SFWR ENG 3A04: Software Design II

Dr. Ridha Khedri

Department of Computing and Software, McMaster University
Canada L8S 4L7, Hamilton, Ontario

Term 1

Acknowledgments: Material based on *Software Architecture Design* by Tao et al. (Chapter 6)

Outline of Part I

1. Repository Architecture Style

2. Blackboard Architecture Style

Outline of Part II

Part I

Review of Previous Lecture

Part II

Today's Lecture

Hierarchy Architecture Overview

- In this architectural style, the system is viewd as a hierarchical structure

Hierarchy Architecture Overview

- In this architectural style, the system is viewd as a hierarchical structure

- The software system is decomposed into functional modules (sub-systems)

Hierarchy Architecture Overview

- In this architectural style, the system is viewd as a hierarchical structure

- The software system is decomposed into functional modules (sub-systems)

- The modules at different levels are connected by explicit method invocations

Hierarchy Architecture Overview

- In this architectural style, the system is viewd as a hierarchical structure
- The software system is decomposed into functional modules (sub-systems)
- The modules at different levels are connected by explicit method invocations
- A lower level module provides services to its adjacent upper level modules

Hierarchy Architecture Overview

- In this architectural style, the system is viewd as a hierarchical structure
- The software system is decomposed into functional modules (sub-systems)
- The modules at different levels are connected by explicit method invocations
- A lower level module provides services to its adjacent upper level modules
- In procedure orientation, the lower level function and procedures may be organized in a header file or library

- In the object orientation, the lower level services may be organized in a package of classes

Hierarchy Architecture Overview

- In the object orientation, the lower level services may be organized in a package of classes
- Many system software (e.g., Unix) are built by hierarchical architecture

Hierarchy Architecture Overview

- In the object orientation, the lower level services may be organized in a package of classes
- Many system software (e.g., Unix) are built by hierarchical architecture
- The services at lower levels provide more specific fundamental utility service

Hierarchy Architecture Overview

- In the object orientation, the lower level services may be organized in a package of classes
- Many system software (e.g., Unix) are built by hierarchical architecture
- The services at lower levels provide more specific fundamental utility service
- The middle layer provides all business logic or core processing services

Hierarchy Architecture Overview

- In the object orientation, the lower level services may be organized in a package of classes
- Many system software (e.g., Unix) are built by hierarchical architecture
- The services at lower levels provide more specific fundamental utility service
- The middle layer provides all business logic or core processing services
- The upper layer provides user with interface

Hierarchy Architecture Overview

- In the object orientation, the lower level services may be organized in a package of classes
- Many system software (e.g., Unix) are built by hierarchical architecture
- The services at lower levels provide more specific fundamental utility service
- The middle layer provides all business logic or core processing services
- The upper layer provides user with interface
- Each layer is supported by its lower layer and provides service interface to its upper layer

Hierarchy Architecture Overview

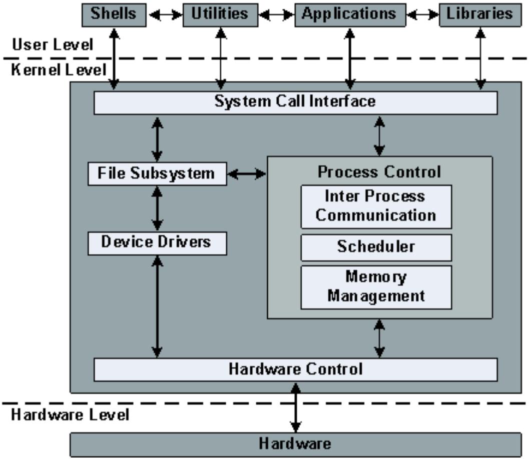

Figure: Unix Architecture

Hierarchy Architecture Overview

- The HA is characterized by explicit method invocation (call-and-return) connection styles

Hierarchy Architecture Overview

- The HA is characterized by explicit method invocation (call-and-return) connection styles

- It is used in the organization of class library

Hierarchy Architecture Overview

- The HA is characterized by explicit method invocation (call-and-return) connection styles
- It is used in the organization of class library
- It can be applied to procedure-oriented design, object-oriented design, component-oriented design, domain-specific design, and many others

Hierarchy Architecture Overview

- The HA is characterized by explicit method invocation (call-and-return) connection styles
- It is used in the organization of class library
- It can be applied to procedure-oriented design, object-oriented design, component-oriented design, domain-specific design, and many others
- It is hard to see any software that only uses one type architectural style

Hierarchy Architecture Overview

- The HA is characterized by explicit method invocation (call-and-return) connection styles
- It is used in the organization of class library
- It can be applied to procedure-oriented design, object-oriented design, component-oriented design, domain-specific design, and many others
- It is hard to see any software that only uses one type architectural style
- The hierarchical structure is one of the most popular styles that often combine with other styles

Hierarchy Architecture
Main/Subroutine Software Architecture

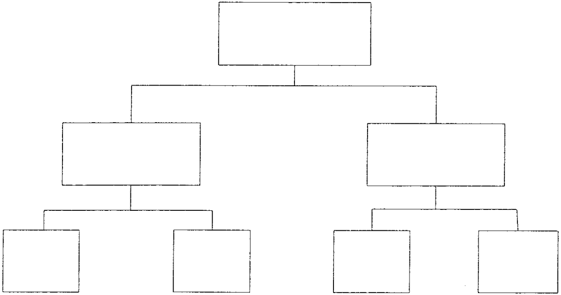

Figure: Typical hierarchical software architecture

Main/Subroutine Software Architecture

- The main/subroutine design architecture has dominated the software design methodologies for very long time

Hierarchy Architecture

Main/Subroutine Software Architecture

- The main/subroutine design architecture has dominated the software design methodologies for very long time
- Its purpose is to have maximum reuse of subroutines and make individual subroutine be developed independently

Hierarchy Architecture

Main/Subroutine Software Architecture

- The main/subroutine design architecture has dominated the software design methodologies for very long time
- Its purpose is to have maximum reuse of subroutines and make individual subroutine be developed independently
- In the classical procedure orientation, often all the data are shared by related subroutines at lowest level

Hierarchy Architecture

Main/Subroutine Software Architecture

- The main/subroutine design architecture has dominated the software design methodologies for very long time
- Its purpose is to have maximum reuse of subroutines and make individual subroutine be developed independently
- In the classical procedure orientation, often all the data are shared by related subroutines at lowest level
- In the object orientation, the data is encapsulated in each individual object

Hierarchy Architecture
Main/Subroutine Software Architecture

- The main/subroutine design architecture has dominated the software design methodologies for very long time
- Its purpose is to have maximum reuse of subroutines and make individual subroutine be developed independently
- In the classical procedure orientation, often all the data are shared by related subroutines at lowest level
- In the object orientation, the data is encapsulated in each individual object
- Often M/S style is refered to as the traditional style

Hierarchy Architecture
Main/Subroutine Software Architecture

- A system is decomposed into subroutines hierarchically according to the desired functionality of the system

Hierarchy Architecture
Main/Subroutine Software Architecture

- A system is decomposed into subroutines hierarchically according to the desired functionality of the system
    - Behaviour hiding (secrets = input formats, screen formats, messages)

Hierarchy Architecture

Main/Subroutine Software Architecture

- A system is decomposed into subroutines hierarchically according to the desired functionality of the system
  - Behaviour hiding (secrets = input formats, screen formats, messages)
  - Software decision hiding (secrets= algorithms and data structures)

Hierarchy Architecture
Main/Subroutine Software Architecture

- A system is decomposed into subroutines hierarchically according to the desired functionality of the system
  - Behaviour hiding (secrets = input formats, screen formats, messages)
  - Software decision hiding (secrets= algorithms and data structures)
  - Machine hiding (secrets= hardware machine, virtual machine, interfaces, etc.)

Hierarchy Architecture
Main/Subroutine Software Architecture

- A system is decomposed into subroutines hierarchically according to the desired functionality of the system
  - Behaviour hiding (secrets = input formats, screen formats, messages)
  - Software decision hiding (secrets= algorithms and data structures)
  - Machine hiding (secrets= hardware machine, virtual machine, interfaces, etc.)
- The refinements are conducted vertically until the decomposed subroutine is simple enough

Hierarchy Architecture
Main/Subroutine Software Architecture

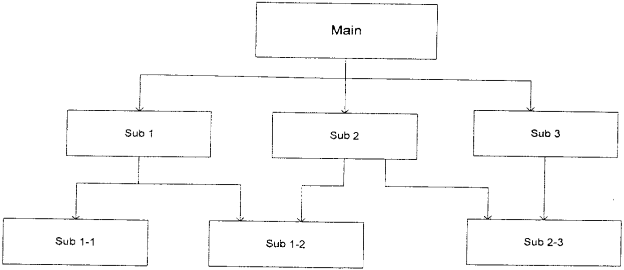

Figure: Main/subroutines architecture

Hierarchy Architecture

Main/Subroutine Software Architecture

- The main program is the program driver that has a mater control over the sequencing of its subroutines

Main/Subroutine Software Architecture

- The main program is the program driver that has a mater control over the sequencing of its subroutines
- How to map a requirement specification to the Main/Subroutine design structure?

Hierarchy Architecture

Main/Subroutine Software Architecture

- The main program is the program driver that has a mater control over the sequencing of its subroutines
- How to map a requirement specification to the Main/Subroutine design structure?
  - Start from a data flow view (Data Flow Diagram) of the requirements

Hierarchy Architecture

Main/Subroutine Software Architecture

- The main program is the program driver that has a mater control over the sequencing of its subroutines
- How to map a requirement specification to the Main/Subroutine design structure?
  - Start from a data flow view (Data Flow Diagram) of the requirements
  - We need to find transform or transaction flows:

Hierarchy Architecture

Main/Subroutine Software Architecture

- The main program is the program driver that has a mater control over the sequencing of its subroutines
- How to map a requirement specification to the Main/Subroutine design structure?
    - Start from a data flow view (Data Flow Diagram) of the requirements
    - We need to find transform or transaction flows:
        - Transform Flow (flow feeds in an external format, it is transformed into an internal format, and then carried out)

Hierarchy Architecture
Main/Subroutine Software Architecture

- The main program is the program driver that has a mater control over the sequencing of its subroutines
- How to map a requirement specification to the Main/Subroutine design structure?
  - Start from a data flow view (Data Flow Diagram) of the requirements
  - We need to find transform or transaction flows:
    - Transform Flow (flow feeds in an external format, it is transformed into an internal format, and then carried out)
    - Transaction Flow (evaluates its incoming data value, and decides on the path to follow)

Hierarchy Architecture
Main/Subroutine Software Architecture

- The main program is the program driver that has a mater control over the sequencing of its subroutines

- How to map a requirement specification to the Main/Subroutine design structure?
  - Start from a data flow view (Data Flow Diagram) of the requirements
  - We need to find transform or transaction flows:
    - Transform Flow (flow feeds in an external format, it is transformed into an internal format, and then carried out)
    - Transaction Flow (evaluates its incoming data value, and decides on the path to follow)

Hierarchy Architecture

Main/Subroutine Software Architecture

- How to map a requirement specification to the Main/Subroutine design structure? (Continued)

Hierarchy Architecture
Main/Subroutine Software Architecture

- How to map a requirement specification to the Main/Subroutine design structure? (Continued)
    - A transform flow is mapped to an M/S architecture with a controlling module for incoming, transform and outgoing information processing

Hierarchy Architecture
Main/Subroutine Software Architecture

- How to map a requirement specification to the Main/Subroutine design structure? (Continued)
  - A transform flow is mapped to an M/S architecture with a controlling module for incoming, transform and outgoing information processing

  - A transaction center is located at the fork origin of action paths

Hierarchy Architecture

Main/Subroutine Software Architecture

- How to map a requirement specification to the Main/Subroutine design structure? (Continued)
    - A transform flow is mapped to an M/S architecture with a controlling module for incoming, transform and outgoing information processing
    - A transaction center is located at the fork origin of action paths
    - Classify each action path to transform or transaction flows

Main/Subroutine Software Architecture

- How to map a requirement specification to the Main/Subroutine design structure? (Continued)
    - A transform flow is mapped to an M/S architecture with a controlling module for incoming, transform and outgoing information processing
    - A transaction center is located at the fork origin of action paths
    - Classify each action path to transform or transaction flows
    - The transaction centre becomes a dispatcher control module

Hierarchy Architecture

Main/Subroutine Software Architecture

- How to map a requirement specification to the Main/Subroutine design structure? (Continued)
  - A transform flow is mapped to an M/S architecture with a controlling module for incoming, transform and outgoing information processing
  - A transaction center is located at the fork origin of action paths
  - Classify each action path to transform or transaction flows
  - The transaction centre becomes a dispatcher control module
  - Factoring analysis continues until each module in the software architecture has its sole responsibility

Hierarchy Architecture

Main/Subroutine Software Architecture

- How to map a requirement specification to the Main/Subroutine design structure? (Continued)
  - A transform flow is mapped to an M/S architecture with a controlling module for incoming, transform and outgoing information processing
  - A transaction center is located at the fork origin of action paths
  - Classify each action path to transform or transaction flows
  - The transaction centre becomes a dispatcher control module
  - Factoring analysis continues until each module in the software architecture has its sole responsibility

Hierarchy Architecture
Main/Subroutine Software Architecture

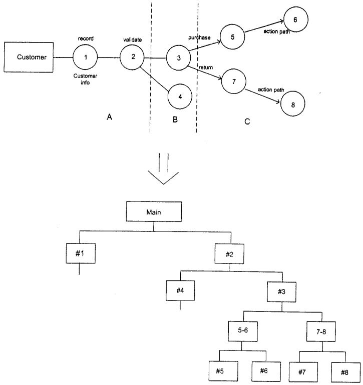

Figure: Mapped M/S structure

Hierarchy Architecture
Main/Subroutine Software Architecture

- Benefits

Hierarchy Architecture

Main/Subroutine Software Architecture

- Benefits
  - Easy to decompose the system based on the definition of the tasks in a top down refinements manner

Hierarchy Architecture
Main/Subroutine Software Architecture

- Benefits
  - Easy to decompose the system based on the definition of the tasks in a top down refinements manner
  - This architecture can still be used in a sub-system of OO Design

Hierarchy Architecture
Main/Subroutine Software Architecture

- Benefits
  - Easy to decompose the system based on the definition of the tasks in a top down refinements manner
  - This architecture can still be used in a sub-system of 00 Design
- Limitation

Hierarchy Architecture

Main/Subroutine Software Architecture

- Benefits
  - Easy to decompose the system based on the definition of the tasks in a top down refinements manner
  - This architecture can still be used in a sub-system of OO Design
- Limitation
  - Globally shared data in classical main/subroutines are vulnerable

Main/Subroutine Software Architecture

- Benefits
  - Easy to decompose the system based on the definition of the tasks in a top down refinements manner
  - This architecture can still be used in a sub-system of 00 Design
- Limitation
  - Globally shared data in classical main/subroutines are vulnerable
  - Tight coupling may cause ripple impacts compared to 00 Design

Master/Slaves Software Architecture

Master/Slaves Software Architecture

- The Master/Slaves architecture is a variant of the main/subroutine architecture style

Master/Slaves Software Architecture

Master/Slaves Software Architecture

- The Master/Slaves architecture is a variant of the main/subroutine architecture style
- It supports fault tolerance and system liability

Master/Slaves Software Architecture

Master/Slaves Software Architecture

- The Master/Slaves architecture is a variant of the main/subroutine architecture style
- It supports fault tolerance and system liability
- The slaves provide replicated services to the master

Hierarchy Architecture

Master/Slaves Software Architecture

Master/Slaves Software Architecture

- The Master/Slaves architecture is a variant of the main/subroutine architecture style
- It supports fault tolerance and system liability
- The slaves provide replicated services to the master
- The master selects a particular result among slaves by certain selection strategy

Hierarchy Architecture

Master/Slaves Software Architecture

Master/Slaves Software Architecture

- The Master/Slaves architecture is a variant of the main/subroutine architecture style
- It supports fault tolerance and system liability
- The slaves provide replicated services to the master
- The master selects a particular result among slaves by certain selection strategy
- The slaves may perform the same functional task by different algorithms and methods

Hierarchy Architecture
Master/Slaves Software Architecture

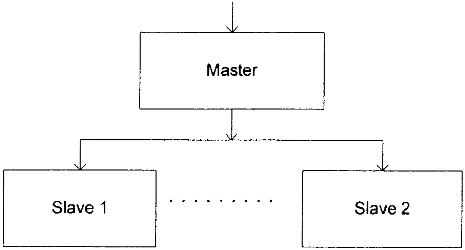

Figure: Block diagram for master/slaves architecture

Hierarchy Architecture
Master/Slaves Software Architecture

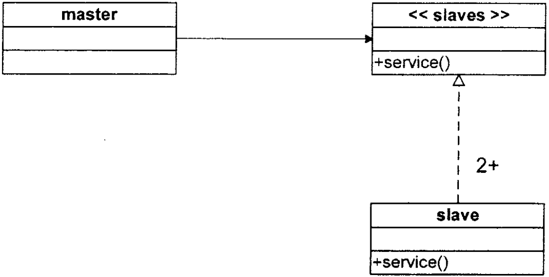

Figure: Class diagram for master/slaves architecture

Master/Slaves Software Architecture

- This architecture is suitable for parallel computing and accuracy of computation

Hierarchy Architecture

Master/Slaves Software Architecture

- This architecture is suitable for parallel computing and accuracy of computation
    - All slaves can be executed in parallel

Hierarchy Architecture

Master/Slaves Software Architecture

- This architecture is suitable for parallel computing and accuracy of computation
    - All slaves can be executed in parallel
    - A task is delegated to several different implementations, inaccurate results can be detected

Master/Slaves Software Architecture

- This architecture is suitable for parallel computing and accuracy of computation
    - All slaves can be executed in parallel
    - A task is delegated to several different implementations, inaccurate results can be detected
- Applicable Design Domains

Hierarchy Architecture
Master/Slaves Software Architecture

- This architecture is suitable for parallel computing and accuracy of computation
  - All slaves can be executed in parallel
  - A task is delegated to several different implementations, inaccurate results can be detected
- Applicable Design Domains
  - Software system where the liability is critical

Hierarchy Architecture

Master/Slaves Software Architecture

- This architecture is suitable for parallel computing and accuracy of computation
    - All slaves can be executed in parallel
    - A task is delegated to several different implementations, inaccurate results can be detected
- Applicable Design Domains
    - Software system where the liability is critical
    - Software system where performance is critical (to a certain limit –communication overhead–)

Hierarchy Architecture
Layered Architecture

Layered Architecture

- System is decomposed into a number of higher and lower layers

Hierarchy Architecture
Layered Architecture

Layered Architecture

- System is decomposed into a number of higher and lower layers

- Each layer consists of a group of related

Layered Architecture

- System is decomposed into a number of higher and lower layers
- Each layer consists of a group of related
  - classes in a format of package or deployed component (OOD)

Layered Architecture

- System is decomposed into a number of higher and lower layers
- Each layer consists of a group of related
  - classes in a format of package or deployed component (OOD)
  - a group of subroutines in the format of method library or header file

Layered Architecture

Layered Architecture

- System is decomposed into a number of higher and lower layers
- Each layer consists of a group of related
    - classes in a format of package or deployed component (OOD)
    - a group of subroutines in the format of method library or header file
- Each layer should have its own sole responsibility for the entire system

Layered Architecture

- A request to layer $i + 1$ invokes the services provided by the layer i via the later interface

Hierarchy Architecture

Layered Architecture

- A request to layer $i+1$ invokes the services provided by the layer i via the later interface

- The response may go back to the layer $i+1$ if the task is completed by this layer i

Hierarchy Architecture
Layered Architecture

- A request to layer $i+1$ invokes the services provided by the layer i via the later interface
- The response may go back to the layer $i+1$ if the task is completed by this layer i
- Otherwise, layer i continually invokes the layer $i-1$ below for services

Hierarchy Architecture

Layered Architecture

- A request to layer $i+1$ invokes the services provided by the layer i via the later interface
- The response may go back to the layer $i+1$ if the task is completed by this layer i
- Otherwise, layer i continually invokes the layer $i-1$ below for services
- The interface of each layer encapsulates all detail service implementations in the current layer or below

Layered Architecture

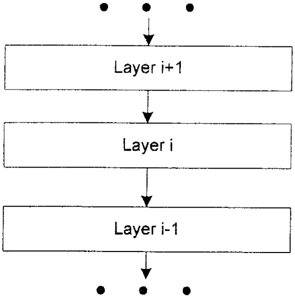

Figure: A partial layered architecture

Hierarchy Architecture

Layered Architecture

- The request from higher layer to the layer below is made via the method invocation and the response goes back up via the method return

Hierarchy Architecture

Layered Architecture

- The request from higher layer to the layer below is made via the method invocation and the response goes back up via the method return
- Each layer has two interfaces

Layered Architecture

- The request from higher layer to the layer below is made via the method invocation and the response goes back up via the method return
- Each layer has two interfaces
  - up interface provides services to its upper layer

Hierarchy Architecture

Layered Architecture

- The request from higher layer to the layer below is made via the method invocation and the response goes back up via the method return

- Each layer has two interfaces
  - up interface provides services to its upper layer
  - low interface requires services from its lower layer

Hierarchy Architecture

Layered Architecture

- The request from higher layer to the layer below is made via the method invocation and the response goes back up via the method return
- Each layer has two interfaces
  - up interface provides services to its upper layer
  - low interface requires services from its lower layer
- In a pure layered hierarchy, each layer only provides services to the adjacent upper layer directly and only requests services from the adjacent layer directly below

Hierarchy Architecture
Layered Architecture

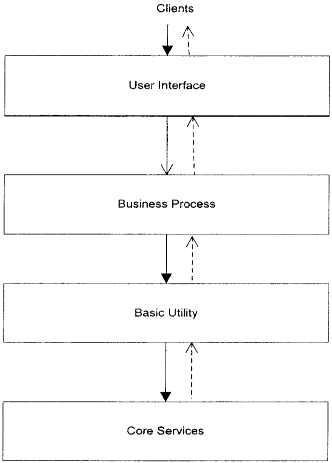

Figure: Business oriented software architecture

Hierarchy Architecture

Layered Architecture

- Higher layer provides more generic or application oriented services (abstract)

Layered Architecture

- Higher layer provides more generic or application oriented services (abstract)
- Lower layer provides more specific utility type services

Hierarchy Architecture

Layered Architecture

- Higher layer provides more generic or application oriented services (abstract)

- Lower layer provides more specific utility type services

- To encapsulate all the services in one layer, we can deploy each layer in a component format (such as a JAR file (Java ARchive)))

Layered Architecture

- Higher layer provides more generic or application oriented services (abstract)
- Lower layer provides more specific utility type services
- To encapsulate all the services in one layer, we can deploy each layer in a component format (such as a JAR file (Java ARchive)))
- A JAR file is a compressed file which is deployed as a component of a package

Hierarchy Architecture
Layered Architecture

- A jar file includes all the service classes from lower level plus other related classes provided in the same layer and provided by Java API

Hierarchy Architecture
Layered Architecture

- A jar file includes all the service classes from lower level plus other related classes provided in the same layer and provided by Java API
    - The Java API is the set of classes included with the Java Development Environment

Hierarchy Architecture

Layered Architecture

- A jar file includes all the service classes from lower level plus other related classes provided in the same layer and provided by Java API
  - The Java API is the set of classes included with the Java Development Environment
  - These classes are written using the Java language and run on the JVM

Hierarchy Architecture
Layered Architecture

- A jar file includes all the service classes from lower level plus other related classes provided in the same layer and provided by Java API
    - The Java API is the set of classes included with the Java Development Environment
    - These classes are written using the Java language and run on the JVM
    - The Java API includes everything from collection classes to GUI classes

Layered Architecture

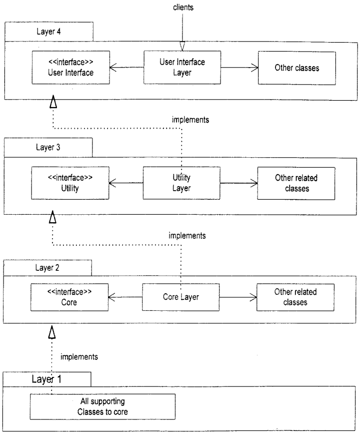

Figure: Component-based layered architecture

Hierarchy Architecture

Layered Architecture

A simple software system may consist of two layers (Interaction& Processing)

- Interaction Layer

Hierarchy Architecture

Layered Architecture

A simple software system may consist of two layers (Interaction& Processing)

- Interaction Layer
  - It provides user Interfaces to clients

Hierarchy Architecture
Layered Architecture

A simple software system may consist of two layers (Interaction& Processing)

- Interaction Layer
    - It provides user Interfaces to clients
    - It takes requests

Layered Architecture

A simple software system may consist of two layers (Interaction& Processing)

- Interaction Layer
  - It provides user Interfaces to clients
  - It takes requests
  - It validates and forwards request to processing layer for processing

Hierarchy Architecture
Layered Architecture

A simple software system may consist of two layers (Interaction& Processing)

- Interaction Layer
  - It provides user Interfaces to clients
  - It takes requests
  - It validates and forwards request to processing layer for processing
  - It responds to clients

Hierarchy Architecture

Layered Architecture

A simple software system may consist of two layers (Interaction& Processing)

- Interaction Layer
  - It provides user Interfaces to clients
  - It takes requests
  - It validates and forwards request to processing layer for processing
  - It responds to clients
- Processing Layer

Hierarchy Architecture
Layered Architecture

A simple software system may consist of two layers (Interaction& Processing)

- Interaction Layer
  - It provides user Interfaces to clients
  - It takes requests
  - It validates and forwards request to processing layer for processing
  - It responds to clients
- Processing Layer
  - It gets the forwarded requests and performs business logic process

Hierarchy Architecture
Layered Architecture

A simple software system may consist of two layers (Interaction& Processing)

- Interaction Layer
    - It provides user Interfaces to clients
    - It takes requests
    - It validates and forwards request to processing layer for processing
    - It responds to clients
- Processing Layer
    - It gets the forwarded requests and performs business logic process
    - It accesses database

Layered Architecture

A simple software system may consist of two layers (Interaction& Processing)

- Interaction Layer
    - It provides user Interfaces to clients
    - It takes requests
    - It validates and forwards request to processing layer for processing
    - It responds to clients
- Processing Layer
    - It gets the forwarded requests and performs business logic process
    - It accesses database
    - It returns the results to its upper layer

Hierarchy Architecture
Layered Architecture

A simple software system may consist of two layers (Interaction& Processing)

- Interaction Layer
  - It provides user Interfaces to clients
  - It takes requests
  - It validates and forwards request to processing layer for processing
  - It responds to clients
- Processing Layer
  - It gets the forwarded requests and performs business logic process
  - It accesses database
  - It returns the results to its upper layer
  - It lets upper layer respond to clients

Hierarchy Architecture
Layered Architecture

A simple software system may consist of two layers (Interaction& Processing)

- Interaction Layer
    - It provides user Interfaces to clients
    - It takes requests
    - It validates and forwards request to processing layer for processing
    - It responds to clients
- Processing Layer
    - It gets the forwarded requests and performs business logic process
    - It accesses database
    - It returns the results to its upper layer
    - It lets upper layer respond to clients

Hierarchy Architecture
Layered Architecture

A simple software system may consist of two layers (Interaction& Processing)

- Interaction Layer
  - It provides user Interfaces to clients
  - It takes requests
  - It validates and forwards request to processing layer for processing
  - It responds to clients
- Processing Layer
  - It gets the forwarded requests and performs business logic process
  - It accesses database
  - It returns the results to its upper layer
  - It lets upper layer respond to clients (since the upper layer has the GUI interface responsibility)

Hierarchy Architecture
Layered Architecture

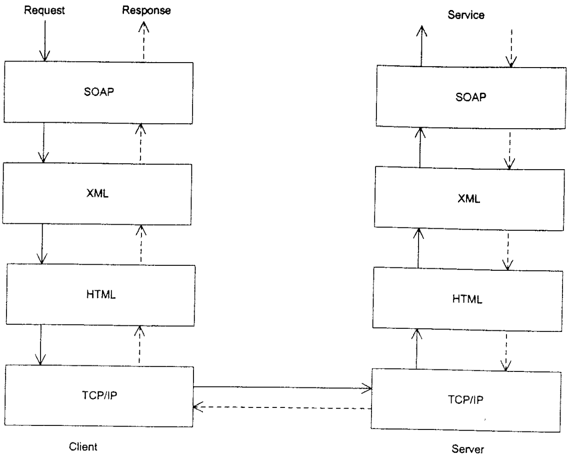

Figure: Object Access Protocol (SOAP) layered architecture

Hierarchy Architecture

Layered Architecture

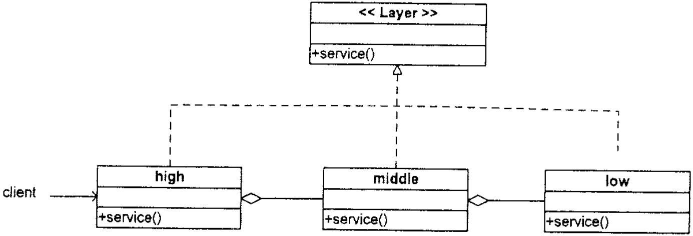

Figure: Class diagram for layered architecture

Hierarchy Architecture
Layered Architecture

- Applicable Design Domains

Hierarchy Architecture

Layered Architecture

- Applicable Design Domains
    - Any system that can be divided between the application specific portions and platform specific portions

Hierarchy Architecture
Layered Architecture

- Applicable Design Domains
  - Any system that can be divided between the application specific portions and platform specific portions
  - Applications that have clean divisions between core services, critical services, user interface services

Hierarchy Architecture

Layered Architecture

- Applicable Design Domains
  - Any system that can be divided between the application specific portions and platform specific portions
  - Applications that have clean divisions between core services, critical services, user interface services
  - Applications that have a number of classes that are closely related to each other so that they can be grouped together to provide the services to others.

Layered Architecture

- Applicable Design Domains
  - Any system that can be divided between the application specific portions and platform specific portions
  - Applications that have clean divisions between core services, critical services, user interface services
  - Applications that have a number of classes that are closely related to each other so that they can be grouped together to provide the services to others.
- Benefit

Hierarchy Architecture

Layered Architecture

- Applicable Design Domains
  - Any system that can be divided between the application specific portions and platform specific portions
  - Applications that have clean divisions between core services, critical services, user interface services
  - Applications that have a number of classes that are closely related to each other so that they can be grouped together to provide the services to others.
- Benefit
  - Incremental software development based on increasing levels of abstraction

Hierarchy Architecture

Layered Architecture

- Applicable Design Domains
  - Any system that can be divided between the application specific portions and platform specific portions
  - Applications that have clean divisions between core services, critical services, user interface services
  - Applications that have a number of classes that are closely related to each other so that they can be grouped together to provide the services to others.
- Benefit
  - Incremental software development based on increasing levels of abstraction
  - Enhanced independence of layers

Hierarchy Architecture
Layered Architecture

- Applicable Design Domains
  - Any system that can be divided between the application specific portions and platform specific portions
  - Applications that have clean divisions between core services, critical services, user interface services
  - Applications that have a number of classes that are closely related to each other so that they can be grouped together to provide the services to others.
- Benefit
  - Incremental software development based on increasing levels of abstraction
  - Enhanced independence of layers
  - Enhanced reusability

Hierarchy Architecture
Layered Architecture

- Applicable Design Domains
  - Any system that can be divided between the application specific portions and platform specific portions
  - Applications that have clean divisions between core services, critical services, user interface services
  - Applications that have a number of classes that are closely related to each other so that they can be grouped together to provide the services to others.
- Benefit
  - Incremental software development based on increasing levels of abstraction
  - Enhanced independence of layers
  - Enhanced reusability
  - Component-based technology is a suitable technology to implement the layered structure (plug-and-play)

Hierarchy Architecture
Layered Architecture

- Applicable Design Domains
  - Any system that can be divided between the application specific portions and platform specific portions
  - Applications that have clean divisions between core services, critical services, user interface services
  - Applications that have a number of classes that are closely related to each other so that they can be grouped together to provide the services to others.
- Benefit
  - Incremental software development based on increasing levels of abstraction
  - Enhanced independence of layers
  - Enhanced reusability
  - Component-based technology is a suitable technology to implement the layered structure (plug-and-play)
  - Promotion of portability: each layer can be an abstract machine deployed independently

Hierarchy Architecture

Layered Architecture

- Limitations

Hierarchy Architecture

Layered Architecture

- Limitations
  - Lower runtime performance (a client's request/response goes through many layer)

Hierarchy Architecture

Layered Architecture

- Limitations
  - Lower runtime performance (a client's request/response goes through many layer)
  - Performance concerns on overhead on the data marshaling and buffering by each layer

Layered Architecture

- Limitations
  - Lower runtime performance (a client's request/response goes through many layer)
  - Performance concerns on overhead on the data marshaling and buffering by each layer
  - Many applications can not fit this architecture

Hierarchy Architecture

Layered Architecture

- Limitations
  - Lower runtime performance (a client's request/response goes through many layer)
  - Performance concerns on overhead on the data marshaling and buffering by each layer
  - Many applications can not fit this architecture
  - Exception and error handling is an issue in the layered architecture

Hierarchy Architecture

Layered Architecture

- Limitations
  - Lower runtime performance (a client's request/response goes through many layer)
  - Performance concerns on overhead on the data marshaling and buffering by each layer
  - Many applications can not fit this architecture
  - Exception and error handling is an issue in the layered architecture
- Related architecture

Hierarchy Architecture
Layered Architecture

- Limitations
  - Lower runtime performance (a client's request/response goes through many layer)
  - Performance concerns on overhead on the data marshaling and buffering by each layer
  - Many applications can not fit this architecture
  - Exception and error handling is an issue in the layered architecture

- Related architecture
  - Repository, client/server, virtual machine

Virtual Machine

- It is built on an existing system

Hierarchy Architecture
Virtual Machine

Virtual Machine

- It is built on an existing system
- It separates a programming language, hardware language, or application from a physical execution platform

Hierarchy Architecture
Virtual Machine

Virtual Machine

- It is built on an existing system
- It separates a programming language, hardware language, or application from a physical execution platform
- It plays the role of an emulation software

Hierarchy Architecture
Virtual Machine

Virtual Machine

- It is built on an existing system
- It separates a programming language, hardware language, or application from a physical execution platform
- It plays the role of an emulation software
  - It provides an emulation of the functions of one system using a different system

Hierarchy Architecture
Virtual Machine

Virtual Machine

- It is built on an existing system
- It separates a programming language, hardware language, or application from a physical execution platform
- It plays the role of an emulation software
    - It provides an emulation of the functions of one system using a different system
    - It allows exact reproduction of external behavior of a system

Hierarchy Architecture

Virtual Machine

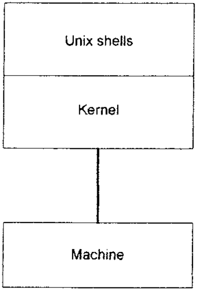

Figure: Unix virtual machine

Hierarchy Architecture
Virtual Machine

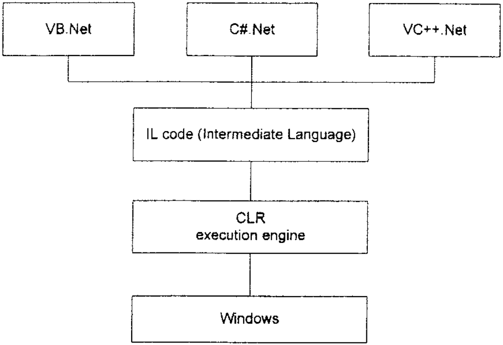

Figure: Common Language Runtime (CLR) virtual machine in .NET platform

Hierarchy Architecture
Virtual Machine

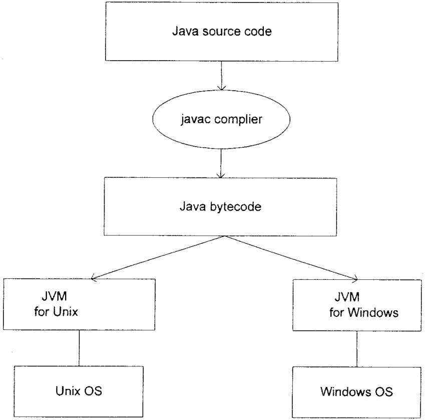

Figure: Java virtual machine

Hierarchy Architecture

Virtual Machine

- Applicable Design Domain

Hierarchy Architecture

Virtual Machine

- Applicable Design Domain
    - Solving a problem by simulation or translation

Hierarchy Architecture
Virtual Machine

- Applicable Design Domain
    - Solving a problem by simulation or translation
    - Interpreters of microprogramming, XML processing, script command language execution, rule-based system execution, Small talk and Java interpreter typed programming language

Hierarchy Architecture
Virtual Machine

- Applicable Design Domain
    - Solving a problem by simulation or translation
    - Interpreters of microprogramming, XML processing, script command language execution, rule-based system execution, Small talk and Java interpreter typed programming language
- Benefits

Hierarchy Architecture
Virtual Machine

- Applicable Design Domain
  - Solving a problem by simulation or translation
  - Interpreters of microprogramming, XML processing, script command language execution, rule-based system execution, Small talk and Java interpreter typed programming language
- Benefits
  - Portability and machine platform independence

Hierarchy Architecture

Virtual Machine

- Applicable Design Domain
  - Solving a problem by simulation or translation
  - Interpreters of microprogramming, XML processing, script command language execution, rule-based system execution, Small talk and Java interpreter typed programming language
- Benefits
  - Portability and machine platform independence
  - Simplicity of the software development

Hierarchy Architecture
Virtual Machine

- Applicable Design Domain
  - Solving a problem by simulation or translation
  - Interpreters of microprogramming, XML processing, script command language execution, rule-based system execution, Small talk and Java interpreter typed programming language
- Benefits
  - Portability and machine platform independence
  - Simplicity of the software development
  - Simulation for non-native model

Hierarchy Architecture
Virtual Machine

- Limitations

Hierarchy Architecture

Virtual Machine

- Limitations
  - Slow execution of the interpreter

Hierarchy Architecture
Virtual Machine

- Limitations
  - Slow execution of the interpreter
  - Additional overhead due to the new layer

Hierarchy Architecture
Virtual Machine

- Limitations
  - Slow execution of the interpreter
  - Additional overhead due to the new layer
- Related architecture

Hierarchy Architecture

Virtual Machine

- Limitations
    - Slow execution of the interpreter
    - Additional overhead due to the new layer
- Related architecture
    - Interpreter, repository, layered architecture