Systems programming

3 – Architectures

MEEC MEAer MEIC MEFT
João Nuno Silva

Bibliography

- Microsoft Application Architecture Guide
  - Chapter 1, 2,3, 4
- An Introduction to Software Architecture
  - David Garlan and Mary Shaw

Systems Architecture

Systems architecture

- Generic discipline to handle systems
  - To support reasoning about its properties
- Systems Architecture can refer to
  - the actual architecture of a system
    - i.e. a model to describe/analyze a system
  - Process of architecting/designing a system
    - i.e. a method to define the architecture of a system
  - body of knowledge for ”architecting” systems to meet business needs
    - i.e. a discipline to master systems design.

Systems architecture

- The **architecture of a system** is
  - a global model of a real system
- Consists of:
  - Structure
  - Properties of various elements involved
  - Relationships between various elements
  - Behaviors and dynamics
  - Multiple views of elements

Systems architecture

- Process of defining a structured solution
  - that meets all of the technical and operational requirements,
  - while optimizing common quality attributes
    - performance, security, and manageability

Systems Architecture

- Body of knowledge for
  - Architecting/ designing systems
    - while meeting business needs,
  - Designing systems
  - Describing
  - Understanding
- Discipline to master systems design.
  - consisting in concepts, principles, frameworks, tools, methods, heuristics, practices

Architectural principles and Design practices

Key Architectural Principles

- Build to change instead of building to last
- Model to analyze and reduce risk
- Use models and visualizations
  - as a communication and collaboration tool
- Identify key engineering decisions

Key Architectural Principles

- Architecture/design testing
  - What assumptions were made in the architecture?
  - What explicit or implied requirements
    - this architecture meets?
  - What are the key risks with this architectural approach?
  - What countermeasures are in place to mitigate key risks?
  - How the architecture is
    - improvement over the baseline or previous architecture?

Key Design Principles

- Separation of concerns.
  - Divide your application into distinct features with minimal overlap in functionality
  - Minimize interaction points to achieve high cohesion and low coupling.
- Single Responsibility principle
  - Each component or module should be responsible for only a specific feature or functionality, or aggregation of cohesive functionality.
- Don’t repeat yourself
  - Specific functionality should be implemented in only one component
  - Each functionality should not be duplicated in any other component.

Key Design Principles

- Principle of Least Knowledge
  - A component or object should not know about internal details of other components or objects.
  - also known as the Law of Demeter or LoD
- Minimize upfront design.
  - Only design what is necessary.
  - In some cases,
    - you may require upfront comprehensive design and testing if the cost of development or a failure in the design is very high.
  - In other cases
    - you can avoid big design upfront

Design Practices

- Keep design patterns consistent within each layer.
- Do not duplicate functionality within an application.
- Prefer composition to inheritance.
  - Inheritance increases the dependency between parent and child classes
  - Composition only relies on the interface
- Establish a coding style and naming convention for development

Architectural Patterns and Styles

Architectural Patterns and Styles

- Problem
  - How to design components?
  - How to structure a logical solution?
  - How to organize the codebase?
- Solution
  - Know the language
  - Know best practices
  - Reuse slutions

Architectural Patterns and Styles

- Set of principles
  - a coarse grained pattern that provides an abstract framework for a family of systems.
- Provides solutions to frequently recurring problems.
  - improves partitioning and promotes design reuse
  - Reduces problems
- Sets of principles that shape an application.

Architectural Patterns and Styles

- Architectural style determines
  - vocabulary of components and connectors that can be used in instances of that style,
  - set of constraints on how they can be combined.

Architectural Patterns and Styles

- Structure
  - Component-Based
  - Object-Oriented
  - Data Abstraction
  - Layered Architecture
  - Table Driven Interpreters
  - State transition systems
- Deployment
  - Client/Server
  - N-Tier / 3-Tier
  - Peer-to-peer
- Communication
  - RPC
  - Service-Oriented Architecture (SOA
  - Message Bus,
  - Pipes and Filters,
  - Repositories
  - Event-based, Implicit Invocation

Structure Patterns

Structure Patterns

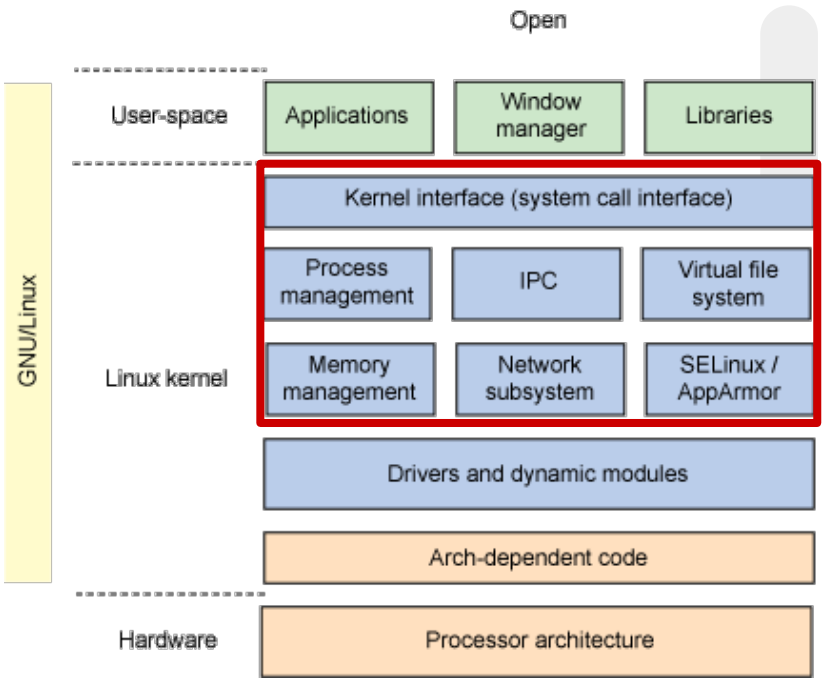
- Problem to be solved
  - How to design components?
  - How to structure a logical solution?
  - How to organize the codebase?
- Structure
  - Component-Based
  - Object-Oriented
  - Data Abstraction
  - Layered Architecture
  - Table Driven Interpreters
  - State transition systems

Data Abstraction

- Data representations and operations
  - encapsulated in an abstract data type
- The components are
  - Modules, libraries and data structures
- Modules interact through functions and procedures
- Modules should preserve
  - integrity of representation (interface)
  - hide implementation

Data Abstraction

- Each modules
  - is implemented in independent files
  - Exports a set of functions
  - Hide internal representation
- Stdlib



Components

- Programming language objects
  - Modules, libraries, functions or procedures
  - Provide data abstraction
- A component should not rely on internal details of other components
  - Each component should call functions of another component,
    - should have information about how to process the request
    - Or how to route it to a subcomponents or external components

Object-Oriented

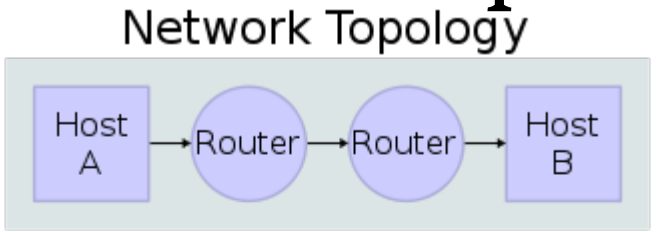
- Classes and Objects
  - Evolution of data abstraction and components
  - encapsulate data and associated operations
  - Structures with pointer to functions
- Each object encapsulates state
  - Different form other objects
- Each object inherits from classes
- Connectors are method invocations

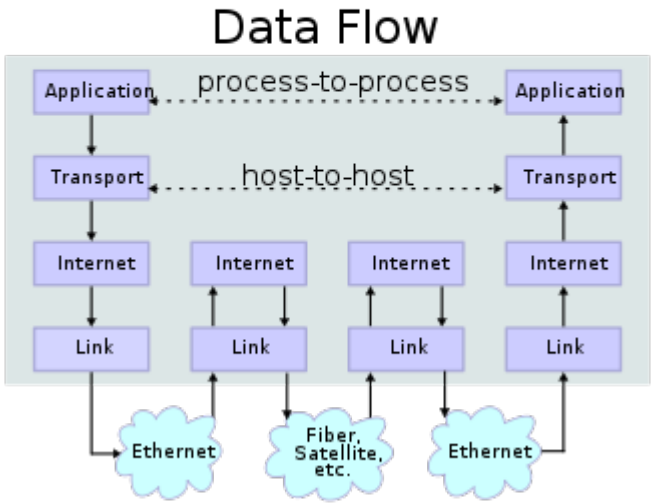
Layered Architecture

- Focuses on the grouping of related functionality
  - into distinct layers.
  - Functionality within each layer is related by a common role or responsibility.
- A layered system is organized hierarchically,
  - each layer provides service to the layer above it and uses the layer below.
- Communication between layers is explicit and loosely coupled.
- Helps to support a strong separation of concerns that, in turn, supports flexibility and maintainability.

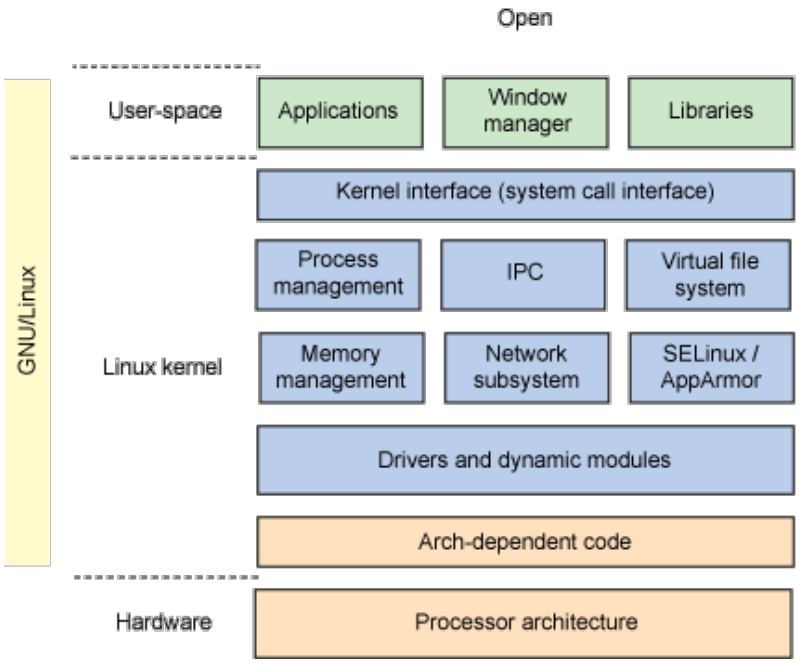
Layered Architecture

communication protocols





Operating systems



Layered Architecture

- Increasing levels of abstraction.
- Support enhancement.
- Support reuse.
- Isolation
- Manageability
- Performance.
- Testability.

Layered Architecture

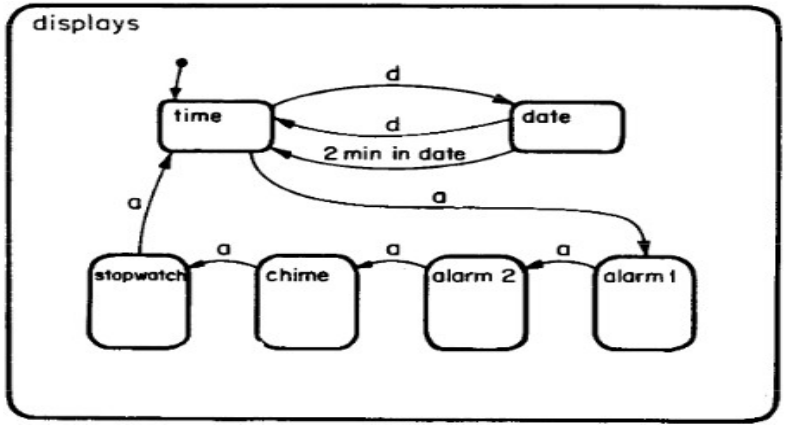
- Not all systems are easily structured in a layered fashion.
  - performance may require closer coupling between logically high-level functions and their lower-level implementations.
- It can be difficult to find the right levels of abstraction.

Table Driven Interpreters/VMs

- Virtual machines are used as interpreters
- An interpreter generally has four components:
  - an interpretation engine to do the work,
  - a memory that contains the pseudo-code to be interpreted
  - a representation of the control state of the interpretation engine,
  - a representation of the current state of the program being simulated.
- Interpreters are commonly used to build virtual machines

State transition systems

- Systems are defined in terms of
  - set of states
  - set of named transitions that move a system from one state to another.



Deployment Patterns

Deployment Patterns

- Problem to be solved
  - How to distribute/deploy components sub-systems on different hardware
  - how to distribute responsibilities among node?
- Deployment
  - Client/Server
  - N-Tier / 3-Tier
  - Peer-to-peer

Client/Server

- Distributed systems that
  - involve a separate client and server system,
  - and a connecting network.
- The simplest form of client/server system
  - involves a server application that is accessed directly by multiple clients,

Client/Server

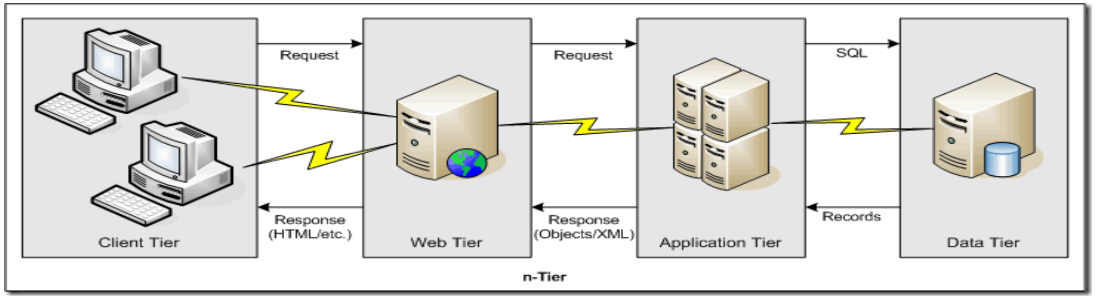
- Describes the relationship between a client and one or more servers,
  - where the client initiates one or more requests
  - waits for replies,
  - processes the replies on receipt.
- The server when receiving a request
  - authorizes the user
  - carries out the processing required to generate the result.
  - Sends a response
- Communication can be done using a range of protocols and data formats

Client/Server

- Higher security.
  - All data is stored on the server, which generally offers a greater control of security than client machines.
- Centralized data access.
  - Because data is stored only on the server, access and updates to the data are far easier to administer than in other architectural styles.
- Ease of maintenance
  - Roles and responsibilities of a computing system are distributed among several servers that are known to each other through a network.
  - Ensures that a client remains unaware and unaffected by a server repair, upgrade, or relocation.

N-Tier / 3-Tier

- Describes the separation of functionality into segments
  - Similar to the layered style,
  - but with each tier on a separate computer.
- In distributed systems a layer can be in different tiers



N-Tier / 3-Tier

- N-tier application architecture is characterized by
  - the functional decomposition of applications, service components
  - improves scalability, availability, manageability, and resource utilization.
  - Each tier is completely independent from all other tiers, except for those immediately above and below it.

Deployment Patterns
N-Tier / 3-Tier

- Maintainability.
  - Each tier is independent of the other tiers,
  - updates or changes can be carried out without affecting the application as a whole.
- Scalability
  - Tiers are based on the deployment of layers,
  - scaling out an application is reasonably straightforward.
- Flexibility.
  - Each tier can be managed or scaled independently
- Availability
  - Applications can exploit the modular architecture of enabling systems using easily scalable components, which increases availability.

Client-Queue-Client

- This approach allows clients to
  - communicate with other clients
  - through a server-based queue.
- Clients can read data from and send data to a server
  - server acts simply as a queue to store the data.
  - This allows clients to distribute and synchronize files and information.

Peer-to-peer

- Evolution of the Client-Queue-Client style,
  - P2P style allows the client and server to swap their roles
    - to distribute/synchronize files and information across multiple clients.
  - extends the client/server architecture through multiple
    - responses to requests,
    - shared data,
    - resource discovery,
    - resilience to removal of peers.

Peer-to-peer

- State and behavior are distributed among peers which can act as either clients or servers.
  - A single component can be a client and a server
- Each peer contributes resources to the system
  - All nodes have the same capabilities and responsibilities
- Correct operation not dependent on a central system
- Can be design to offer some anonymity

Communication Patterns

Communication Patterns

- Problem to be solved
  - How to connect 2 components
    - Just one main.c
    - Copy & paste of A.c B.c into main.c
    - gcc main.c
  - How to connect 2 components
    - Just one main.c
    - Transform B.c → libB.o A.c → libA.o
    - gcc main.c -lB -lA
  - How to connect 2 components
    - In different computers…
    -
  - How to integrate multiple applications
    - so that they work together
    - and can exchange information
    - and guarantee consistency?
    - With ease of programming

Communication Patterns

- How to connect 2 components
- Solution
  - Make them independent components
- Select a communication pattern
  - To be studied latter

- Communication
  - RPC
  - Service-Oriented Architecture
  - Message Bus,
  - Pipes and Filters,
  - Repositories
  - Event-based, Implicit Invocation