are written between <% and %> tags. So it takes the following form: <%= some expression %>. In this example, we have used <%= "new java.util.Date( )" %> which displays the current date.

Q.27. What are the advantages of JSP over various server side programming techniques ?

Ans. Advantages of JSP are as follows –

(i) JSP vs. Active Server Pages (ASP) – ASP is a similar technology from Microsoft. The advantages of JSP are twofold –

(a) The dynamic part is written in Java, not Visual Basic or other MS-specific language as is the case with ASP, therefore, it is more powerful and easier to use.

(b) It is portable to other operating systems and non-microsoft based Web servers.

(ii) JSP vs. Pure Servlets – JSP is similar to a servlet. But it is easier to write regular HTML than to have a zillion println statements that generate the HTML. It also separates the look from the content.

(iii) JSP vs. Server-side Includes (SSI) – SSI is a technology for including externally-defined pieces into a static Web page. JSP is better since it lets the use of servlets instead of a separate program to generate that dynamic part. Besides, SSI is really only intended for simple inclusions, not for real programs that use form data, make database connections, and the like.

(iv) JSP vs. JavaScript – JavaScript can generate HTML dynamically on the client. This is a useful capability, but handles only those situations where the dynamic information is based on the client's environment. With the exception of cookies, HTTP and form submission data is not available to JavaScript. And, since it runs on the client, JavaScript cannot access server-side resources like databases, catalogs, pricing information, etc.

(v) JSP vs. Static HTML – Regular HTML cannot contain dynamic information. JSP is feasible to augment HTML pages that only benefit marginally by the insertion of small amounts of dynamic data.

Q.28. Discuss steps for creating simple JSP page.
(R.G.P.V., June 2015, 2016)

Ans. The major stages of JSP life cycle are as follows –

(i) Page Translation – The Web container translates the JSP document into an equivalent Java code i.e. servlet. The objective of page translation is to convert a document chiefly consisting of HTML/XML code to one that has more of Java code which can be executed by the JVM. The translation of JSP can take place either at the time of deploying the JSP or at the time when a request for the JSP is received for the first time.

(ii) Compilation Stage – The JSP container compiles the Java source code for the corresponding servlet and converts it into Java byte (class) code. The container can decide to either discard the code or retain it for debugging after the class file is generated. Generally, most containers discard the generated Java source code by default.

(iii) Loading and Initialization – The JSP container loads and instantiates the servlet that has been generated and compiled in the translation and compilation stages respectively. The Web container as part of this process perform three operations, namely, loading, instantiation and initialization.

(iv) Request Handling – The Web container uses only those objects of a JSP equivalent servlet that are initialized successfully to handle the client request.

(v) Destroying – The servlet container decides to destroy the JSP page's servlet instance, that is, if it decides to end the services provided by the servlet instance.

Q.29. Explain the various components (or elements) of JSP in detail.
Or
Explain different types of tags or scripting element used in JSP.
(R.G.P.V., June 2014)
Or
Explain basic component of JSP. ***(R.G.P.V., June 2015, 2016)***

Ans. A JSP page is composed of directives, declarations, scriptlets, expressions, standard actions, and custom tags.

(i) Directives – Directives are the instructions to the JSP container. These instructions tell the container that some action needs to be taken. For example, if we want to import some standard/non-standard Java classes or packages into our JSP, we can use a directive for that purpose. Directives are contained inside special delimiters, namely <%@ and %>. The syntax of JSP directive is –

<%@ directive attribute = "value"%>

where directive may be –

(a) *page* is used to provide the information about it.

Example – <%@ page language = "java"%>

(b) *include* is used to include a file in the JSP page.

Example – <%@ include file= "/header.jsp"%>

(c) *taglib* is used to include the custom tags in the JSP pages (custom tags allows us to defined our own tags).

Example – <%@ taglib uri= "tlds/taglib.tld" prefix= "mytag"%>

and attribute may be –

(a) language= "java"

This tells the server that the page is using the Java language. Current JSP specification supports only Java language.

Example – <%@page language = "java"%>

(b) extends= "mypackage.myclass"

This attribute is used when we want to extend any class. The comma(,) can be used to import more than one packages.

Example – <%@page language = "java" import= "java.sql.*,mypackage.myclass"%>

(c) session = "true"

By default, this value is true. When this value is true, session data is available to the JSP page otherwise not.

Example – <%@page language= "java" session= "true"%>

(d) errorPage = "error.jsp"

This is used to handle the unhandled exceptions in the page. Which is not handled by exceptions in the program.

Example – <%@page language= "java" session= "true"
errorPage = "error.jsp"%>

(ii) Declarations – Declarations should be used, when we need to make any declarations in the JSP page. The syntax of making declarations is as follows –

```
<%!
    //Declare all the variables here
%>
```

Here are some declaration examples –

```
<%! int j = 0; %>
<%! int x, y; double z; %>
<%!circle a = new circle (2.0); %>
```

Fig. 3.16 shows an example of using declarations in JSP.

```
<html>
    <body>
    <%! int counter = 0; %>
    The page count is now : <% = + + counter %>
    </body>
    </html
```

Fig. 3.16 JSP Declarations

(iii) Scriptlets – Scriptlets are one or more Java statements in a JSP page. The syntax of scriptlets is as follows –

<% scriptlet code; %>

JSP scriptlets begin with <% and ends with %>. We can embed any amount of Java code in the JSP scriptlets. JSP Engine places these code in the _jspService( ) method. Variables available to the JSP scriptlets are –

(a) *request* represents the clients request and is a subclass of HttpServletRequest. Use this variable to retrieve the data submitted along with the request.

(b) *response* is subclass of HttpServletResponse.

(c) *session* represents the HTTP session object associated with the request.

(d) *out* is an object of output stream and is used to send any output to the client.

Example –

```
<table border = 2>
<%
    for(int j = 0; j< n; j ++) {
%>
    <tr>
    <td> Number </td>
    <td> <%=i+1%></td>
    </tr>
<%
    }
%>
</table>
```

(iv) Expressions – Expressions are means of accessing the values of Java variables or other expressions that directly yield a value. The results of an expression can be merged with the HTML page that gets generated. The syntax of JSP expressions is as follows –

<% = "Any thing" %>

For example –

<% = "Hello World"%>

This code will display Hello World.

(v) Standard Action – The syntax for the following JSP standard actions is –

(a) Include

<jsp:include page = "<filename>"/>

This inclusion of file is dynamic and the specified file is included in the JSP file at run-time. That is, output of the included file is inserted into the JSP file.

(b) Forward

<jsp:forward page= "<filename>"/>

This will redirect to the different page without notifying browser.

(vi) Custom Tags – The syntax for custom tags is –

taglib

<%@ taglib uri= "<tag library uri>" prefix = "<tagprefix>"%>

The attributes for the above tag are –

(a) uri = "<relative path of the tag library uri>"

(b) prefix = "<tagprefix>"

Prefix is alias name for the tag library name.

Q.30. What is servlet ? Explain life cycle of servlet.
(R.G.P.V., June 2011, Dec. 2015)

Ans. A servlet is just a Java class that a Web server instantiates when the server is started. A particular method is called on this instance when the server receives certain HTTP requests. Code in the servlet method can obtain information about the request and produce information to be included in an HTTP response by calling methods on parameter objects passed to the method. When the servlet returns control to the server, the server creates an HTTP response from the information dynamically generated by the servlet.

Life Cycle of Servlet – Servlets follow the life-cycle which governs the multithreaded environment that servlets run in and provides an insight to some of the mechanisms available to a developer for sharing server-side resources.

The servlets follow a three-phase life namely, initialization, service, and destruction.

The initialization is the first-phase of the servlet life-cycle. It represents the creation and initialization of resources, the servlet may require in response to service requests. The javax.servlet.Servlet interface must be implemented by all servlets. This interface defines the init( ) method to match the initialization phase of a servlet life-cycle. When a container loads a servlet, it invokes the init( ) method prior to servicing any requests.

The second phase of a servlet life-cycle is the service phase. It represents all interactions along with requests until the servlet is destroyed. The servlet interface matches the service phase to the service( ) method. The service( ) method of a servlet is invoked once as per the request. Then, it is responsible to generate the response to that request.

The third and final phase of the servlet life-cycle is the destruction phase. It represents the removal of the servlet from the container. Here, the servlet interface defines the destroy( ) method to correspond to the destruction life-cycle phase. When a servlet is to be removed, a container calls the destroy( ) method.

Q.31. What is the conceptual difference between JSP and servlet ?

Ans. Coding a JSP seems to be simpler than coding the corresponding servlet. In the JSP, we do not have to write complex Java code and worse still, HTML inside that Java code. We can write HTML tags, and wherever needed, write Java code in between HTML tags. In other words, servlets are HTML inside Java, whereas JSPs are Java inside HTML. This is shown in fig. 3.17.

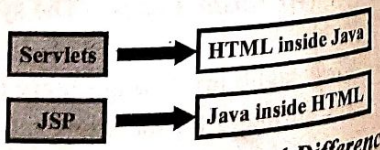


Fig. 3.17 The Conceptual Difference between Servlets and JSP

Q.32. Explain the features of Java servlet that make it more useful over CGI.
(R.G.P.V., June 2012, Dec. 2015)

Ans. Servlet provide many advantages in comparison with CGI. First, performance is more better. Servlet run within the address space of a Web server. It is not required to create a separate process to handle each client request. Second, servlets are platform-independent since they are written in Java. Third, the Java security manager on the server enforces a set of restrictions to protect the resources on a server machine. Finally, the full functionality of the Java class libraries is available to a servlet. It can communicate with applets, databases, or other software through the sockets and RMI mechanisms.

Q.33. Write the advantages of servlet.

Ans. Advantages of servlet are as follows –

(i) Servlets are multithreaded. That is, whenever the servlet container receives a request for the execution of a servlet, the servlet container loads the servlet in its memory, and assigns a thread of this servlet for processing this client's requests. If more clients send requests for the same servlet, the servlet container does not create new servlet instances. Instead, it creates new threads of the same servlet instance, and allocates these thread instances to the different client requests. This makes the overall processing faster, and also reduces the memory demands on the servlet container/Web server.

(ii) Because servlets execute inside a controlled environment (i.e., container), they are usually quite stable and simple to deploy.

(iii) Because servlets are nothing but more specific Java programs, they inherit all the good features of the Java programming language, such as object-orientation, inherent security, networking capabilities, integration with other Java Enterprise technologies, etc.

Q.34. Write the disadvantages of servlet.

Ans. Disadvantages of servlet are as follows –

(i) The dynamic construction of HTML documents is still low-level i.e., fragments of textual HTML are written to an output stream, eventually forming a complete HTML document. There are no compile time guarantees that the result is valid according to some schema or even well-formed. For nontrivial programs it can be more difficult to avoid this kind of bug.

(ii) It is difficult to separate the concerns of programmers and HTML designers.

(iii) The control flow through a session involving different servlets can be difficult to follow for the programmer. The connection between the code that generates an HTML document with a form and the code that subsequently handles the form field data is often not obvious. Moreover, the way session state is managed through the setAttribute/getAttribute mechanism

is vulnerable. Since the session control-flow is not always clear, the programmer must be careful about assuming that certain attributes have or have not been set in an earlier interaction. Also, as we have seen it is necessary to downcast all results from getAttribute, which is an additional source of errors that are not detected statically.

Q.35. Discuss the architecture of servlet. ***(R.G.P.V., June 2017)***

Ans. Fig. 3.18 shows the interaction between server and servlet. A web Server handling a servlet request generally operates as follows –

(i) When an HTTP request message is received by a servlet-capable server, it first determines, based on the URL of the request, that the request should be handled by a servlet. For example, a server might be configured to treat any request to any URL for which the path component begins with servlet as a request that should be handled by a servlet.

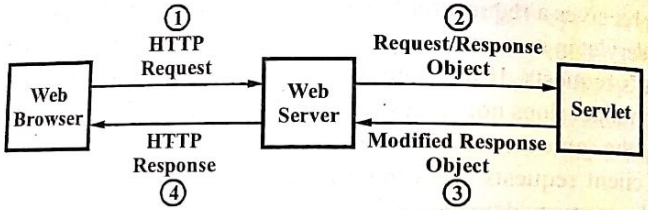


Fig. 3.18 Architecture of Servlet

(ii) The server then determines from the URL which servlet should handle the request and calls a method on that servlet. Two parameters are passed to the method – an object implementing the HttpServletRequest interface, and an object implementing the HttpServletResponse interface. Both of these interfaces are defined as part of the Java Servlet API, which is implemented by the server. The first object provides methods that can be used by the servlet to access the information contained in the HTTP request message received by the server. The second object can be used to record information that the servlet wishes to include in the HTTP response message that the server will send to client (i.e., Web browser) in reply to the request.

(iii) The servlet method executes, typically calling methods on the HttpServletRequest and HttpServletResponse objects passed to it. The information stored in the latter object by the servlet method typically includes an entire HTML document along with some HTTP header information like the document content-type. When the servlet method has finished its processing it returns control to the server.

(iv) The server formats the information stored in the HttpServletResponse object by the servlet into an HTTP response message, which it then sends to the client (i.e., Web browser) that initiated the HTTP request.

Q.36. How to write servlets in Java ? Explain with example. ***(R.G.P.V., June 2015)***

Ans. Servlets are Java classes which service HTTP requests and implement the javax.servlet.servlet interface. Web application developers typically write servlets that extend javax.servlet.http. When we want to write our own servlet, we need to write a Java class that extend this HttpServlet. HttpServlet, an abstract class that implements the servlet interface and is specifically designed to handle HTTP requests. It has three methods initializing servicing, and destruction of servlets.

Sun's standard definition of a Java servlet –

```
public abstract class HttpServlet extends GenericServlet
{
    public void init( );
    public void service (HttpServletRequest request, HttpServlet
Response response);
    void destroy( );
}
```

A servlet example is as follows –

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class OrderServlet extends HttpServlet
{
    public void init( )
    {
    system.out.println("In init( ) method");
    }
    public void service(HttpServletRequest request, HttpServlet
Response response).
    {
        system.out.println("In doGet( ) method");
    }
    public void destroy( )
    {
        system.out.println("In destroy( ) method");
    }
}
```

Our servlet class extends HttpServlet class provided by Sun. Our servlet is able to inherit the service( ) Java method from this HttpServlet. Similarly, the HttpServlet itself, in turn, has been inherited from GenericServlet. The

GenericServlet defines the other two methods, namely init( ) and destroy( ). HttpServlet inherits these from GenericServlet and passes them on to our OrderServlet. The OrderServlet has some code written in the methods. The servlet container would call them as and when it required.

Q.37. How does a servlet communicate with a JSP page ? How do I use a scriptlet to initialilze a newly instantiated bean ? (R.G.P.V., June 2012)

Ans. When a servlet JSP communication is happening, it is not just about forwarding the request to a JSP from a servlet. There may be a need to transfer a string value or an object itself.

Following are the steps in servlet JSP communication –

(i) Servlet instantiates a bean and initializes it.

(ii) The bean is then placed into the request.

(iii) The call is then forwarded to the JSP page, using request dispatcher.

Following is a servlet and JSP source code example to perform servlet JSP communication.

```
public class ServletToJSP extends HttpServlet
{
    public void doGet (HttpServletRequest request, HttpServletResponse
                        response) throws ServletException, IOException
    {
    //Communicating a simple string message
        String message ="Example source code of Servlet to JSP
                                Communication";
        request.setAttribute("message", message);
    //Communicating a vector object
        Vector vecobj = new Vector( );
        vecobj.add("Servlet to JSP communicating an object");
        request.setAttribute("vecBean", vecobj);
    //Servlet JSP communication
        RequestDispatcher reqDispatcher = getServletConfig( )
                    getServletContent( ).getRequestDispatcher("/JSP/
                                                    Javapapers.JSP");

        reqDispatcher.forward(request, response);
    }
}
```

Example JSP source code : (Javapapers.JSP)

```
<html>
    <body>
        <%
            String message = (String) request.getAttribute ("message");
            out.println("Servlet communicated message to JSP : "+message);
            Vector vecobj = (Vector) request.getAttribute ("vecBean");
            out.print/n("Servlet to JSP communication of an object:" +
                                                    vecobj.get(0));

        %>
    <%body>
</html>
```


Scriptlet to Initialize a Newly Instantiated Bean – A JSP:useBean action may optionally have a body. If the body is specified, its contents will be automatically invoked when the specified Bean is instantiated. Generally, the body will contain Scriptlet or JSP:setProperty tags to initialize the newly instantiated bean, although you are not restricted to using those alone.

The following example shows the bean initialized to the current date when it is instantiated.

```
value = "<%=java.text.DateFormat.getDateInstance( ).format(new.java.
                                                    util.Date( ))%>

<% – Scriptlets calling bean Setter methods go here ...%>"
```

Q.38. Explain how Java servlets perform session handling. ***(R.G.P.V., Dec. 2016)***

Or

How Java servlets perform session handling ? (R.G.P.V., June 2017)

Ans. A session is a collection of HTTP requests shared between a client and a Web server over a period of time. A session lifetime is set to thirty minutes by default. The session is destroyed after expiring its lifetime and all its resources are returned back to the servlet engine. In session tracking, we gather the detailed information from the Web pages and store user generated information. The server side applications keep some state information and maintain a dialouge with the client. The most common example of session handling is shopping cart application. In this application, a client accesses the server many times using the same browser and visits several Web pages. Thereafter the client buy some items offered for sale at the Web site. In this case, if each transaction is being served by a stateless server side object, and there is no identification from the client's side on each request, it would be impossible to maintain a filled shopping cart over several HTTP requests from the client. If the user visits a Web page multiple times and chooses different items to be added to the shopping cart in each visit, the stateless nature of HTTP might not relate each visit to the same session. Therefore, even writing a stateless transaction data to persistent storage would not be a solution in this regard. Therefore, session tracking involves identifying the user sessions by related ID numbers and tying the requests to their session by using the said ID

number. Cookies or URL are the typical mechanisms for session tracking. In accordance with the servlet specification, the servlet container in the application server implements session tracking through HTTP session objects, which are instances of a class and implement the javax.servlet.http.HttpSession interface. When a servlet uses the getsession( ) method, it creates an HTTP session object and the stateful client interaction. There is only one HTTP session object for each client in each application. Some session tracking techniques are cookies, hidden form fields, URL rewriting and SSL sessions.

The server creates a small text file, called as a cookie and associates this particular user with that cookie. The cookie is created by the server, and sent to the browser along with the first HTTP response. The browser accepts it and stores it inside the browser's memory. Whenever the browser sends the next HTTP request to the server, it reads this cookie from its memory and adds it to the request. Thus, the cookie keeps travelling between the browser and the server for every request-response pair.

The following syntax can be used to implement hidden form fields in an HTML page –

<INPUT TYPE = "HIDDEN" NAME = "SESSION" VALUE = ".....">

This hidden field can be used to store information about the session. However, it only works when every page is dynamically generated by a form submission. That's why, general session tracking cannot be supported by the hidden form fields, but can only support tracking within a specific series of operations.

The URL rewriting mechanism uses the encode URL( ) method of the response object and the session ID is encoded into the URL path of a request. In the following example, the name of the path parameter is jsessionid –

http://host:port/myapp/index.html?jsessionid=1234

The value of the rewritten URL is used by the server to look up session state information and pass it to the servlet. It is similar to cookies. Although, cookies are typically enabled, but to ensure session tracking using URL Rewriting, use encodeURL( ) in your servlets, or encodeRedirectURL( ), for redirecting to a resource.

SSL is used for protection of data in transmit that encompasses all network services using TCP/IP to support typical application tasks of communication between the clients and the servers. It is an encryption technology that runs on top of TCP/IP and below application level protocols, such as HTTP. SSL ensures the security of data transported and routed through HTTP. SSL is designed to make use of TCP as a communication layer protocol to provide a reliable end-to-end secure and authenticated connection between two points over a network. It is used mostly in HTTP server and client applications.

Q.39. What is EJB ?

Ans. Enterprise JavaBeans (EJB) is not an alternative for JSP/Servlets, Struts, etc. Instead these technologies, such as JSP/Servlets and Struts use EJB for performing business processing. EJB is also referred to as *transaction-oriented middleware*. In other words, it takes care of the *heavy-duty* work, such as transaction management, security, load balancing etc., for providing better throughput.

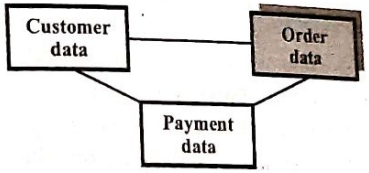


Fig. 3.19 Components Concept

EJB encourages component-based development. For example, suppose that we need to create a shopping cast-based application. Then, we can think of three main aspects customer data, order data, and payment data. EJB looks at these three as components and aims at building an integration layer between them. This concept is shown in fig. 3.19.

Q.40. What is the EJB architecture and how it is related to the J2EE ?

Ans. The EJB architecture is a server-side component-based architecture that models business logic of enterprise architecture. EJB is at the heart of the J2EE architecture, which provides the big picture of enterprise applications. J2EE provides all infrastructure services to EJB, such as JDBC, JNDI, JMS and JTA.

The EJB architecture simplifies enterprise applications by basing them on standardized, modular, reusable components. The EJB architecture provides a complete set of services to those components and handles many details of application behaviour automatically. By automating many of the time cosuming and difficult task of application development, J2EE technology eanables enterprise developers to focus on adding value, which enhances business logic, rather than building infrastructure.

Q.41. What features does EJB provide ?

Ans. EJB provides the following features –

(i) Transaction Management – A developer can specify that your enterprise beans need a transactional environment by setting a specific property of the bean you develop. This means that the code inside the enterprise bean would automatically run inside a transaction, which is managed by the EJB infrastructure. That is, you can be rest assured that either the entire code in the enterprise bean would be executed completely or none at all. For this, the enterprise bean, in turn, calls an API of the EJB container implicitly. A software developer does not have to worry about it. Notice that the transaction management applies to the whole bean, and not to any specific error checking

within that bean. That is, suppose an end-of-day stock update bean performs the following two steps –

(a) Read each record (sales/purchase) from a daily transaction file.

(b) Update the corresponding master file record with the results of the transaction.

Now, when the code for the above operation is ready, the developer could set the transaction-enabled property of the bean to true, that is whenever the bean is executed, the responsibility of making sure that the whole transaction file is processed, and the master file updated correctly, is left to the bean. If a failure occurs, the bean would automatically roll back the changes done to the master file since the bean was invoked thus ensuring database consistency.

(ii) Persistence – Persistence means storing the state of an object in some form of permanent storage like a disk. When a developer indicates to the EJB container that the wishes to make an enterprise bean persistence-enabled, the EJB container automatically ensures that the last state of the enterprise bean object is preserved on the disk, and later on retrieved. This is important in situations where an enterprise bean has to store certain values on the server-side. For example, suppose a user visits a shopping cart. Then the user disconnects. Now, the state of the user's transaction can be recorded in a database or the enterprise bean managing the user conversation can store it. When the user connects back, say after three days, the enterprise bean brings back the values for that user from the disk, so that the user can continue purchasing or complete his purchasing process.

(iii) Remote Awareness – EJB is all about remote objects. Since objects and clients can be in different parts of the world, it is important that all these objects are allowed to communicate over networks. A developer does not have to write any kind of network code to make the enterprise beans that he develops, network-aware/distributed. The EJB container automatically does this. For this, the EJB container wraps the enterprise bean in a network-enabled object. This network-enabled object intercepts calls from remote clients, and delegates then to the appropriate enterprise bean.

(iv) Multi-user Support – The EJB container implicitly adds the code required for allowing an enterprise bean to work with several clients at the same time. It provides built-in support for multithreading, instantiating multiple instances of an enterprise bean whenever needed, etc., automatically.

(v) Location Transparency – The client of an enterprise application bean does not worry about the actual physical location of the bean. It is handled by the EJB container.

In addition to these, the EJB server is responsible for creating instances of new components, managing database connections, threads and sockets, etc.

Q.42. Discuss about session beans.

Ans. A session bean contains some specific business-processing related logic. It is a Java object that encapsulates the necessary code for performing a set of operations corresponding to one more business processes. The business processes are business logic, business rules or workflow. A session bean is a reusable piece of software. For example, a session bean *Update salary* could be used to update the salary of one or all employees.

The term session stems from the fact that the life of a session bean is limited to the time for which a client uses the services of a session bean. When the client code invokes the services of a session bean the application server creates an instance of the session bean. The session bean then services the client as long as necessary. When the client completes the job and disconnects, the application server destroys the instance of the session bean.

An instance of a session bean is unique i.e., no two or more clients can share a single session bean. This is essential for ensuring transaction management. If two or more clients use the services of the same instance of a session bean, there would be confusion. Because they might accidentally access the same data. To avoid this, session beans can be made thread-safe, so that two or more session beans can share code, but maintain separate copies of data. However, this is an implementation issue that needs to be decided by the application server vendor. From a common developer's perspective, a session bean is never shared among users.

A client never explicitly creates an instance of, or destroys, a session bean. It is always done by the EJB container running inside the application server. It ensures optimum utilization of bean resources. Also, this frees the client from issues such as stack management, memory management etc., and provides him with an interface. The application server does the bean management.

There are two types of session beans, *stateful session beans* and *stateless session beans*.

(i) Stateful Session Beans – A session bean corresponds to a business process. However, a business process may complete in just one stroke, or it might need more than one interaction between the client and the server. The concept of transactions is more relevant for the latter. In this case, when the clients and servers interact more than once during the entire lifecycle of these interactions, the state of the application must be preserved. If all the steps during these set of interactions completes sucessfully, the operation as a whole can be considered to be successful. For handling such situations, or more correctly transactions, stateful session beans are very important.

A typical situation that needs multiple interactions between the client and the server is a shopping cart in an e-commerce application. Initially, the application might present a shopping cart to the user. Then the user might add items to it,

remove items from it, or change some of the items. This interaction goes on for quite some time. In such a scenario, a stateful session is very useful.

(ii) Stateless Session Beans – An interaction between a Web browser and a Web server consists of a request-response pair. It means that the browser sends a request for an HTML document, the server finds the document and sends it back as a response to the browser. In such situations, where the client and the server interact in a request-response mode, and then forget about it, there is no necessity for maintaining the state of the application. Stateless session beans are candidates for such business processes.

For example, in an e-commerce application, the client might enter credit cards details, such as its issuing company, number, expiry date and the customer's name. It might request a stateless session bean to verify this credit card. This stateless session bean might perform the verification, and send a success or failure message back to the client, depending on whether the credit card is valid. This requires no more interactions between the client and the server. Such business scenarios are useful for stateless session beans.

Q.43. Can a stateless session bean maintain state ?

Ans. Yes, a stateless session beans can contain non-client specific state across client invoked methods. For example, states such as socket connection, database connections, reference to an EJBObject, and so on can be maintained. However, they cannot have state specific to any client across client-invoked methods.

Q.44. What are the classes and interfaces that make a stateful session bean ?

Ans. A stateful session bean is an EJB component, which extends two interfaces – a home interface (EJBHome) and a remote interface (EJBObject), and a bean class that implements the SessionBean interface.

Q.45. Describe the steps to implement a stateless session bean in detail.
(R.G.P.V., June 2013)

Ans. The implementation of the remote home interface SignOnHome, remote interface SignOn and the stateless session bean class SignOnEJB is discussed below –

Defining the Home Interface – The home interface, SignOnHome, is defined as follows –

```
package day04;
import java.rmi.RemoteException;
import javax.ejb.*;
public interface SignOnHome extends EJBHome {
    SignOn create( )
        throws CreateException, RemoteException;
}
```


Therefore, the create( ) method on the home interface is called to create a bean instance and a reference to the remote interface is received.

Defining the Component Interface –

The SignOn remote interface is defined as follows –

```
package day04;
import java.util.*;
import javax.ejb.*;
public interface SignOn extends EJBobject
{
    public boolean validateUser(String login, String password)
    throws InvalidLoginException, RemoteException;
}
```

This interface has one business method, *validateuser*, which is callable by the client. The remote interface is a Java RMI interface. Hence, method arguments and return types of a remote method must be legal types for RMI/IIOP and the method must have *java.rmi.RemoteException* in its throws clause. *InvalidLoginException* is a customized application exception thrown by the *SignOn* enterprise bean to report an unsuccessful login attempt.

Implementing the Enterprise Bean Class – The implementation of the SignOnEJB enterprise bean class is given below. The stateless session bean implements the javax.ejb.SessionBean interface. It implements the methods setSessionContext( ), ejbCreate( ), ejbActivate( ), ejbPassivate, and ejbRemove. The ejbCreate method creates an instance of javax.naming.InitialContext and looks up the environment naming context through the InitialContext under the name "java:comp/env". Besides, it implements the validate user method that accepts the user's login name and password as parameters and returns true if the login is successful. The method throws InvalidloginException if the login name and password are invalid.

```
package day04;
import java.util.*;
import java.ejb.*;
import javax.naming.*;
public class SignOnEJB implements SessionBean {
private SessionContext ctx;
private context environment;
public SignOnEJB( ) {
    print("The container created this instance.\n");
}
public void setsessioncontext(SessionContext c) {
    print("The container called the setSessionContext method");
    print("so that the bean instance can be initialized.\n");
```

```
            try {
                InitialContext ic = new InitialContext( );
                environment = (context) ic.lookup ("java:comp/env");
            }   catch (NamingException ne) {
                throw new CreateException ("could not look up context");
            }
        /* Methods ejbActivate and ejbPassivate are not used by stateless
session beans
        /*
        public void ejbActivate( ) { }
        public void ejbPassivate( ) { }
        public void ejbRemove ( ) {
            print("This instance is in the process of being removed");
            print("by the container.\n");
        }
        public boolean validateUser (String userName, String password)
            throws InvalidLoginException {
            try {
                String storedPassword = (String) environment.lookup
                                                              (userName);
                if (storedPassword.equals(password)) {
                    return true;
                }
                else {
                    throw new InvalidLoginException("Invalid login/
                                                              password");
                }
            }   catch(NamingException ne) {
                throw new InvalidLoginException ("Invalid login/
password");
            }
            void print (String s) {
                System.out.println(s);
            }
        }
```

Writing the Exception Class – The InvalidLoginException class is defined below. This class has been derived from java.lang.Exception.

```
package day04;
public class InvalidException extends Exception
{
    public InvalidLoginException( )
```



```
    {
        super( );
    }
    public InvalidLoginException(Exception e)
    {
        super(e.toString( ));
    }
    public InvalidLoginException(String s)
    {
        super(s);
    }
}
```

Declaring the Deployment Descriptors – The deployment descriptor describes a component's deployment settings. The ejb-jar.xml deployment descriptor for the SignOn enterprise bean is given below. It describes the enterprise bean's deployment properties like its bean type and structure.

```
<!xml version = "1.0"?>
<!DOCTYPE ejb-jar PUBLIC
'-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans2.0//
EN'
'http://java.sun.com/dtd/ejb-jar-2-0.dtd'>
<ejb-jar>
    <enterprise-beans>
    <session>
    <ejb-name>SignOnEJB</ejb-name>
    <home>day04.SignOnHome</home>
    <remote>day04.SignOn</remote>
    <ejb-class>day04.SignOnEJB</ejb-class>
    <session-type>Stateless</session-type>
    <transaction-type>Container</transaction-type>
    <env-entry>
    <env-entry-name>student</env-entry-name>
    <env-entry-type>java.lang.String</env-entry-type>
    <env-entry-value>password</env-entry-value>
    </env-entry>
    <env-entry-name>student 1</env-entry-name>
    <env-entry-type>java.lang.string</env-entry-type>
    <env-entry-value>password1</env-entry-value>
    </env-entry>
    </session>
    </enterprise-beans>
    </ejb-jar>
```

Q.46. Discuss about entity beans.

Ans. The entity beans represent database objects, which either bring data from databases to running applications, when required, or update data into the databases, when requested by the application. An entity bean is an in-memory object representation of persistent data stored in a database. Thus, an entity bean can be used for modelling data items such as a bank account, an item in a purchase order, an employee record, and so on. They can also represent real-world objects such as products, employees and customers. Thus, entity beans do not associate themselves directly with business processes. They are useful for modelling data elements only. In contrast, session beans handle the business processes.

Thus, *transfer amount* could be a business process, which can be modelled by a session bean. This process would need to credit one account and debit another. The information regarding which accounts to credit and debit, and the end result, must be represented by one or more entity beans. Thus, session beans use entity beans whenever they want to access or update persistent data from databases.

Entity beans were devised for a reason that whereas most of today's databases are in the relational form, the applications that make use of these database, use the object technology. Thus, a mapping is needed between the relational view and the object view. This is provided by entity beans. They permit session beans to treat the persistent data in relational tables, as objects.

For example, in the *transfer amount* example, an entity bean could be used by a session bean for reading the account details in an *account* object. Suppose the name of account holder is abc. The session bean might then issue a *transfer* instruction on that account i.e., the abc *account* object. For example, the instruction could be in form abc.transfer (1100, 2100, 200). As far as the session bean is concerned, it is sticking to the object-oriented paradigm of creating objects, and manipulating them with the help of their methods such as *transfer*. However, internally, the *account* object might represent one particular row of an accounts relational table, which gets updated as a result of this operation. The entity bean hides these implementation details from the session bean, and allows it to treat every piece persistent data as real-world objects. This is shown in fig. 3.20.

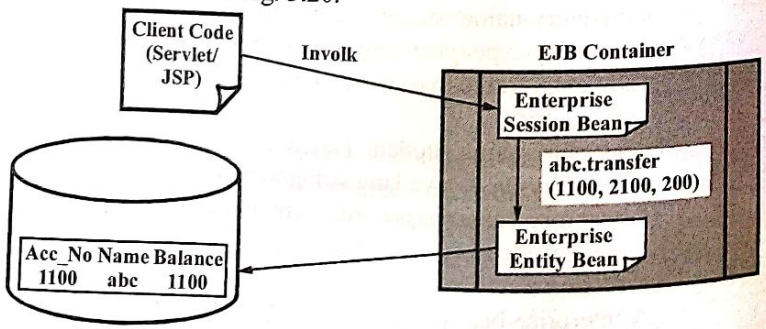


Fig. 3.20 Session and Entity Beans

Obviously, entity beans have a much longer life than session beans, because entity beans are used for representing data that is preserved across user sessions. If an application crashes, or a client disconnects from a server for some reason, an entity bean can always be reconstructed from its underlying database.

Entity beans differ from session beans in one more respect. Whereas only one user can use a single session bean instance at a time, an entity bean can service more than one client at the same time. Another point is that since entity beans model data stored in databases, they are vary useful when there is huge data existing in legacy applications that need to be Web-enabled by using EJB.

Entity beans are of two types, *bean-managed persistent entity beans*, and *container-managed persistent entity beans*.

(i) Bean-managed Persistent Entity Beans – In this type of entity beans, the responsibility of locating, changing, and writing back the data between an entity bean and the persistent storage of the database is left to the developer. It means that the developer has to explicitly write program code for performing all these tasks.

(ii) Container-managed Persistent Entity Beans – In this type of entity beans, the EJB container performs automatic persistence on behalf of the developer. The developer does not hard code any statements required for locating, changing or writing back the data between an entity bean and the underlying database. Instead, the developer describes what he wants, and the EJB container performs the translation from the developer's description to the actual program code. It makes the application database independent, because the developer does not write code specific for a database, and instead, leaves it to the EJB container.

Q.47. Write short note on message driven beans.

Ans. Message-driven beans are stateless components that are asynchronously invoked by the container as a result of the arrival of a Java Message Service (JMS) message. A message-driven bean receives a message from a JMS destination, like a queue or topic, and performs business logic on the basis of the message contents, like logic to receive the process a client notification.

A shopper making an online purchase order is an example of a message-driven bean. An order bean could inform a credit verification bean. A credit verification bean could check the shopper's credit card in the background and send a notification message for approval. Since this notification is asynchronous, the shopper does not have to wait for the background processing to complete.

Q.48. What is middleware concept ? Explain with diagram. ***(R.G.P.V., June 2011)***

Ans. Fig. 3.21 shows the basic idea of middleware at a high level.

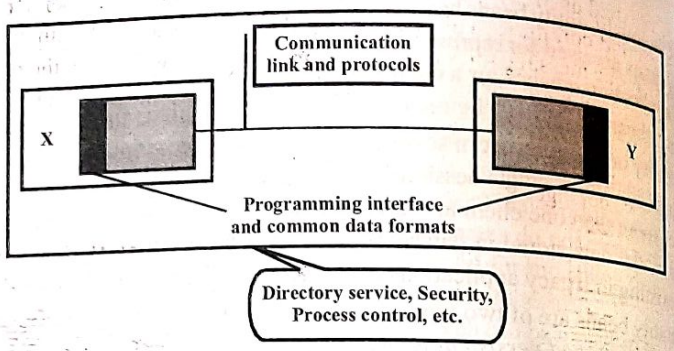


Fig. 3.21 Middleware Concept

Obviously, if two computers X and Y want to (remotely) communicate with each other and perform any business operations, middleware has an important role to play in many ways. The various aspects of middleware are discussed below –

(i) The ***communication link*** and the ***protocols*** allow the basic communication between X and Y. The physical communication between X and Y can be done using wired networks such as LAN or WAN, or it can also be done wirelessly such as cellular network or wireless LAN. However, the important thing is that here are two sets of protocols that we are talking of. The first is the lower layer communication protocol, which is responsible for the actual transmission of bits from X to Y and vice-versa. Another is the middleware protocol, which allows the dialog between X and Y. The middleware protocol assumes the availability and reliability of the lower layer protocol.

(ii) The *programming interface* and the *common data formats* specify how X and Y can communicate with each other through the middleware. That is, we are not worried about the communication of X and Y directly in this sense, but we are worried about their communication with the middleware. The data formats used should enable the middleware to communicate between X and Y in a uniform manner, and the programming interface should also be such that there are no gaps in communication.

(iii) The other elements are add-ons. For example, the *directory service* would help X to locate the various services available on Y, and to make use of them as appropriate. *Security* would ensure that the communication between X and Y is safe. *Process control* would ensure that Y is able to handle multiple requests efficiently, providing good response time.

Q.49. Write short note on Message Oriented Middleware (MOM). ***(R.G.P.V., Dec. 2010, June 2013)***

Ans. Asynchronous communication allows the parties to communicate indirectly through a message queue. The software that manages these message queues is called as ***Message Oriented Middleware (MOM).*** The sender sends a message and continues with its other work without waiting for an acknowledgement. The message goes into a message queue and waits until the receiver is ready to receive and process it. Usually, both, the sender and the receiver, have message queue software set up at their respective ends. The message queues, in turn, store outgoing messages into and read incoming messages from a local messages database. This is shown in fig. 3.22.

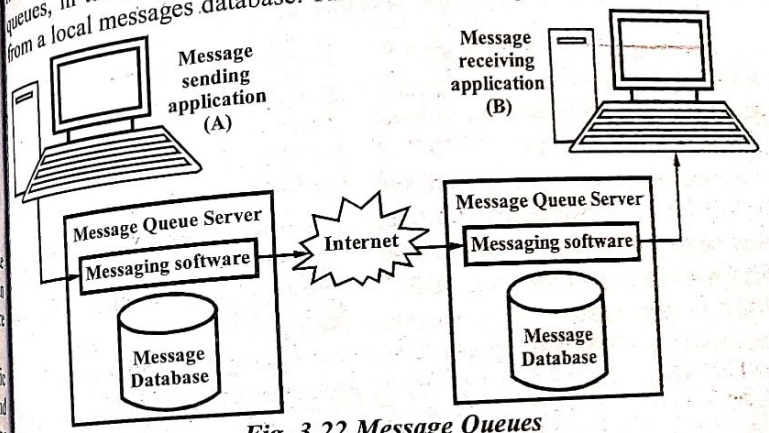


Fig. 3.22 Message Queues

Here, the sender A sends a message for the receiver B. The message goes to the message queue server of A. This server has messaging software and a message database. The new message is added to the queue maintained in the database. The messaging software is responsible for depositing these messages in the database scheduling them, retrieving them one by one when their turn comes, and then transporting them to the receiver. In this case, it is received by the messaging software of the message queue server at B. The software at B stores it in the database, until B retrieves it.

Q.50. Write in brief on ODBC. ***(R.G.P.V., June 2010, Dec. 2010)***

Or

What is ODBC ? ***(R.G.P.V., June 2015)***

Ans. Many enterprise applications that are created using Java EE technology, need to interact with databases for storing application specific information. For example, search engines use database for storing information about the Web pages. Job portals use database for storing information about the candidates and employers, accessing the site for searching and advertising jobs on the Internet. However, interacting with database requires database connectivity. This can be achieved by using the ODBC driver, which is used with Java Database Connectivity (JDBC) to interact with database such as Oracle, MS Access, My SQL and SQL server. JDBC is an API which is used in Java programming for interacting with database.

Q.51. What is JDBC ?

Ans. Java Database Connectivity (JDBC) is a set of classes and interfaces for allowing any Java application to work with an RDBMS in a uniform manner. It means that the programmer need not worry about the differences in various RDBMS technologies, and can consider all RDBMS products as same DBMS, which all work in a similar fashion.

The conceptual view of JDBC is shown in fig. 3.23.

From fig. 3.23, the main idea of JDBC is to provide a layer of abstraction to our programs while dealing with the various RDBMS products. Instead of our programs having to understand and code in the RDBMS-specific language, they can be written in Java. It means that our Java code needs to speak in JDBC. JDBC, in turn, transforms our code into the appropriate RDBMS language.

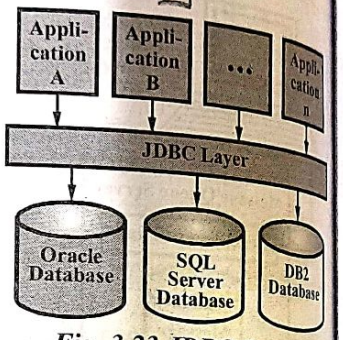


Fig. 3.23 JDBC Concept

The JDBC interface is contained in the packages java.sql and javax.sql. JDBC uses more interfaces than classes, so that different vendors are free to provide an appropriate implementation for the specification. Overall, about 30 interfaces and 9 classes are provided, such as Connection, Statement, Prepared Statement, ResultSet, and SQLException.

(i) Connection Object – It is the pipe between a Java program and the RDBMS. It is the object through which commands and data flow between our program and the RDBMS.

(ii) Statement Object – This object sends SQL commands, using the pipe, that can be executed on the RDBMS. There are three types of commands that can be executed by using this object –

(a) Statement Object – This object is used to define and execute static SQL statements.

(b) PreparedStatement – This object is used to define and execute dynamic SQL statements.

(c) CallableStatement – This object is used to define and execute stored procedures.

(iii) ResultSet Object – The result of executing a Statement is usually some data. This data is returned inside an object of type ResultSet.

(iv) SQLException Object – This object is used to deal with errors in JDBC.

Q.52. What are the components of JDBC ?

Ans. The components of JDBC drivers are as follows –

(i) The JDBC API
(ii) The JDBC DriverManager
(iii) The JDBC Test Suite
(iv) The JDBC-ODBC bridge.

Q.53. Explain various types of JDBC drivers in Java. *(R.G.P.V., Dec. 2010)*

Ans. JDBC drivers can be obtained from a number of sources, provide different levels of functionality, and vary in their use of supporting software. Sources for JDBC drivers are the J2SE SDK itself, database vendors, and third-party developers. Functionality is defined by the version of JDBC supported by the driver.

The supporting software required by a driver relies on the driver's type. One type, called a Type 4 driver, is Java software that is loaded onto the client machine and supports direct communication with a DBMS. Type 4 drivers require no supporting software other than the DBMS itself. All of the other types depend on additional software to complete their functionality. A Type 3 driver is Java client software that communicates with intermediary server software, which in turn communicates with one or more actual database systems. A Type 2 driver permits JDBC software on a client machine to communicate with non-Java DBMS communication software also loaded on the client, which in turn communicates with the DBMS itself. Finally, a Type 1 driver connects JDBC software on the client to Open Database Connectivity (ODBC) software also residing on the client. The ODBC software can then be used to access any of the many ODBC-compliant DBMSs. A Type 1 driver is also known as a JDBC - ODBC bridge.

Q.54. Explain how to connect to a local MS access database using a JDBC driver ?

Or

Write the steps of setting up an Open Database Connectivity (ODBC). *(R.G.P.V., June 2011, Dec. 2015)*

Or

What is database connectivity ? Explain the basic connectivity, middleware support, ODBC and JDBC technologies. *(R.G.P.V., June 2014)*

Ans. We wish to connect to a Microsoft Access database on the same Windows XP machine that hosts our Java servlet. Because Access is ODBC-compliant, we can do this with a JDBC-ODBC bridge (Type 1 driver). There is no JDBC driver software to install in this case, because a Type 1 driver is included in the J2SE 1.4 SDK. We also do not need to install any ODBC client software, because this is installed by default on Windows XP. But we do need to tell the Windows ODBC client software about our database.

For the moment, suppose that we have already created an Access database named Employee and stored it in a file named Employee.mdb. Next, we associate an ODBC driver with this database as follows. From the Windows Control Panel, select **Administrative Tools** and then **Data Sources (ODBC)**. In the ODBC Data Source Administrator window, select the **User DSN** tab and click the Add...button. In the Create New Data Source window, select Microsoft Access Driver (*.mdb) and click the Finish button. In the ODBC Microsoft Access Setup window, enter EmployeeDB in the Data Source Name text box, click the select....button in the Database area, and browse to Employee.mdb file. After selecting file and clicking OK in the select Database window, click OK in the ODBC Microsoft Access Setup Window. Now, EmployeeDB will appear in the list of User Data Sources in the ODBC Data Source Administrator window. Finally, click OK in this window.

If we don't have an Access database already, then we can create through the ODBC tool as follows. Follow the instructions in the preceding paragraph, but in the ODBC Microsoft Access Setup Window, after entering the Data Source Name click the Create....button instead of the Select....button. Then give our database a file name, like Employee.mdb, and click OK. We should see a pop-up message telling us that your database was successfully created. As in the preceding paragraph, click OK several times to complete the ODBC setup.

Once the ODBC association has been made, we can connect to Employee database from a Java servlet (or from any Java program) as follows –

```
String dataSourceName = "EmployeeDB";
String dbURI = "jdbc:odbc:" + dataSourceName;
try {
    Class.forName ("sun.jdbc.odbc.JdbcOdbcDriver");
    java.sql.Connection con = java.sql.DriverManager.getConnection
                                                    (dbURI, "", "");
    con.close( ); //To close the connection
}
catch(Exception e) {
    e.printStackTrace( );
}
```

The static forName( ) method of Class causes the Java virtual machine to load the specified class, which is the name of the SDK-supplied Type 1 JDBC driver. When getConnection( ) is subsequently called on the JDBC DriverManager class, it uses this driver to establish a connection with the database indicated by its first argument, a URI. When connecting to a database through the JDBC-ODBC bridge, the URI is a URN consisting of the prefix jdbc:odbc: followed by an ODBC data source name. In this example, we use the data source name EmployeeDB that we associated with the Employee database using the Data

Sources administrative tool. The other two arguments to getConnection( ) are the log-in name and password associated with the ODBC EmployeeDB data source, which by default are both the empty string. The connection object returned by getConnection( ) will also use the driver loaded by forName( ) for all database communication needed to support JDBC methods called on this object.

Q.55. Explain how to connect to MySQL database using a JDBC driver?

Ans. We wish to connect to a MySQL DBMS running on a server machine from a Java servlet running on a client machine. This is accomplished using a Type 4 (Java) driver provided by MySQL.

Let's assume that we have already created a MySQL database named Employee on a machine with host name db.example.net. Also, assume that the MySQL DBMS accepts TCP/IP connections (local or remote) on its default port 3306 and that the user *someuser* with password *mypassword* has been granted full access to the Employee database from the client machine. We will need to download the MySQL connector/J driver JAR file to the client machine.

After uncompressing the download file, we see two directories, one beginning with mysql-connector. In this directory, we will find the driver JAR file, which also has a name beginning with mysql-connector. Copy this file to the *shared/lib* subdirectory under our JWSDP installation directory. The next time we start or restart Tomcat, this JAR file will automatically be added to the CLASSPATH for any servlets run by the server.

Once the Connector/J JAR file has been installed, the following code can be used to access the Employee database from a servlet, or from any Java code that has the Connector/J JAR file on its CLASSPATH –

```
//set the following four string's as appropriate
String host="db.example.net:3306";
String dbname= "Employee";
String username = "someuser";
String password = "mypassword";
String dbURI = "jdbc:mysql:" + "//" + host + "/" + dbname + "?user="
                              + username + "password =" + password;

try {
    //The newInstance( ) call is a work around for
    //some broken Java Implementations
    Class.forName("com.mysql.jdbc.Driver").newInstance( );
    connection  con = DriverManager.getConnection(dbURI);
    //JDBC calls to con methods go here.....
    con.close( );
}
catch (Exception e) {
    servletOut.print(e);
}
```

Q.56. Explain JDBC methods for accessing a database.

Ans. A connection object is used to obtain information about the database, like the tables it contains, as well as to set parameters of the interaction with the database, like whether or not changes to database will be committed automatically. For database access, however, the only method called on Connection is createStatement( ).This method returns an object of type Statement that can be used to send SQL statements to the database and to retrieve results produced by the statements. In particular, if a String representing a SQL statement is passed to the execute( ) method of a Statement object, then the database will execute that statement. The result of executing a SQL statement may be nothing, such as creation of a table, a row count such as performing an INSERT, UPDATE, or DELETE, or a result set such as performing a SELECT. The methods getUpdateCount( ) and getResultSet( ) are used to retrieve the row count and result set, respectively, after executing a SQL statement.

For example, suppose that a Connection object named con has been created, the following statements will create a table, insert a row in the table, and output to a message indicating that one row inserted –

```
Statement stmt = con.createStatement( );
stmt.execute("CREATE TABLE Interests" + "(Name VARCHAR (50),
                                    Interests VARCHAR(50)))";
stmt.execute("INSERT INTO Interests VALUES" + "('Amit', drawing,
                                    swimming, writing')");
System.out.println("Row count is" +stmt.getUpdateCount( ));
```

The getResultSet method returns an object of ResultSet, or null, if a result set was not produced by the most recent call to execute( ). The ResultSet object provides methods for positioning to a certain row within the result set and for obtaining data from the fields in a row. Continuing the example, the following code can be used to iterate through all of the rows in the EmployeeInterests table created by the preceding code and to output both fields of each row –

```
stmt.execute("SELECT*FROM EmployeeInterests");
ResultSet rs = stmt.getResultSet( );
if(rs!=null) {
    while (rs.next( ))
    {
        System.out.println(rs.getString("Name"));
        System.out.println(rs.getString("Interests"));
    }
}
```

The next( ) method positions the result set cursor at the next row each time it is called. Initially, the cursor is positioned just before the first row, then the first call to next( ) positions the cursor of the first row. The calls to getString( ) access fields in the row pointed to by the cursor. Fields can also be accessed by ordinal (i.e., 1, 2 etc.) rather than name.

Q.57. What is JNDI ? Explain.

Ans. Java Naming and Directory Interface (JNDI) is a unified Java API designed to standardized access to a variety of naming and directory services. This abstract mechanism is what makes J2EE an attractive enterprise architecture for Internet and intranet applications. Applications are written in a standard way to use the JNDI API, which transparently calls the underlying naming or directory service. A JNDI compliant service must implement part of JNDI API. Here JNDI architecture describe in two parts JNDI API and JNDI SPI. An application-level programming interface (API) are used by the application components to access naming and directory services. A service provider interface (SPI) is used to plug in a provider of a naming and directory service to the J2EE platfrom.

The JNDI architecture is shown in fig. 3.24.

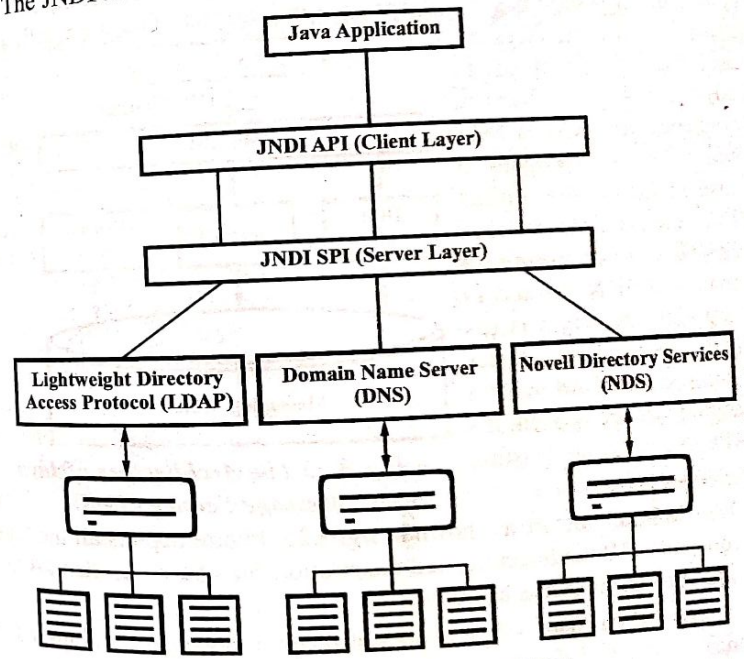


Fig. 3.24 Architecture of JNDI

LDAP (Lightweight directory access protocol) is the most popular directory protocol and is used as a standard network directory services.

DNS (Domain naming service) is used to refer to a host by its name instead of its numeric IP address.

NDS (Novell directory services) is used as a naming service to store user and group information for authentication purposes.

The JNDI model defines a hierarchical namespace in which you name objects. Each object in the namespace may have attributes that can be used to search for the object.

Naming and directory services are intimate partners. In fact most existing products provide both sets of functionality.

Naming services provide name-to-object mapping and directory services provide information about the object and tools for searching for them.

As a part of the common J2EE services JNDI enables seamless connectivity to heterogeneous enterprise naming and directory services. Developer can build powerful and portable directory-enabled applications using the JNDI standards.

Q.58. Explain in brief Java message service (JMS).

Ans. Java message service (JMS) is a common API (application level programming interface) and provider framework that enables the development of portable, message-based enterprise applications. The JMS API is an abstraction of the interfaces and classes that JMS clients use to handle messages when in communication with a JMS provider. This is analogous to the use of JDBC as a unified API to access data sources using JDBC to connect to a database, or JDNI to access naming and directory services using JNDI for naming services and components. JMS is not a messaging system by itself, it's an API to access an existing messaging system.

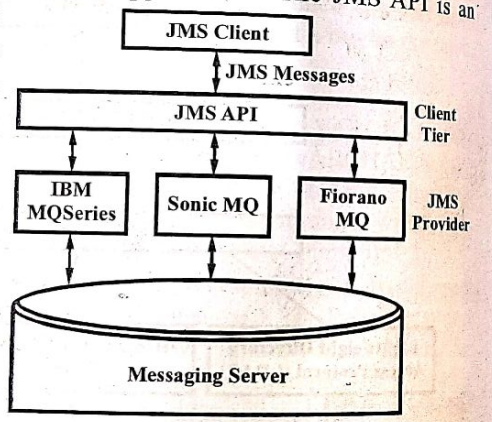


Fig. 3.25 The Architecture of Java Message Service (JMS)

The JMS architecture as shown in fig. 3.25. Figure depicts all the layers that constitute JMS architecture, and the relationships between them. A brief description of each layer is as follows –

(i) JMS Clients – It is send and receive messages through a JMS provider.

(ii) JMS Messages – Applications define a set of messages that are used to communicate information between its clients.

(iii) JMS API – Unified interfaces and classes to be used by all JMS clients.

(iv) JMS Provider – The messaging system (MOM) that implements JMS in addition to other administrative and control functionality required of a full-featured messaging product.

(v) Administered Objects – These are pre-configured JMS objects created by the JMS provider's administrator for the use of clients. Administered objects are not shown in fig. 3.25.

Java message service (JMS) specification defines these architecture components to facilitate writing portable enterprise applications. It does not address certain operational functionality such as clustering, security and administration, and error notification.

A number of JMS providers offer products of varying JMS support. Some of these products are SonicMQ from Progress, FioranoMQ from Fiorano, WebLogic from BEA, MQSeries from IBM and the open soruce JBossMQ from JBoss.

Q.59. Write short note on JRMI. ***(R.G.P.V., June 2013)***

Or

Define Remote Method Invocation (RMI). (R.G.P.V., Dec. 2010, 2015)

Ans. The Java programming language has built-in support for the distributed component-based computing model. This support is provided by the Remote Method Invocation (RMI) service. RMI is an alternative technology for CORBA, although functionally, it is much similar to CORBA. It is the Java standard for distributed components. RMI allows components written in Java to communicate with other Java components residing on other physical machines. For this purpose RMI provides a set of application programming interface (API) calls, as well as the basic infrastructure, similar to what CORBA's ORB provides.

CORBA uses IIOP as the protocol for communication between ORBs across a network, whereas the early versions of RMI used the Java Remote Method Protocol (JRMP), which performed the same task as IIOP. However, the latest versions of Java now support IIOP for RMI as well. That is, RMI can now have IIOP as the underlying protocol for communication between distributed components across a network.

Both, RMI and CORBA, are similar in terms concepts. RMI has two types of components – stubs and skeletons. A ***stub*** is a client-side component, which is a representation of the actual component on the server and executes on the client machine. In contrast, a ***skeleton*** is a server component. It has to deal with the fact that it is a remote component. This means that it has to take care of responding to other components that request for its services. This is the same interface-implementation concept. The beauty of this scheme is that a developer does not have to write the stub and skeleton interfaces. The Java environment generates it once the basic class is ready. For example, suppose there is a search class written in Java that allows the user to search for a specific record from a database. Then, the developer has to write the search class. A special compiler can generate the stub and skeleton interfaces for this class.

RMI is the Java version of Remote Procedure Calls (RPCs). The basic infrastructure of an RMI-based system looks similar to an RPC-based system, as shown in fig. 3.26.

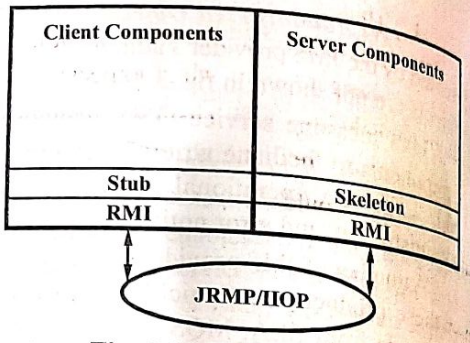


Fig. 3.26 RMI Architecture

The RMI philosophy is very similar to that of CORBA. Components call upon the RMI services first. The RMI services then use the JRMP/IIOP protocols for client-server communications. Whenever any component wants to use the services of another component, the caller must obtain a reference to the components to be used. The working of RMI is explained below with an example –

Suppose a client component wants to invokes a server-side component called as SearchBookServer that allows a book to be searched. This requires the following steps –

(i) The client has to create a variable of type *SearchBookServer*. This is similar to declaring an integer variable, when we want to use that integer in some calculations. It means that the client component has to declare a variable that can act as a reference to a component of type SearchBookServer. This would be done with this statement

SearchBookServer ref = null;

Here, we are declaring that the variable is not pointing to any object in memory by setting the variable to null. Consequently, at the client side, a variable called as ref is created. However, at this time, it is not serving any purpose. We have told the Java system that it can, in future, point to an object of type SearchBookServer. We would have an interface of the SearchBookServer class on the client. Hence, the compiler would have no problems in understanding that SearchBookServer is a class on the server, whose interface is available on the client as well.

(ii) RMI provides certain naming services to allow a client to query about a remote component. It is similar to a telephone directory service. Just as we can request for a person's telephone number based on the name and address, here, the naming service accepts the component's name along with its full URL and returns a reference to it. For this, the Naming.lookup method needs to be used. This method accepts the URL name and returns a reference to the object being remotely referred to. This is done by the following statement

ref = Naming.lookup ("rmi://www.myserver.com/SearchBookServer");

(iii) Having obtained a reference to the remote component, we can now call a remote method of that component. Suppose the component supports a method called as getAuthor, which expects a book title as the input and returns the author name to the caller. Then, this method can be invoked as follows –

uAuthor = ref.getAuthor("Freud for beginners");

This method would accept the string passed as book title, call the getAuthor method of the remote component and return the author's name. This returned value would be stored in the variable uAuthor, and can be sent back to the caller's computer using JRMP or IIOP, which, in turn, uses TCP/IP as a basic method of transport.

Obviously, from the above discussion, the RMI infrastructure is very similar to CORBA. In fact, an e-commerce architecture based on RMI would look very similar to CORBA based architecture.

The client would be a Web browser that supports Java. There would be two servers – a Web server and an application server. The interaction between the browser and the Web server would continue to be based on HTTP. This results into the downloading of HTML pages and applets from the Web server to the browser. As soon as the applets are downloaded to the browser, the applets would take charge and invoke the remote methods of the server-side components using RMI. The client applet can invoke the remote methods without worrying about their implementation details. All they need to do is to obtain a reference of the remote object and then they can invoke any methods belonging to that object as if it were a local method. This is shown in fig. 3.27.

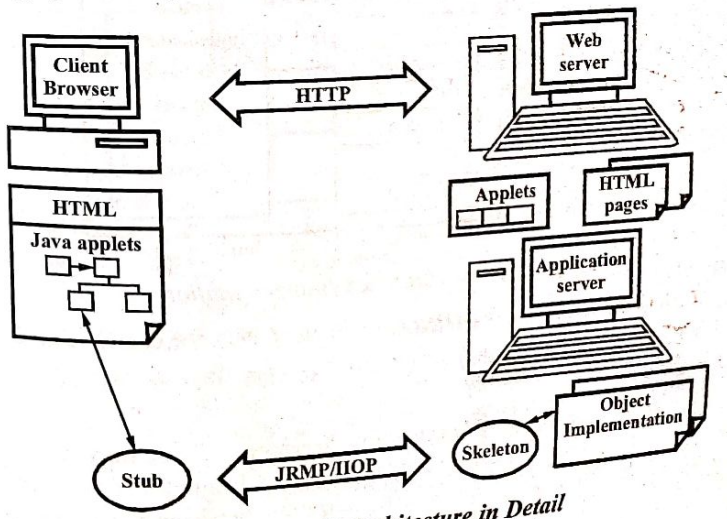


Fig. 3.27 RMI Architecture in Detail

Q.60. Explain the concept of CORBA. ***(R.G.P.V., June 2010)***

Or

Write short note on CORBA. ***(R.G.P.V., Dec, 2010)***

Ans. The Common Object Request Broker Architecture (CORBA) is a specification. It specifies how components can interact with each other over a network. CORBA allows developers to define distributed component architectures without considering about the underlying network communications and programming languages. The components in CORBA are declared in a language known as Interface Definition Language (IDL) provides the language independence. Developers can write the actual internal code of the components in a programming language like C++, Java, and COBOL.

The object-oriented principle of **interfaces** is used in order to achieve programming language independence. An interface is a set of functions or methods that signify what that interface can do i.e., the behaviour of the interface. It does not contain the implementation details i.e., how it is done. Let us understand this with the help of an example.

When we buy an audio system, we do not worry about the internal things such as the electronic components, the currents and voltages needed to work with them, etc. Instead, the manufacturer provides a set of interfaces to us. We can press a button to change the volume, to skip a track, and to eject a CD. Internally, that translates to different operations at the electronic component level. This set of internal operations is referred to as implementation. This is illustrated in fig. 3.28.

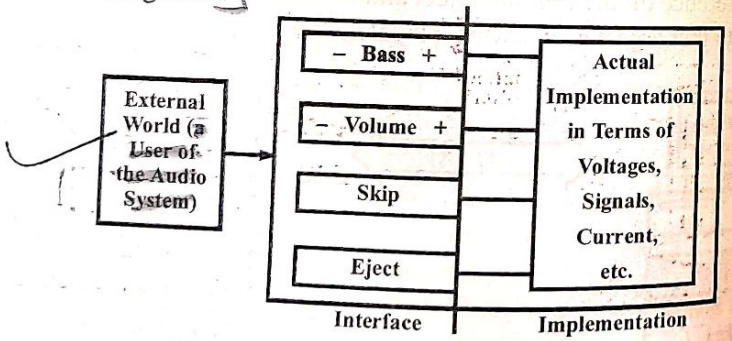


Fig. 3.28 Interface and Implementation

Q.61. What services of CORBA. ***(R.G.P.V. June 2011)***

Ans. There are a number of CORBA services that are categorized as follows –

(i) Information Management Services – This category include basic services for manipulation and retrieval of data. Examples of such services are properties, relationship, query, externalization, persistency and collection services.

(ii) Infrastructure Services – It includes service elements that are tightly coupled to the ORB, such as security, interoperatibility and messaging services.

(iii) Task Management Services – It includes services for managing distributed object events and transactions.

(iv) System Management Services – It includes basic services for enabling the management of metadata, licensing and object life cycle.

Q.62. Explain the architecture of CORBA in brief. ***(R.G.P.V., June 2014)***

Or

Explain CORBA architecture. ***(R.G.P.V., June 2015)***

Or

Explain CORBA architecture. Also discuss about its components. ***(R.G.P.V., June 2016)***

Ans. The architecture is designed to support the role of an object request broker that enables clients to invoke methods in remote objects, where both clients and servers can be implemented in a variety of programming languages. CORBA provides for both static and dynamic invocations. Static invocations are used when the remote interface of the CORBA object is known at compile time, enabling client stubs and server skeletons to be used. If the remote interface is not known at compile time, dynamic invocation must be used. Most programmers prefer to use static invocation because it provides a more natural programming model. Fig. 3.29 shows the main components of CORBA architecture which are as follows –

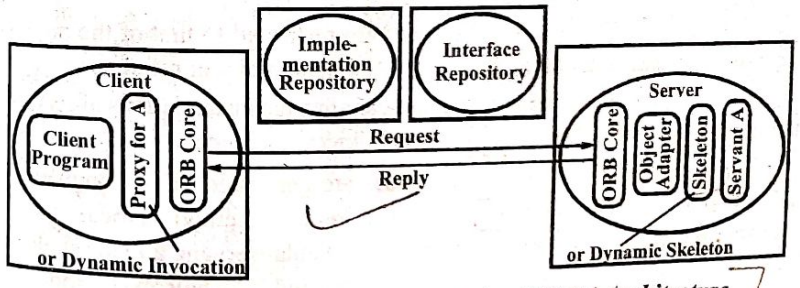


Fig. 3.29 The Main Components of the CORBA Architecture

(i) ORB Core – The role of ORB core is similar to the communication module of remote invocation. In addition, an ORB core provides an interface that includes the following –

(a) Operation enabling it to be started and stopped.

(b) Operations to convert between remote object references and strings.

(c) Operations to provide argument lists for requests using dynamic invocation.

(ii) Object Adapter – The role of an object adapter is to bridge the gap between CORBA objects with IDL interfaces and the programming language interfaces of the corresponding servant classes. An object adapter has the following tasks –

(a) It creates remote object references for CORBA objects.

(b) It activates and deactivates servants.

(c) It dispatches each RMI via a skeleton to the appropriate servant.

An object adapter gives each CORBA object a unique object name, which forms part of its remote object reference. The same name is used each time an object is activated. The object name may be specified by the application program or generated by the object adapter. Each CORBA object is registered with its object adapter, which may keep a remote object table that maps the name of CORBA objects to their servants. Each object adapter has its own name, which also forms part of the remote object references of all of the CORBA objects it manages. This name may either be specified by the application program or generated automatically.

(iii) Portable Object Adapter – The CORBA 2.2 standard for object adapters is called Portable Object Adapter (POA) because it allows applications and servants to be run on ORBs produced by different developers. This is achieved by means of the standardization of the skeleton classes and of the interactions between the POA and the servants. The POA supports CORBA objects with two different sorts of lifetimes –

(a) Those whose lifetimes are restricted to that of the process their servants and instantiated in. It has the transient object references.

(b) Those whose lifetimes can span the instantiations of servants in multiple processes. It has the persistent object references.

(iv) Skeletons – Skeleton classes are generated in the language of the server by an IDL compiler. As before, remote method invocations are dispatched via the appropriate skeleton to a particular servant, and the skeleton unmarshals the arguments in request messages and marshals exceptions and results in reply messages.

(v) Client Stubs/Proxies – These are in the client language. The class of a proxy or a set of stub procedures is generated from an IDL interface by an IDL compiler for the client language. As before, the client stubs/proxies marshal the arguments in invocation requests and unmarshal exceptions and results in replies.

(vi) Implementation Repository – An implementation repository is responsible for activating registered servers on demand and for locating servers

that are currently running. The object adapter name is used to refer to servers when registering and activating them. An implementation repository stores a mapping from the names of object adapters to the path names of files containing object implementations. Object implementations and object adapter names are generally registered with the implementation repository when server programs are installed. When object implementations are activated in servers, the hostname and port number of the server are added to the mapping.

Implementation repository entry –

Object Adapter Name	Pathname of Object Implementation	Host Name and Port Number of Server

(vii) Interface Repository – The role of interface repository is to provide information about registered IDL interfaces to clients and servers that require it. For an interface of a given type it can supply the names of the methods and for each method, the names and types of the arguments and exceptions. Thus, the interface repository adds a facility for reflection to CORBA. When an IDL compiler processes an interface, it assigns a type identifier to each IDL type it encounters. For each interface registered with it, the interface repository provides a mapping between the type identifier of that interface and the interface itself. Thus, the type identifier of an interface is sometimes called the repository ID because it may be used as a key to IDL interfaces registered in the interface repository. Every CORBA remote object reference includes a slot that contains the type identifier of its interface, enabling clients that hold it to enquire of its type with the interface repository. Those applications that use static invocation with client proxies and IDL skeletons do not require an interface repository. Not all ORBs provide an interface repository.

(viii) Dynamic Invocation Interface – The dynamic invocation interface allows clients to make dynamic invocations on remote CORBA objects. It is used when it is not practical to employ proxies. The client can obtain from the interface repository the necessary information about the methods available for a given CORBA object. The client may use this information to construct an invocation with suitable arguments and send it to the server.

(ix) Dynamic Skeletons – It may be necessary to add to a server a CORBA object whose interface was unknown when the server was compiled. If a server uses dynamic skeletons, then it can accept invocations on the interface of a CORBA object for which it has no skeleton. When a dynamic skeleton receives an invocation, it inspects the contents of the request to discover its target object, the method to be invoked and the arguments. It then invokes the target.

(x) Legacy Code – The term legacy code refers to existing code that was not designed with distributed objects in mind. A piece of legacy code may be made into a CORBA object by defining an IDL interface for it and providing an implementation of an appropriate object adapter and the necessary skeletons.

Q.63. Discuss Object Request Broker (ORB) in the CORBA architecture.

Ans. The Object Request Broker (ORB) is at the heart of distributed components architecture. ORB is the glue that holds the distributed objects together. ORB is actually a software program that runs on the client as well as the application server, where most of the components reside. It is responsible for making the communication between various distributed objects possible. ORB does two main tasks –

(i) The ORB locates a remote component, given its reference.

(ii) The ORB also makes sure that the called method receives the parameters over the network correctly. This process is known as ***marshaling***. In the reverse manner, the ORB also makes sure that the calling component receives back return values, if any, from the called method. This process is known as *unmarshaling*. Marshaling and unmarshaling actually carry out the conversions between the application data formats and the network data formats.

This process is shown in fig. 3.30. The client is not aware of these operations. Once a reference to a method of interest is obtained, the client believes that these operations were performed locally. Internally, ORB handles all the issues. For this reason, some portion of ORB is resident at all the clients and servers participating in this method. Whenever a client wants to execute a method of a component residing somewhere else, it only requests its local ORB portion (called ORB client). The ORB client makes connection to the appropriate server ORB, which in turn locates the component, executes that method and sends back the results to the client ORB. However, the client believes that it was all a local operation.

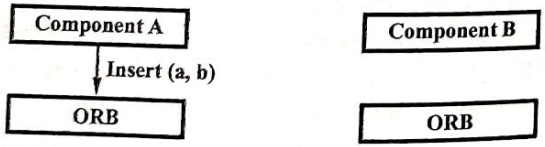


Fig. 3.30 Component Calls an Insert Method

(i) Component A calls the Insert method and passes two parameters a, and b to this method.

(ii) The ORB receives this request and realizes that the Insert method is defined in component B. How does it know this ? For this to be possible, whenever a CORBA component is created by a developer, it is registered with

the local operating system. Therefore, it passes the method name and the parameters in binary form (i.e., marshals it) across the network to the ORB at the node where component B is located. This is shown in fig. 3.31.

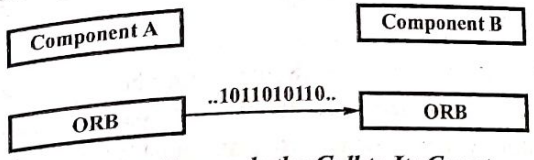


Fig. 3.31 ORB Forwards the Call to Its Counterpart

(iii) Now the ORB at component B's end receives this request and converts the binary data back into its original form (i.e., unmarshaling the request). It realizes that it needs to call the Insert method of component B with parameters as a and b. Therefore, it calls the Insert method, passing it the appropriate parameters. The Insert method is executed and its return value is returned to the ORB running on the same machine where the called component B resides. This is shown in fig. 3.32.

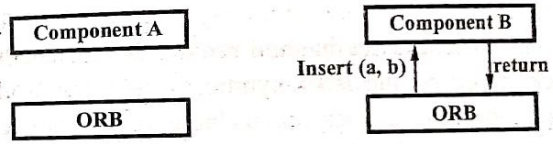


Fig. 3.32 Actual Insert Method Gets Called

(iv) The ORB at component B's end takes this return value, converts it into binary data and sends it across the network back to the ORB at component A's end, as shown in fig. 3.33.

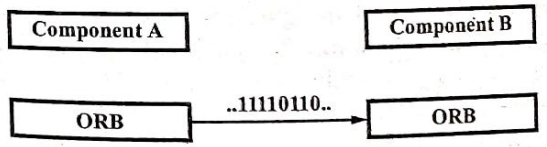


Fig. 3.33 Called ORB Returns Results to Calling ORB

(v) Finally, the ORB at component A's end receives this binary data, converts it back into the original form and gives it to component A, as shown in fig. 3.34.

Fig. 3.34 Calling ORB Returns Results to the Original Component

Q.64. Discuss Interface Definition Language (IDL) in CORBA architecture.

Ans. IDL specifies the interfaces between the various CORBA components. IDL ensures that CORBA components can interact with each other without regard to the programming language used.

When a component is interested in invoking a method of another component, the calling must know about the interface of the called component. As we know that IDL is used to describe the interfaces of CORBA components. Thus, it does not matter in which programming language the component is actually written in, it has to expose its interface through IDL. From the outside world's perspective, it is IDL interface that is seen. Internally, the component may be implemented in any language. Therefore, a CORBA component can expect every other CORBA component to expose its interface using IDL.

Consider, for an example, an interface called as StockServer, defined in IDL, that provides stock market information. The StockServer interface is expected to run at an application server, therefore, the naming conventions identify it as a server method to make it more readable. The interface contains two methods.

(i) The getStockPrice method returns the current stock price of a particular stock, based on the stock symbol it receives. It takes one input parameter as a string. It has no intention of changing this parameter, and hence, it is prefixed with the word 'in', which means it is input only. This method would return a floating-point value to the caller.

(ii) The getStockSymbolList method returns a list of all the stock symbols present in this stock exchange and does not expect any input parameters indicated by empty brackets after the method name. The return type is sequence <string>, which means a list of string values.

A portion of the IDL definition for this interface is as follows –

```
Interface StockServer
{
    float getStockPrice (in string symbol);
    sequence <string> getStockSymbolList( );
};
```

The actual code for this interface and the two methods that it contains can be written in any programming language such as C++, Java, etc. Any component calling the StockServer interface would not bother about its implementation, and therefore, its programming language. Consequently, the caller's implementation can be in say Java, whereas StockServer could be implemented in C++. This is fine, because both would have their respective external interfaces defined in IDL, which does not depend on either Java or C++.

Q.65. Discuss Internet Inter-ORB Protocol (IIOP) in CORBA architecture.

Ans. CORBA ORBs communicate with each other using the Internet Inter-ORB Protocol (IIOP). The early versions of CORBA were concerned only with creation of portable component based applications; that is, a component created on machine A could be ported to machine B and executed there. However, if the component was to remain on machine B where it is desired to be executed remotely from machine A, there was no standard way of communication between these various nodes. Obviously, HTTP had not been useful here. We needed a different protocol. Thus, the actual implementation of ORBs that facilitates the communication between these components was not a part of the standards in those days. Hence, although components were portable, they were not interoperable. It means that there was no standard mechanism for components to interact with each other.

For example, if a calling component A wanted to call a method named as *sum* belonging to another component B residing on a different computer, there was no guarantee that this would be possible. This happened because there was no standard mechanism for a component to remotely call a method of other component, if the two components were not on the same physical computer. This could lead to problems such as some protocols passed the parameters from left to right, others from right to left, some considered the sign bit as the first bit, other interpreted the sign bit as the last bit and so on. Therefore, remote distributed component-based computing was not standardized. Some vendors provided this feature with proprietary solutions. Consequently, the solution provided by one vendor was not compatible with another. Therefore, even if distributing components and then facilitating communication between them was possible with some proprietary solutions, this would not be compatible with another set of components that used a different vendor's solution. In essence, if the calling and the called components resided on the some machine, only then an interaction between them was guaranteed.

Therefore, the next version of CORBA came up with IIOP. IIOP is basically an additional layer to the underlying TCP/IP communication protocol. An ORB uses this additional CORBA messaging layer for communicating with other ORBs. Thus, every ORB must provide for IIOP stack, just like every browser and Web server on the Internet must provide for HTTP stack.

Because HTTP and IIOP both use the Internet infrastructure (i.e., TCP/IP) as the backbone, they can exist together on the same network. It means that the interaction between a client and the server can be through HTTP or IIOP. Hence, HTTP would be primarily used for downloading Web pages, applets and images from the Web server, whereas IIOP would be used for the

component-level communication between CORBA clients usually applets and CORBA servers usually component-based applications running on an application server. This situation is shown in fig. 3.35.

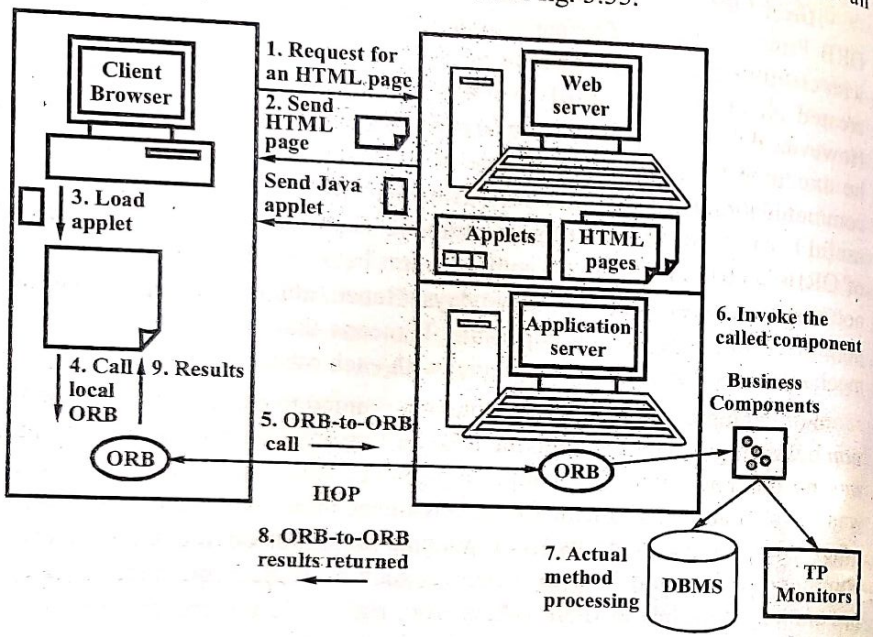


Fig. 3.35 Use of IIOP for OR-to-ORB Communication

In this figure we will realize that in steps 1 and 2, there is an interaction between the browser and Web server by using HTTP for requesting and obtaining HTML pages and Java applets. In step 3, the client invokes the Java applet, which in turn, invokes the services of one or more business components on the application server using the CORBA ORB. Note that it uses IIOP and not HTTP. The business components are shown to interact with databases and Transaction Processing monitors for server-side processing.

Q.66. When CORBA already exists, why is RMI required at all ?

Ans. The reasons for this are as follows –

(i) CORBA is a standard. Developers using Java or any other language can implement it. However, RMI is a part of the Java programming language itself. That is why, RMI is tightly integrated with Java only.

(ii) The aim of the Java creators was to have a full-fledged programming language that is platform independent. That is, they wanted to support maximum functionality that is required for all types of applications. Since remote method calls is an important issue nowadays, RMI was perceived as a necessity.

Q.67. Write a short note on UML. ***(R.G.P.V., June 2014)***

Ans. Unified Modeling Language (UML) is a language used to create an abstract system scenario, by visualizing, specifying, constructing, and documenting various parts and components of the system into a representative model enabling software system development. UML has its syntax and semantics. It provides a set of notations, such as rectangles, lines, ellipses, etc., to create models of systems that would be useful in documenting the design and analysis results.

The main goals in the design of UML are as follows –

(i) Offer users with a ready-to-use, expressive, and visual modeling language to create models.

(ii) Offer a language and notations to enhance concepts to higher order representation.

(iii) Do not depend on OO languages.

(iv) Assist higher-level development concepts such as component technology, rapid application development, reusability, interoperability, and portability.

The following benefits are provided by the UML-based modeling –

(i) Enhances communications among project teams.

(ii) Enhances the developer's insight and visualization of the complex system.

(iii) Developers learn faster to include the system's intricacies properly in the design.

(iv) Prototype design is more suitable, where the specific complexity of structure and behaviour is taken into account in each iteration. This enhances the system in increments, and part by part.

Q.68. What are the different system views that can be modelled using UML ? What are the different UML diagrams which can be used to capture each of the views ?

Or

Explain the use of UML for object-oriented design. ***(R.G.P.V., June 2010, 2011)***

Or

What are the different system views that can be modelled using UML? ***(R.G.P.V., June 2015)***

Ans. In UML, a system is represented using following views (models) to describe the system from distinctly different perspectives –

(i) User Model View – This view defines the functionalities provided by the system to its users.

(ii) Structural Model View – This view represents the structure of the problem in terms of the types of objects required to understand the working of a system and to its implementation.

(iii) Behavioural Model View – This view represents the interactions between various objects to realize the system behaviour.

(iv) Implementation Model View – This view represents the structural and behavioural aspects of the system because they are to be built.

(v) Environment Model View – This view represents the structural and behavioural aspects of the environment in which the system is to be implemented.

Fig. 3.36 shows the different UML diagrams which can be used to capture each of the views.

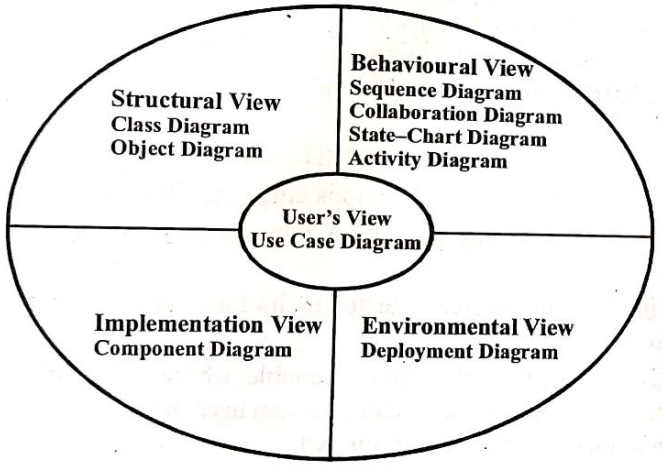


Fig. 3.36 Different Types of Diagrams and Views Supported in UML

⌘⌘

UNIT 4

SOFTWARE ARCHITECTURE ANALYSIS & DESIGN

SOFTWARE ARCHITECTURE ANALYSIS AND DESIGN – REQUIREMENTS FOR ARCHITECTURE AND THE LIFE-CYCLE VIEW OF ARCHITECTURE DESIGN AND ANALYSIS METHODS

Q.1. Describe software architecture analysis and design.

Ans. **Software Architecture Analysis** – The goal with software architecture analysis is to learn about the system to be built with the software architecture. This kind of analysis requires mappings between the software architecture and the system to be built. The accuracy of the results from such analyses are very dependent on how ambiguous these mappings are. The mappings, or semantics, of the elements of the software architecture descriptions are today very unclear.

The analysis of software architecture for the purpose of learning about the system that is going to be implemented would benefit from having a clear and universally defined semantics of a software architecture description technique.

Software architecture has much impact on the quality of a software system and it is important to be able to make informed decisions concerning the software architecture in a number of situations. Decision making regarding software architecture includes –

(i) Compare two alternatives relatively.

(ii) Compare the original and the modified software architecture relatively.

(iii) Compare one software architecture with the requirements.

(iv) Compare a software architecture to a theoretically viable software architecture, or

(v) Grading the software architecture on an interval or absolute scale.

An important source of information is the software architecture itself, and by analyzing the software architecture using different techniques we gather information that allows the stakeholders make more informed decisions about the situation.

The analysis take into account that when the software architecture is designed; detailed design is done on every module in the architecture and the

implementation. This is a source of variation in what could be expected from the implemented system. For example, a brilliant team of software engineers may still be able to do a good job with a poor software architecture. Or a perfect software architecture may lead to unacceptable results in the hand of a team of inexperienced software engineers that fails to understand the rationale behind the software architecture.

Software Architecture Design – A software architecture design method implies the definition of two things. First, a process or procedure for going about the included tasks. Second, a description of the results or type of results to be reached when employing the method. From the software architecture point-of-view, the first of the aforementioned two, includes the activities of specifying the components and their interfaces, the relationships between components, and making design decisions and document the results to be used in detail design and implementation. The second is concerned with the definition of the results, i.e. what is a component and how is it described etc.

Object-oriented methods describe an iterative design process to follow and their results. There is no guarantee that you will reach the desired results from following the prescribed process. The reason is that the processes prescribes no technique or activity for evaluation of the halting criterion for the iterative process, i.e. the software engineer is left for himself to decide when the design is finished. This is both from a method perspective and from a design perspective, insufficient since the stopping criterion relates to whether or not the requirements on the design result will be achieved.

Software architecture is the highest abstraction level at which we construct and design software systems. The software architecture sets the boundaries for the quality levels resulting systems can achieve. Consequently, software architecture represents an early opportunity to design for software quality requirements, e.g. reusability, performance, safety, and reliability.

The design method must in its process have an activity to determine if the design result, in this case the software architecture, has fulfilled the requirements. We only consider design methods with such an activity as considered complete.

Q.2. What is a software requirement ? What are its objectives ?

Ans. Requirement is a condition or capability possessed by a software or system component in order to solve a real world problem. The problems can be to automate a part of a system, to correct shortcomings of an existing system, to control a device, and so on. IEEE defines requirement as –

(i) A condition or capability needed by a user to solve a problem or achieve an objective.

(ii) A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed documents.

(iii) A documented representation of a condition or capability as in (i) or (ii).

Objectives – The objectives of software requirements are as follows –

(i) To introduce the concepts of user and system requirements.

(ii) To describe functional and nonfunctional requirements.

(iii) To explain two techniques for describing system requirements.

(iv) To explain how software requirements may be organized in a requirements document.

Q.3. Discuss the role of software requirements ?

Ans. Software requirements serve two major roles in a development effort. They specify what to develop and when the development is completed. A requirements document has all the software requirements of the system that is to be developed. It communicates the customer's needs, wishes, and expectations to the developers of the system. A requirements document may also have descriptions of the required computational functionality and the system behaviour, guidelines for the user interface, technical aspects of the hardware/software interface, and operational characteristics.

The second role of software requirements is to form the basis for determining when the software product is to be completed. The verification of the software functionality against the requirements document serves to demonstrate that the contractual agreement between the customer and the developers has been met and generally implies the end of the development phase. This verification may need the construction of test situations to objectively prove that the software products comply with the requirements document.

Q.4. What are functional and non-functional requirements ?
(R.G.P.V., June 2016)

Or

Compare the functional and non-functional requirements.
(R.G.P.V., Dec. 2010)

Ans. **Functional Requirements** – The functional requirements, also called ***behavioural requirements***, describe the functionality or services that software should provide. For this, functional requirements describe the interaction of software with its environment and specify the inputs, outputs, external interfaces, and the functions that should not be included in the software. Also, the services provided by functional requirements specify the procedure by which the software should react to particular inputs or behave in particular situations. IEEE defines function requirements as a function that a system or component must be able to perform.

Consider for example the functional requirements of an online banking system –

(i) The user of the bank should be able to search the desired services from the available ones.

(ii) There should be appropriate documents for users to read. This implies that when a user wants to open an account in the bank, the forms must be available so that the user can open an account.

(iii) After registration, the user should be provided with a unique acknowledgement number so that he can later be given an account number.

These requirements indicate user requirements and specify that functional requirements may be described at different levels of detail in an online banking system.

The functional requirements should be complete and consistent. Completeness implies that all the user requirements are defined. Consistency implies that all requirements are specified clearly without any contradictory definition. Generally, it is observed that completeness and consistency cannot be achieved in large software or in a complex system due to the errors that arise while defining the functional requirements of these systems.

Non-functional Requirements – The non-functional requirements, also called ***quality requirements***, relate to system attributes such as reliability and response time. Non-functional requirements arise due to user requirements, budget constraints, organizational policies etc. These requirements are not related directly to any particular function provided by the system.

Non-functional requirements should be accomplished in a software to make it perform efficiently. For example, if an aeroplane is unable to fulfil reliability requirements, it is not approved for safe operation.

Different types of non-functional requirements are shown in fig. 4.1.

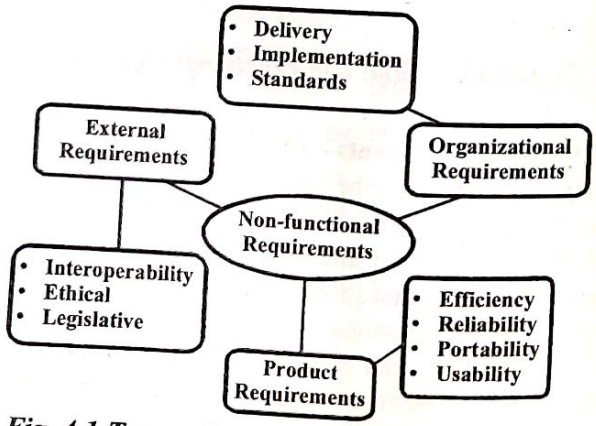


Fig. 4.1 Types of Non-functional Requirements

Q.5. Write short note on domain requirements.

Ans. Requirements which are derived from the application domain of a system instead from the needs of the users are called domain requirements. These requirements may be new functional requirements or specify a method to perform some particular computations. Moreover, these requirements include any constraint that may be present in the exiting functional requirements. It is important to understand these requirements because domain requirements reflect the fundamentals of the application domain. Also, if these requirements are not fulfilled, it may be difficult to make the system work as desired. A system can include a number of domain requirements.

Q.6. What is software development life cycle model ? Why is it important to adhere to a life cycle model while developing a large software product ? ***(R.G.P.V., June 2010)***

Or

Explain in detail about the life cycle process. ***(R.G.P.V., Dec. 2017)***

Ans. A life cycle model specifies the different activities that need to be performed to develop a software product and the sequencing of these activities. The software life cycle is also sometimes called as the systems development life cycle (SDLC). Basically, the classical waterfall model is the basic life cycle model. Every software product starts with a request for the product by the customer. This is called ***product conception***. Starting with this stage it undergoes transformations through a series of identifiable stages until it is fully developed and released to the customer. After release, the product is used by the customer and is finally retired when it is no longer useful. This forms the essence of the life cycle of every software product. Life cycle is not strange to software development. In fact, each business organizations conducts its business through a certain sequence of well-defined steps. Likewise, manufacturing industries use some steps to produce their product. The software life cycle can be viewed as the business process for software development, and therefore, a software life cycle is also often called as a ***software process***.

Traditionally, the feasibility study is the first stage in the life cycle of any software product. The subsequent stages include – requirement analysis and specification, design, coding, testing and maintenance. Each of these stages is referred to as a life cycle phase. During each life cycle phase, usually several kinds of activities need to be performed and several documents produced before the end of the phase. A software life cycle model is a diagrammatic and descriptive representation of the software life cycle. A life cycle model maps the various activities conducted on a software product from its inception to retirement into a set of life cycle phases. Different life cycle models may map the fundamental development activities to phases in different ways. Thus, no

matter which life cycle model is used, the basic activities are included in all life cycle models, though the activities may be performed in different order in various life cycle models.

Software development organizations have felt that adherence to an appropriate well-defined life cycle model helps to produce good quality products and that too without time and cost overruns. The main benefit of adhering to a life cycle model is that it enables development of software in a systematic and disciplined way.

Q.7. Discuss the life cycle view of architecture design and analysis methods.

Ans. Many architecture-centric analysis and design methods have been created in the past ten years, beginning with the software architecture analysis method (SAAM), which inspired the creation of other methods. The first such method that we created at the software engineering institute (SEI) was the architecture tradeoff analysis method.

As we gained experience from the ATAM, we expanded our repertoire into more phases of the life cycle with the following methods –

(i) Quality attribute workshop (QAW)

(ii) Cost-benefit analysis method (CBAM)

(iii) Active reviews for intermediate designs (ARID)

(iv) Attribute-driven design (ADD) method.

We examine these methods and their relationship to the software development life cycle (SDLC).

These methods share not only a common heritage, but also a common set of characteristics, aside from being architecture-centric. First, they all are scenario driven, with the scenarios serving as the "engine" for directing and focusing the methods' activities. Second, they all are directed by operationalized quality attribute models. The SAAM focused on modifiability. The ATAM looks at tradeoffs among multiple quality attributes, while the ADD method shapes design decisions around quality attribute considerations. The QAW attempts to elicit and document quality attribute requirements accurately, particularly in the absence of explicit architectural documentation. Third, the methods all focus on documenting the rationale behind the decisions made; in this way, the rationale serves as a knowledge base on which to base both existing and future decisions. Last, they all involve stakeholders so that multiple views of quality are elicited, prioritized, and embodied in the architecture.

A typical SDLC, as practiced in relatively mature software development organizations, includes the following activities –

(i) Understanding of business needs and constraints

(ii) Elicitation and collection of requirements

(iii) Architecture design

(iv) Detailed design

(v) Implementation

(vi) Testing

(vii) Deployment

(viii) Maintenance.

As architecture-centric methods become more widespread, more widely adopted, and integrated into an SDLC, organizations inevitably will want to tailor them. Consequently, organizations that wish to include the eliciting and gathering of quality-attribute-based requirements, explicit architecture design, and architecture analysis in their life cycles will be best served if they can do so "organically". The steps and artifacts of the five architecture-centric methods are – QAW, ADD, ATAM, CBAM and ARID – therefore may need to be tailored, blended, and, in some cases, removed entirely when the activities of these methods are integrated into an organization's existing life cycle.

ARCHITECTURE BASED ECONOMIC ANALYSIS – COST BENEFIT ANALYSIS METHOD (CBAM), ARCHITECTURE TRADEOFF ANALYSIS METHOD (ATAM), ACTIVE REVIEWS FOR INTERMEDIATE DESIGN (ARID), ATTRIBUTE DRIVEN DESIGN METHOD (ADD)

Q.8. Write short note on architecture based economic analysis.

Ans. Architecture based economic analysis is grounded on understanding the utility-response curve of various scenarios and casting them into a form that makes them comparable. Once they are in this common form – based on the common coin of utility – the value of cost (VFC) for each architecture improvement, with respect to each relevant scenario, can be calculated and compared.

Applying the theory in practice has a number of practical difficulties, but in spite of those difficulties, we believe that the application of economic techniques is inherently better than the ad hoc decision-making approaches that projects (even quite sophisticated ones) employ today. Our experience with the cost benefit analysis method (CBAM) tells us that giving people the appropriate tools to frame and structure their discussions and decision making is an enormous benefit to the disciplined development of a complex software system.

Q.9. Explain the cost benefit analysis method (CBAM).

Ans. The cost benefit analysis method facilitates architecture based economic analyses of software intensive system. CBAM has for the most part been applied when an organization was considering a major upgrade to an existing system and they wanted to understand the utility and value for cost of making the upgrade, or they wanted to choose between competing architectural strategies for the upgrade. CBAM is also applicable for new systems as well, especially for helping to choose among competing strategies. Its key concepts (quality attribute response curves, cost, and utility) do not depend on the setting.

Inputs – The inputs of CBAM are as follows –

(i) The system's business/mission drivers

(ii) A list of scenarios

(iii) The existing architectural documentation.

Steps of the CBAM – A process flow diagram for the CBAM is given in fig. 4.2.

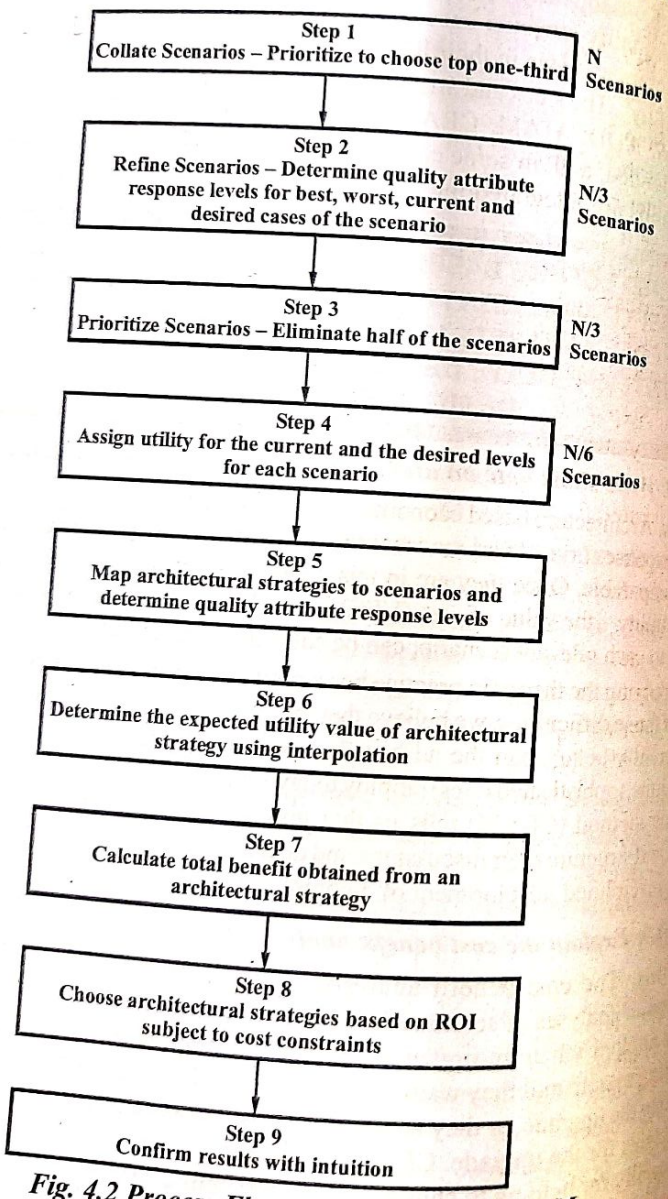


Fig. 4.2 Process Flow Diagram for the CBAM

This method includes the following steps –

Step 1. Collate Scenarios – Collate the scenarios elicited during the ATAM exercise and give the stakeholders the chance to contribute new ones. Prioritize these scenarios based on satisfying the business goals of the system and choose the top one-third for further study.

Step 2. Refine Scenarios – Refine the scenarios, focusing on their stimulus/response measures. Elicit the worst, current, desired, and best-case quality-attribute-response level for each scenario.

Step 3. Prioritize Scenarios – Allocate 100 votes to each stakeholder to be distributed among the scenarios, where the stakeholder's voting is based on considering the desired response value for each scenario. Total the votes and choose the top 50% of the scenarios for further analysis. Assign a weight of 1.0 to the highest rated scenario. Relative to that scenario, assign the other scenarios a weight that becomes the number used in calculating the architectural strategy's overall benefit. Make a list of the quality attributes that concern the stakeholders.

Step 4. Assign Intra-scenario Utility – Determine the utility for each quality-attribute-response level (worst-case, current, desired, best-case) for the scenarios under study. The quality attributes of concern are the ones in the list generated during Step 3.

Step 5. Develop Architectural Strategies for Scenarios and Determine their Expected Quality-attribute-response Levels – Develop (or capture already developed) architectural strategies that address the chosen scenarios and determine the expected quality-attribute-response levels that will result from implementing these architectural strategies. Given that an architectural strategy may affect multiple scenarios, this calculation must be performed for each affected scenario.

Step 6. Determine the Utility of the Expected Quality-attribute-response Levels by Interpolation – Using the elicited utility values (that form a utility curve), determine the utility of the expected quality-attribute-response level for the architectural strategy. Determine this utility for each relevant quality attribute enumerated in the previous step.

Step 7. Calculate the Total Benefit Obtained from an Architectural Strategy – Subtract the utility value of the current level from the expected level and normalize it using the votes elicited previously. Sum the benefit of a particular architectural strategy across all scenarios and relevant quality attributes.

Step 8. Choose Architectural Strategies based on Return on Investment (ROI) Subject to Cost and Schedule Constraints – Determine

the cost and schedule implications of each architectural strategy. Calculate the ROI value for each remaining strategy as a ratio of benefit to cost. Rank the architectural strategies according to the ROI value and choose the top ones until the budget or schedule is exhausted.

Step 9. Confirm Results with Intuition – Of the chosen architectural strategies, consider whether they seem to align with the organization's business goals. If not, consider issues that may have been overlooked while doing this analysis. If significant issues exist, perform another iteration of these steps.

Outputs – Outputs of the CBAM are as follows –

(i) A set of architectural strategies, with associated costs, benefits, and schedule implications

(ii) Prioritized architectural strategies, based on ROI

(iii) The risk of each architectural strategy, quantified as variability in cost, benefit, and ROI values.

Q.10. What are the benefits of CBAM ?

Ans. The benefits that an architectural decision may bring to an organization are as important – or perhaps more important – than the costs.

The CBAM enables you to make informed decisions about software requirements and software investments based on an analysis of the economic and architectural implications of those decisions.

Q.11. Explain in detail about the architecture tradeoff analysis method (ATAM).

Ans. The architecture tradeoff analysis method (ATAM) is a spiral model of architecture design, it is iterative in its nature and its each iteration is used to reduce risk that could result from competing quality attributes that a software inherits. ATAM has been used for over a decade to evaluate software architecture in domains ranging from automotive to financial to defense. This method is designed so that evaluators need not be familiar with the architecture or its business goals, the system need not yet be constructed, and there may be large number of stakeholders.

Participants in the ATAM – The ATAM requires the participation and mutual cooperation of three groups –

(i) The Evaluation Team – This group is external to the project whose architecture is being evaluated.

(ii) Project Decision Makers – These people are empowered to speak for the development project or have the authority to mandate changes to it.

(iii) Architecture Stakeholders – Stakeholders have a vested interest in the architecture performing as advertised. They are the ones whose ability

to do their job hinges on the architecture promoting modifiability, security, high reliability or the like. Stakeholders include developers, testers, integrators maintainers, users, builders of systems interacting with the one under consideration.

Inputs to the ATAM – Inputs include the

(i) System's business/mission drivers

(ii) Existing architectural documentation.

Steps of the ATAM – This method includes the following steps –

Step 1. Present Business Drivers – A project spokesperson (ideally the project manager or system customer) describes which business goals are motivating the development effort and identifies the primary architectural drivers (e.g., high availability, time to market, or high security).

Step 2. Present Architecture – The architect describes the architecture, focusing on how it addresses the business drivers.

Step 3. Identify Architectural Approaches – The architect identifies, but does not analyze, architectural approaches.

Step 4. Generate Quality Attribute Utility Tree – The quality factors that make up system "utility" (performance, availability, security, modifiability, etc.) are specified down to the level of scenarios, annotated with stimuli and responses, and prioritized.

Step 5. Analyze Architectural Approaches – Based on the high-priority factors identified in the utility tree, the architectural approaches that address those factors are elicited and analyzed (e.g., an architectural approach aimed at meeting performance goals will be subjected to a performance analysis). Architectural risks, sensitivity points, and tradeoff points are identified.

Step 6. Brainstorm and Prioritize Scenarios – A larger set of scenarios is elicited from stakeholders and prioritized through a voting process.

Step 7. Analyze Architectural Approaches – The highest ranked scenario are treated as test cases – they are mapped to the architectural approaches previously identified. Additional approaches, risks, sensitivity points, and tradeoff points may be identified.

Outputs of the ATAM – Outputs include –

(i) List of architectural approaches

(ii) List of scenarios

(iii) Set of attribute-specific questions

(iv) Utility tree

(v) List of risks

(vi) List of non-risks

(vii) List of risk themes

(viii) List of sensitivity points

(ix) List of tradeoffs.

Q.12. Why we use architecture tradeoff analysis method (ATAM) ?

Ans. All design, in any discipline, involves tradeoffs; this is well accepted. What is less well understood is the means for making informed, and possibly even optimal tradeoffs. Design decisions are often made for non-technical reasons – strategic business concerns, meeting the constraints of cost and schedule, using available personnel, and so forth.

Having a structured method helps ensure that the right questions will be asked early, during the requirements and design stages when discovered problems can be solved cheaply. It guides users of the method – the stakeholders – to look for conflicts in the requirements and for resolutions to these conflicts in the software architecture.

In realizing the method, we assume that attribute-specific analyses are interdependent, and that each quality attribute has connections with other attributes, through specific architectural elements. An architectural element is a component, a property of the component, or a property of the relationship between components that affects some quality attribute. For example, the priority of a process is an architectural element that could affect performance. The ATAM helps to identify these dependencies among attributes; what we call tradeoff points. This is the principal difference between the ATAM and other software analysis techniques – that it explicitly considers the connections between multiple attributes, and permits principled reasoning about the tradeoffs that inevitably result from such connections. Other analysis frameworks, if they consider connections at all, do so only in an informal fashion, or at a high level of abstraction tradeoff points arise from architectural elements that are affected by multiple attributes.

Q.13. What do you understand by active reviews for intermediate design (ARID) ?

Ans. The active reviews for intermediate designs (ARID) method blends Active Design Reviews with the ATAM, creating a technique for investigating designs that are partially complete. Like the ATAM, the ARID method engages the stakeholders to create a set of scenarios that are used to "test" the design for usability – that is, to determine whether the design can be used by the software engineers who must work with it. The ARID method helps to find issues and problems that hinder the successful use of the design as currently conceived.

Inputs to ARID – Inputs include –

(i) A list of seed scenarios

(ii) The existing architectural/design documentation.

Steps of ARID – This method includes the following steps –

Step 1. Present the Design – The lead designer presents an overview of the design and walks through the examples. During this time, participants follow the ground rule that no questions concerning implementation or rationale are allowed, nor are suggestions about alternate designs. The goal is to see if the design is "usable" to the developer, not to find out why things were done a certain way or to learn about the secrets behind implementing the interfaces. This step results in a summarized list of potential issues that the designer should address before the design can be considered complete and ready for production.

Step 2. Brainstorm and Prioritize Scenarios – Participants suggest scenarios for using the design to solve problems they expect to face. After they gather a rich set of scenarios, they winnow them and then vote on individual scenarios. By thier votes, the reviewers actually define a usable design – if the design performs well under the adopted scenarios, they must agree that it has passed the review.

Step 3. Apply the Scenarios – Beginning with the scenario that received the most votes, the facilitator asks the reviewers to craft code (or pseudo-code) jointly that uses the design services to solve the problem posed by the scenario. This step is repeated until all scenarios are covered or the time allotted for the review has ended.

Output of ARID – The output includes a list of "issues and problems" preventing successful use of the design.

Q.14. What are the benefits of ARID ?

Ans. ARID helps architecture designers engage stakeholders and get their buy-in early in the design process. It also informs designers about whether their design is suitable for the overall system being developed.

Reviewing a design in its pre-release stage provides valuable early insight into the design's viability and allows for timely discovery of errros, inconsistencies, and inadequacies.

Q.15. Comparision of ATAM and ARID.

Ans. Comparision of ATAM and ARID is shown in table 4.1.

Table 4.1

S.No.	*Description*	*ATAM*	*ARID*
(i)	Artifact examined	Architecture	Conceptual design approach, with embryonic documentation.
(ii)	Who participates	Architect, stakeholders	Lead designer, stakeholders.
(iii)	Basic approach	Elicit drivers and qualities, build utility tree, catalog approaches, perform approach-based analysis.	Present design, elicit desired uses, have reviewers as a group use the design.

(iv)	Outputs	Identified risks, sensitivity points, and trade-off points.	Issues and problems preventing successful usage.
(v)	Approximate duration	3 full days of meetings, over approx. 2 weeks, plus unstructured work and communication between the meetings.	1 day pre-meeting plus 1 day review meeting.

Q.16. What do you mean by attribute driven design method (ADD) ? Explain.

Ans. The attribute driven design method (ADD) is an application of the generate and test philosophy. It keeps the number of requirements that must be satisfied to a humanly achievable quantity. ADD is an iterative method that, at each iteration, helps the architect to do the following –

(i) Select an element of the system to design.

(ii) Marshal all the architecturally significant requirements for the selected element.

(iii) Create and test a design for that selected element.

The output of ADD is not an architecture complete in every detail, but an architecture in which the main design approaches have been selected and validated. It produces a "workable" architecture early and quickly, one that can be given to other project teams so they can begin their work while the architect or architecture team continues to elaborate and refine.

ADD is a five-step method are as follows –

Step 1. Select the Element of the System to Design – For green-field designs, the "part" to begin with is simply the entire system. For designs that are already partially completed (either by external constraints or by previous iterations through ADD), the part is an element that is not yet designed. Selecting the next element can proceed in a breadth-first, depth-first, or mixed manner.

Step 2. Identify the ASRs for the selected element.

Step 3. Generate a Design Solution for the Selected Element – Using design collateral such as existing systems, frameworks, patterns and tactics, and the design checklists.

Step 4. Verify and Refine Requirements and Generate Input for the Next Iteration – Either the design in step 3 will satisfy all of the selected element's ASRs or it won't. If it does not, then either they can be allocated to elements that will be elaborated in future iterations of ADD, or the existing design is inadequate and we must backtrack. Furthermore, non-ASR requirements will either be satisfied, allocated to children, or indicated as not achievable.

Step 5. Repeat Steps 1-4 – Until all the ASRs have been satisfied or until the architecture has been elaborated sufficiently for the implementers to use it.

Q.17. Briefly discuss how ADD uses three commons views.

Ans. The ADD uses these three common views are as follows –

(i) Module Decomposition View – Our discussion above shows how the module decomposition view provides containers for holding responsibilities as they are discovered. Major data flow relationships among the modules are also identified through this view.

(ii) Concurrency View – In the concurrency view dynamic aspects of a system such as parallel activities and synchronization can be modeled. This modeling helps to identify resource contention problems, possible deadlock situations, data consistency issues, and so forth. Modeling the concurrency in a system likely leads to discovery of new responsibilities of the modules, which are recorded in the module view. It can also lead to discovery of new modules, such as a resource manager, in order to solve issues of concurrent access to a scarce resource and the like.

To understand the concurrency in a system, the following use cases are illuminating –

(a) Two Users doing Similar Things at the Same Time – This helps in recognizing resource contention or data integrity problems. In our garage door example, one user may be closing the door remotely while another is opening the door from a switch.

(b) One User Performing Multiple Activities Simultaneously – This helps to uncover data exchange and activity control problems. In our example, a user may be performing diagnostics while simultaneously opening the door.

(c) Starting Up the System – This gives a good overview of permanent running activities in the system and how to initialize them. It also helps in deciding on an initialization strategy, such as everything in parallel or everything in sequence or any other model. In our example, does the startup of the garage door opener system depend on the availability of the home information system ? Is the garage door opener system always working, waiting for a signal, or is it started and stopped with every door opening and closing ?

(d) Shutting Down the System – This helps to uncover issues of cleaning up, such as achieving and saving a consistent system state.

(iii) Deployment View – If multiple processors or specialized hardware is used in a system, additional responsibilities may arise from deployment to the hardware. Using a deployment view helps to determine and design a deployment that supports achieving the desired qualities. The deployment view results in the virtual threads of the concurrency view being decomposed into virtual threads within a particular processor and messages that travel between processors to initiate the next entry in the sequence of actions. Thus, it is the basis for analyzing the network traffic and for determining potential congestion.

ARCHITECTURE REUSE, DOMAIN-SPECIFIC SOFTWARE ARCHITECTURE

Q.18. What do you mean by software reuse ?

Ans. By software reuse, we mean the repeated use of any part of a software system – documentation, code, design, requirements, test cases, test data, and more. Basili encourages us to think of maintenance as reuse. We take an existing system and reuse parts of it to build the next version. Similarly, he suggests that processes and experience can be reused, as can any tangible or intangible product of development.

Q.19. Discuss the advantages and disadvantages of reused code in software development. ***(R.G.P.V., June 2005, 2008)***

Ans. **Advantages of Reused Code –**

(i) Increased Reliability – Reused components that have been exercised in working systems should be more reliable than new components. They have been tried and tested in a variety of different environments. Design and implementation faults are discovered and eliminated in the initial use of the components, thus reducing the number of failures when the component is reused.

(ii) Reduced Process Risk – If a component exists, there is less uncertainty in the costs of reusing that component than in the costs of development. This is an important factor for project management as it reduces the uncertainties in project cost estimation. This is particularly true when relatively large components like sub-systems are reused.

(iii) Standards Compliance – Some standards, such as user interface standards, can be implemented as a set of standard components. For example, reusable components may be developed to implement menus in a user interface. All applications present the same menu formats to users. The use of standard user interface improves reliability as users are less likely to make mistakes when presented with a familiar interface.

(iv) Effective Use of Specialists – Instead of application specialists doing the same work on different projects, these specialists can develop reusable components which encapsulate their knowledge.

(v) Accelerated Development – Bringing a system to market as early as possible is often more important than overall development costs. Reusing components speeds up system production because both development and validation time should be reduced.

Disadvantages of Reused Code –

(i) Increased Maintenance Costs – If component source code is not available then maintenance costs may be increased as the reused elements of the system may become increasingly incompatible with system changes.

(ii) Maintaining a Component Library – Populating a component library and ensuring that software developers can use this library can be expensive. Our current techniques for classifying, cataloguing and retrieving software components are immature.

(iii) Lack of Tool Support – CASE toolsets do not support development with reuse. It may be difficult or impossible to integrate these tools with a component library system. The software process assumed by these tools may not take reuse into account.

(iv) Finding and Adapting Reusable Components – Software components have to be discovered in a library, understood and sometimes adapted to work in a new environment. Engineers must be reasonably confident of finding a component in the library before they will routinely include a component search as part of their normal development process.

(v) Non-invented-here Syndrome – Some software engineers sometimes prefer to rewrite components as they believe that they can improve on the reusable components. This is partly to do with trust and partly to do with the fact that writing original software is seen as more challenging than reusing other people's software.

Q.20. Write short note on domain-specific software architecture.

Ans. A domain-specific software architecture (DSSA) has been defined as –

"An assemblage of software components, specialized for a particular type of task (domain), generalized for effective use across that domain, composed in a standardized structure (topology) effective for building successful applications" or, alternately.

"A context for patterns of problem elements, solution elements, and situations that define mappings between them.

Q.21. Explain domain-specific software architecture with construction method.

Ans. Domain-specific software architecture consists of a domain model, reference requirements and referential architecture used to support a group of applications in a specific problem domain, and its aim is to support the application in one specific domain.

Function – Software development in certain domains is more feasible. System development is based on indirectly mapping a problem space to a solutions space in software architecture through description of abstract layers of the system structure. However, it is difficult to implement due to the complexity of the problem space, abstraction and implementation. On the contrary, the development method of the domain-specific software architecture first divides the problem space into different domains and then realizes the solving scheme in this domain. Practice has proved this kind of software development more feasible.

Software reuse is easier in specific domains. Through deep analysis of specific domains and abstraction of the common features and dynamic behaviour of the application systems in the domain, it is possible to apply the architecture to other application system development in the same domain, to reuse the software architecture, and to reduce the complexity of architecture.

Construction Method – Domain-specific software architecture is constructed after domain analysis and on the basis of the domain model. To identify the domain model, we determined the scope of domain with the help of an expert in the domain development environment. Then we designed an integration scheme according to the identified domain model component. Finally, we constructed the domain architecture and a mapping domain model for a realizable information framework. The model is shown in fig. 4.3.

Because a single E-learning system can hardly satisfy individualized application in the teaching process and because it is difficult to integrate many systems, we put forward construction of domain-specific software architecture oriented to E-learning. Within the restrictions of the architecture, we encapsulated the public business logic in the domain into the software architecture and stipulated the standard component interface. Through the plug-in and dynamic binding of the components, we built an individualized E-learning system. In the meantime, the standardized and opened interface made reuse of the components and the system integration possible and built a flexible E-learning system to support individualized teaching.

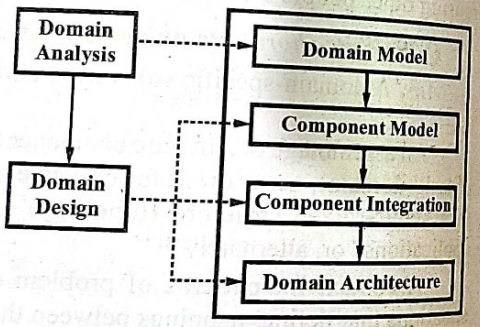


Fig. 4.3 Construction Model of Domain-specific Software Architecture

⌘⌘

UNIT 5

SOFTWARE ARCHITECTURE DOCUMENTATION

SOFTWARE ARCHITECTURE DOCUMENTATION, PRINCIPLES OF SOUND DOCUMENTATION, REFINEMENT, CONTEXT DIAGRAMS, VARIABILITY, SOFTWARE INTERFACES

Q.1. What do you understand by software architecture documentation?

Ans. Architectural documentation describes the structure of a system through one or more views, each of which identifies a collection of high-level components and relations among those components. A component is usually documented visually as some sort of geometrical object, and represents a coherent unit of functionality. The granularity of components will depend on the kind of documentation being developed in some situations a component may be as large as a major subsystem, in others it might be as small as a single object class. Typically components represent system structures such as major modules, computational elements, and run-time processes.

The relations between components are documented visually using lines or adjacency. Typically such relations indicate what aspects of one component are used by other components, and how inter-component communication proceeds over time.

Different views are used to represent distinct aspects of a system, each view providing a model of some aspect of the system. For example, as we will see, one architectural view might document the structure of a system as a layered description in which the components represent logical groupings of code, while another view might document the structure of a system in terms of its run-time configuration in which components represent communicating processes.

Different views, or models, are useful for different purposes. Deciding which views to use is one of the chief jobs of a software architect. Often the choice of views will depend strongly on the needs for design analysis. Despite this variability there are typically at least four classes of views that are required to provide a reasonable set of architectural documents.

(i) Context-based Views – These indicate the setting in which the system is to be employed, and often identify the abstract domain elements that determine the system's overall requirements and business context.

(ii) Code-based Views – These describe the structure of the code, indicating how the system is built out of implementation artifacts, such as modules, tables, classes, etc. Such views are particularly useful as a guide to implementation and maintenance. They can also be used to indicate boundaries of abstraction between different parts of the system, and between the system-under-construction other parts of the system that it uses or that use it. A special, but common, case of a code-based view is a layered diagram. By partitioning a system into layers, one can improve portability, modifiability, and ease of use via standard APIs.

(iii) Run-time Views – These describe the structure of the system in operation, indicating what are the main run-time entities and how they communicate between each other. Run-time views allow one to reason about behavioural properties and "quality attributes" such as run-time resource consumption, performance, throughput, latencies, reliability, etc.

(iv) Hardware-based Views – These describe the physical setting in which the system is to run, indicating the number and kinds of processors and communication links. The information contained in these views is often combined with that in run-time views to derive system performance properties.

Q.2. Describe the uses of architectural documentation.

Ans. The uses of architectural documentation is given in table 5.1.

Table 5.1 Uses of Architectural Documentation

S.No.	*Stakeholder*	*Use*
(i)	Architect and requirements engineers who represent customer(s)	To negotiate and make tradeoffs among competing requirements.
(ii)	Architect and designers of constituent parts	To resolve resource contention and establish performance and other kinds of runtime resource consumption budgets.
(iii)	Implementors	To provide inviolable constraints (plus exploitable freedoms) on downstream development activities.
(iv)	Testers and integrators	To specify the correct black-box behaviour of the pieces that must fit together.

(v)	Maintainers	To reveal areas a prospective change will affect.
(vi)	Designers of other systems with which this one must interoperate	To define the set of operations provided and required, and the protocols for their operation
(vii)	Quality attribute specialists	To provide the model that drives analytical tools such as rate-monotonic real-time schedulability analysis, simulations and simulation generators, theorem provers, verifiers, etc. These tools require information about resource consumption, scheduling policies, dependencies, and so forth. Architecture documentation must contain the information necessary to evaluate a variety of quality attributes such as security, performance, usability, availability, and modifiability. Analyses for each attributes have their own information needs.
(viii)	Managers	To create development teams corresponding to work assignments identified, to plan and allocate project resources, and to track progress by the various teams.

Q.3. What are the principles of sound documentation ?

Ans. The seven principles of sound documentation are as follows –

(i) Write from the Reader's Viewpoint – A document is read only if it meets the needs of, and is usable by, its intended audience. Material written in streams of consciousness or using arcane terminology is unlikely to meet the reader's needs and thus is unlikely to be read or consulted often.

(ii) Avoid Unnecessary Repetition – While repetition sometimes reinforces a point, its use in technical information becomes troublesome over time. Repetition is the root of inconsistency. Keeping track of all repeats is difficult, if not impossible; thus, repeated information becomes inconsistent over time, and attempts to avoid these inconsistencies are costly.

(iii) Avoid Ambiguity – This principle might better be stated as "avoid unintended ambiguity" because software architecture, by its nature, is

ambiguous in areas that remain undecided until the system is implemented. Nevertheless, if a decision is made, the documentation must communicate it unambiguously so that system stakeholders do not misinterpret it. Such misinterpretation can lead to confusion, incorrect implementation, or problems during system verification and validation.

(iv) Use a Standard Organization – Usually a document is not read more than once, if that. Yet, if it is successful, readers will refer to it numerous times. Providing a standard organization not only helps a reader quickly find information, but also provides the architect with guidance on what needs to be captured and what has or has not been captured at any given time.

(v) Record Rationale – The reasoning behind the decisions is just as important as the decisions themselves. Architecture documentation lives with the system and, as most developers have experienced, the reasoning behind a decision may be forgotten in as little as a few weeks. Understanding the rationale behind decisions helps the architect refrain from revisiting decisions, helps designers understand why specific choices were made, and supports system evolution by stating explicitly that certain decisions were based on the context and technological constraints imposed at the time.

(vi) Keep Documentation Current but Not too Current – While documentation should not become out-of-date, disseminating recent modifications to certain stakeholders may be ill-advised at times. Documentation remains the final authority, and stakeholders consult it for guidance when making decisions about the system. Including information that might not be final does not help them. Organizations are well-advised to determine a documentation release plan that is appropriate to their practices and processes.

(vii) Review Documentation for Fitness of Purpose – Documentation is successful only if it meets its readers' needs. Thus, these readers are the ones who determine its usefulness and should be encouraged to provide feedback about whether it meets their needs.

Q.4. Write short note on refinement.

Ans. Actually, refinement is a process of elaboration. A macroscopic statement of function is decomposed in a stepwise fashion to develop a hierarchy until programming language statements are reached. One or several instructions of the given program are decomposed into more detailed instructions in each step. The concepts of abstraction and refinement are complementary.

Q.5. Write short note on context diagram.

Ans. A context diagram is a top-level data flow diagram (DFD). It only contains one process node (process 0) that generalizes the function of the entire system in relationship to external entities.

Data flow diagrams (DFD) are used for portraying the overview of the entire system under development to depicting the detailed processing of a single transaction. The context-level DFDs show the main sinks, sources, processes and scope of the system under development using DFD symbols. The context diagram is shown in fig. 5.1.

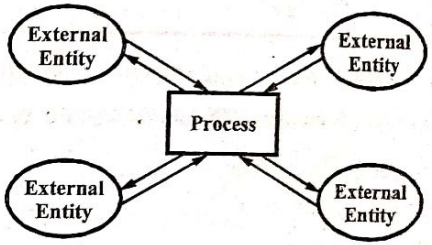


Fig. 5.1 Context Diagram

Q.6. Write short note on variability.

Ans. Variability is a special form of modifiability. It refers to the ability of a system and its supporting artifacts such as requirements, test plans and configuration specifications to support the production of a set of variants that differ from each other in a preplanned fashion. Variability is an especially important quality attribute in a software product line, where it means the ability of a core asset to adapt to usages in the different product contexts that are within the product line scope. The goal of variability in a software product line is to make it easy to build and maintain products in the product line over a period of time. Scenarios for variability will deal with the binding time of the variation and the people time to achieve it.

Q.7. Explain the term document interfaces in brief.

Ans. An interface is a boundary across which two independent entities meet and interact or communicate with each other. The document interfaces divided into nine parts as shown in fig. 5.2.

(i) Interface Identify – When an element has multiple interfaces, identify the individual interfaces to distinguish them. This usually means naming them. You may also need to provide a version number.

(ii) Resources Provided – The heart of an interface document is the resources that the element provides.

Section 2C. Element Interface Specification

Section 2.C.1. Interface identity
Section 2.C.2. Resources provided
Section 2.C.a. Resource syntax
Section 2.C.b. Resource semantics
Section 2.C.c. Resource usage restrictions
Section 2.C.3. Locally defined data types
Section 2.C.4. Exception definitions
Section 2.C.5. Variability provided
Section 2.C.6. Quality attribute characteristics
Section 2.C.7. Element requirements
Section 2.C.8. Rationale and design issues
Section 2.C.9. Usage guide

Fig. 5.2 The Nine Parts of Interface Documents

(a) Resource Syntax – This is the resource's signature

(b) Resource Semantics –
Assignment of values of data
Changes in state
Events signaled or message sent
how other resources will behave differently in future
humanly observable results

(c) Resource Usage Restrictions –
initialization requirements
limit on number of actors using resource

(iii) Data Type Definitions – If used if any interface resources employ a data type other than one provided by the underlying programming language, the architect needs to communicate the definition of that type. If it is defined by another element, then reference to the definition in that element's documentation is sufficient.

(iv) Exception Definitions – These describe exceptions that can be raised by the resources on the interface. Since the same exception might be raised by more than one resource, if it is convenient to simply list each resource's exceptions but define them in a dictionary collected separately.

(v) Variability Provided by the Interface – Does the interface allow the element to be configured in some way? These configuration parameters and how they affect the semantics of the interface must be documented.

(vi) Quality Attribute Characteristics – The architect needs to document what quality attribute characteristics (such as performance or reliability) the interface makes known to the element's users.

(vii) Element Requirements – What the element requires may be specific, named resources provided by other elements. The documentation obligation is the same as for resources provided – syntax, semantics, and any usage restrictions.

(viii) Rationale and Design Issues – Why these choices the architect should record the reasons for an elements interface design. The rationale should explain the motivation behind the design, constraints and compromises, what alternatives designs were considered.

(ix) Usage Guide – Item 2 and item 7 document an element's semantic information on a per resource basis. This sometimes falls short of what is needed. In some cases semantics need to be reasoned about in terms of how a broad number of individual interactions interrelate.

DOCUMENTING THE BEHAVIOUR OF SOFTWARE ELEMENTS AND SOFTWARE SYSTEMS, DOCUMENTATION PACKAGE USING A SEVEN PART TEMPLATE

Q.8. Write short note on documenting behaviour.

Ans. Views present structural information about the system. However, structural information is not sufficient to allow reasoning about some system properties behaviour description add information that reveals the ordering of interactions among the elements, opportunities for concurrency, and time dependencies of interactions. Behaviour can be documented either about an ensemble of elements working in concert. Exactly what to model will depend on the type of system being designed. Different modeling techniques and notations are used depending on the type of analysis to be performed. In UML, sequence diagrams and state charts are examples of behavioural descriptions. These notations are widely used.

Q.9. Discuss the documenting behaviour of software element.

Ans. Documenting an architecture requires behaviour documentation that complements structural views by describing how architecture elements interact with each other. There are two kinds of notations available for documenting behaviour. The first kind of notation is called trace-oriented languages, the second is called comprehensive languages.

A trace describes a sequence of activities or interactions between structural elements of the system. Although it is conceivable to describe all possible traces to generate the equivalent of a comprehensive behavioural model, it is not the intention of trace oriented documentation to do so. The four notations

for documenting traces are use cases, sequence diagrams, communication diagrams and activity diagrams. These four notations chosen as a representative sample of trace-oriented languages.

(i) Use Cases – These are frequently used to capture the functional requirement for a system. UML provides a graphical notation for use case diagrams but does not say how the text of a use case should be written. The UML use case diagram can be used effectively as an overview of the actors and the behaviour of a system.

The use case description is textual and should contain the use case name and brief description, the actor or actors who initiate the use case as primary actors, other actors who participate in the use case as secondary actors, alternative flows, flow of events, and nonsuccess cases.

(ii) Sequence Diagram – A UML sequence diagram shows a sequence of interactions among instances of elements pulled from the structural documentation. It shows only the instance participating in the scenario being documented. A sequence diagram has two dimensions vertical and horizontal. Vertical is representing time and horizontal is representing the various instances. The interactions are arranged in time sequence from top to bottom. A simple example of a UML sequence diagram is shown in fig. 5.3.

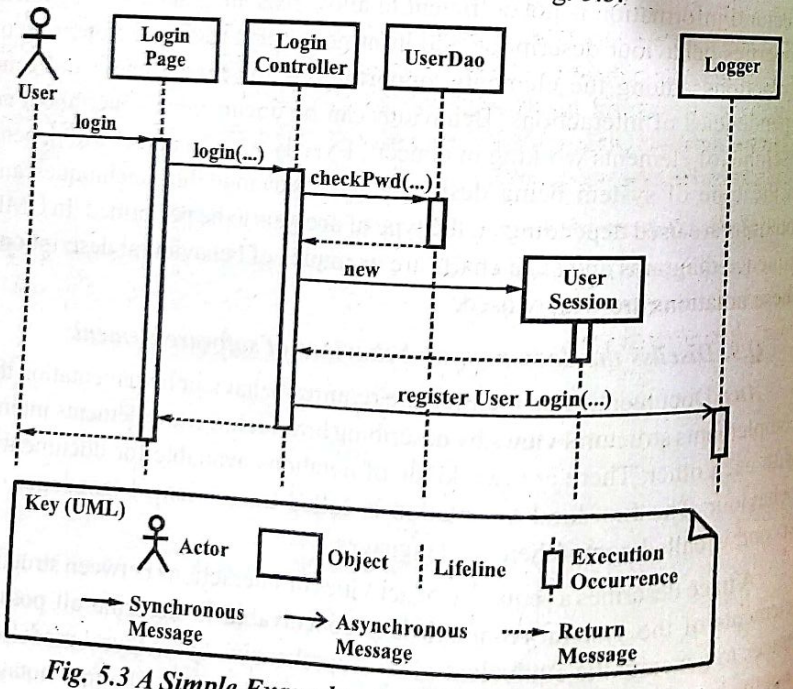


Fig. 5.3 A Simple Example of a UML Sequence Diagram

Objects (i.e. element instances) have a lifeline drawn as a vertical dashed line along the time axis. The sequence is usually started by an actor on the far left. The instances interact by sending messages, which are shown as horizontal arrows. A message can be a method or function call, an event sent through a queue, or something else. The message usually maps to a resource in the interface of the receiver instance. A filled arrowhead on a solid line represents a synchronous message, whereas the open arrowhead represents an asynchronous message. The dashed arrow is a return message. The execution occurrence bars along the lifeline indicate that the instance is processing or blocked waiting for a return.

(iii) Communication Diagram – This diagram shows a graph of interacting elements and annotates each interaction with a number denoting order. Communication diagrams are useful when the task is to verify that an architecture can fulfill the functional requirements. The diagrams are not useful if the understanding of concurrent actions is important, as when conducting a performance analysis.

(iv) Activity Diagram – These UML diagrams are similar to flow charts. They show a business process as a sequence of steps and include notation to express conditional branching and concurrency, as well as to show sending and receiving events. Arrows between actions indicate the flow of control. Optionally activity diagrams can indicate the architecture element or actor performing the actions. Activity diagrams can express concurrency. Activity diagrams are useful to broadly describe the steps in a specific workflow. Comprehensive models show the complete behaviour of structural elements in contrast to trace notations.

Q.10. Discuss about the documentation a cross-views.

Ans. Cross-view documentation consists of just three major aspects, which we can summarize as how-what-why. Fig. 5.4 shows the summary of cross-view documentation.

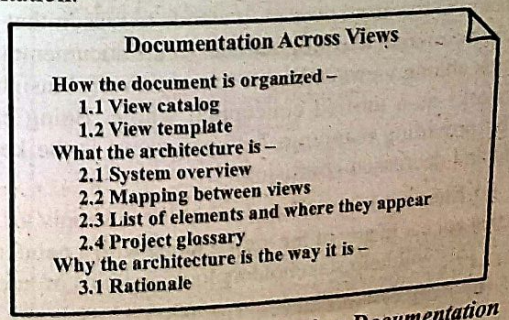


Fig. 5.4 Summary of Cross-view Documentation

(i) How the Documentation is Organized to Serve a Stakeholder – Every suite of architectural documentation needs an introductory piece to explain its organization to a novice stakeholder and to help that stakeholder access the information he or she is most interested in. There are two types of "how" information –

(a) View Catalog – A view catalog is the reader's introduction to the views that the architect has chosen to include in the suite of documentation. There is one entry in the view catalog for each view given in the documentation suite. Each entry should give the following –

(1) The name of the view and what style it instantiates

(2) A description of the view's element types, relation types, and properties.

(3) A description of what the view is for

Management information about the view document, such as the latest version, the location of the view document, and the owner of the view document.

(b) View Template – A view template is the standard organization for a view. It helps a reader navigate quickly to a section of interest, and it helps a writer organize the information and establish criteria for knowing how much work is left to do.

(ii) What the Architecture Is – This section provides information about the system whose architecture is being documented, the relation of the views to each other, and an index of architectural elements.

(a) System Overview – This is a short prose description of what the system's function is, who its users are, and any important background or constraints. The intent is to provide readers with a consistent mental model of the system and its purpose. Sometimes the project at large will have a system overview, in which case this section of the architectural documentation simply points to that.

(b) Mapping between Views – Since all of the views of an architecture describe the same system, it stands to reason that any two views will have much in common. Helping a reader of the documentation understand the relationships among views will give him a powerful insight into how the architecture works as a unified conceptual whole. Being clear about the relationship by providing mappings between views is the key to increased understanding and decreased confusion.

(c) Element List – The element list is simply an index of all of the elements that appear in any of the views, along with a pointer to where each one is defined. This will help stakeholders look up items of interest quickly.

(d) Project Glossary – The glossary lists and defines terms unique to the system that have special meaning. A list of acronyms, and the

meaning of each, will also be appreciated by stakeholders. If an appropriate glossary already exists, a pointer to it will suffice here.

(iii) Why the Architecture is the Way it is –

(a) Rationale – Cross-view rationale explains how the overall architecture is in fact a solution to its requirements. One might use the rationale to explain –

(1) The implications of system-wide design choices on meeting the requirements or satisfying constraints.

(2) The effect on the architecture when adding a foreseen new requirement or changing an existing one.

(3) The constraints on the developer in implementing a solution. Decision alternatives that were rejected.

In general, the rationale explains why a decision was made and what the implications are in changing it.

Q.11. Explain documentation package using a seven-part template.

Ans. View and beyond using seven part template for building the documentation package. A view divided five sections and beyond views divided into two sections.

A template for documenting a view is shown in fig. 5.5.

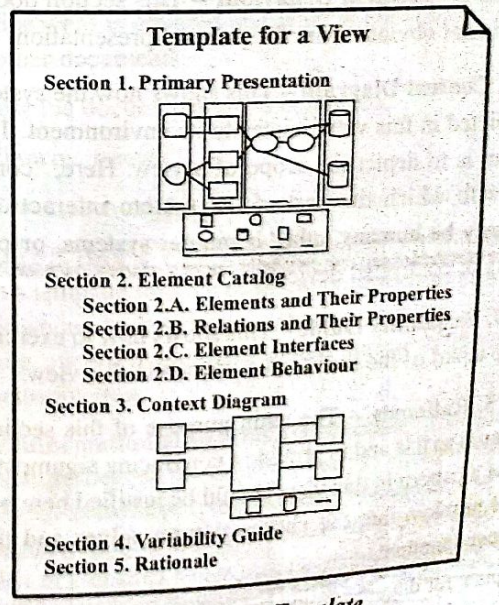


Fig. 5.5 View Template

The documentation for a view can be placed into a standard organization consisting of these parts –

Section 1. The Primary Presentation – This shows the elements and relations of the view. The primary presentation should contain the information you wish to convey about the system in the vocabulary of that view.

Section 2. The Element Catalog – This catalog details at least those elements depicted in the primary presentation. If elements or relations relevant to this view were omitted from the primary presentation, they should be introduced and explained in the catalog. Specific parts of the catalog divided in following subsection –

Section 2.1. Elements and their Properties – This section names each element in the view and lists the properties of that element.

Section 2.2 Relations and their Properties – Each view has specific relation types that it depicts among the elements in that view.

Section 2.3 Element Interface – This section documents element interfaces.

Section 2.4 Element Behaviour – This section documents element behaviour that is not obvious from the primary presentation.

Section 3. Context Diagram – This shows how the system or portion of the system depicted in this view relates to its environment. The purpose of a context diagram is to depict the scope of a view. Here, "context" means an environment with which the part of the system interacts. Entities in the environment may be humans, other computer systems, or physical objects, such as sensors or controlled devices.

Section 4. Variability Guide – This shows how to exercise any variation points that are a part of the architecture shown in this view.

Section 5. Rationale – The main purpose of this section is to explain why the design is as it is and to provide a convincing argument that it is sound. The choice of a pattern in this view should be justified here by describing the architectural problem that the chosen pattern solves and the rationale for choosing it over another.

A template for documentation beyond views is shown in fig. 5.6.

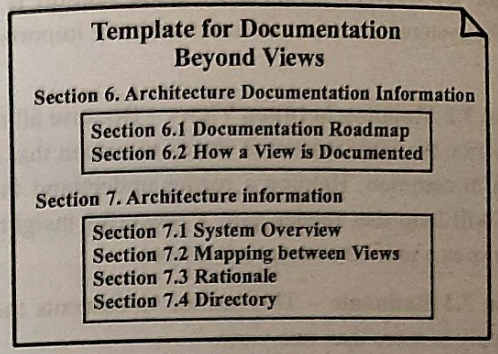


Fig. 5.6

Section 6. Overview of the Architecture Documentation – This tells how the documentation is laid out and organized so that a stakeholder of the architecture can find the information he or she needs efficiently and reliably. This divided two subsection as follows –

Section 6.1 Documentation Road Map – This tells the reader what information is in the documentation and where to find it. A documentation road map consists of four subsections –

(i) Scope and Summary – Explain the purpose of the document and briefly summarize what is covered and what is not covered and also explain the relation to other documents.

(ii) How the documentation is organized.

(iii) View overview.

(iv) How stakeholders can use the documentation.

Section 6.2 How a View is Documented – If your organization has standardized on a template for a view, as it should, then we can simply refer to that standard. If we are lacking such a template, then text such as that given above describing our view template should appear in this section of our architecture documentation.

Section 7. Information about the Architecture – The information that remains to be captured beyond the views themselves is a short system overview to ground any reader as to the purpose of the system and the way the views are related to one another, an overview of and rationale behind system-wide design approaches, a list of elements and where they appear, and a glossary and an acronym list for the entire architecture. This divided in four subsections –

Section 7.1 System Overview – This section is a short prose description of the system's function, its users and any important background or constraints.

Section 7.2 Mapping between Views – Because all the views of an architecture describe the same system, it stands to reason that any two views will have much in common. Helping a reader understand the associations between views will help that reader gain a powerful insight into how the architecture works as a unified conceptual whole.

Section 7.3 Rationale – This section documents the architectural decisions that apply to more than one view.

Section 7.4 Directory – The directory is a set of reference material that helps readers find more information quickly.

⌘⌘⌘