SOFTWARE ENGINEERING

Week 7

Software Architecture and High Level Design

Agenda

1. Architectural Design of Software
2. Architecture Models

1. Architectural Design of Software
2. Architecture Models

Architectural Design of Software

7.1

Software architecture

- The design process for identifying the sub-systems making up a system and the framework for sub-system control and communication is architectural design.
- The output of this design process is a description of the software architecture.

Architectural design

- An early stage of the system design process.
- Represents the link between specification and design processes.
- Often carried out in parallel with some specification activities.
- It involves identifying major system components and their communications.

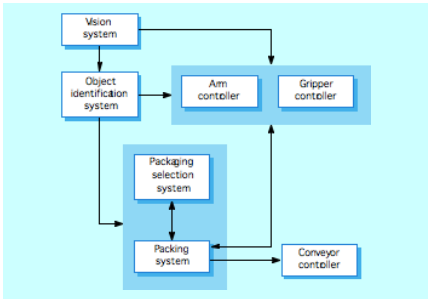
Advantages of explicit architecture

- Stakeholder communication
  - Architecture may be used as a focus of discussion by system stakeholders.
- System analysis
  - Means that analysis of whether the system can meet its non-functional requirements is possible.
- Large-scale reuse
  - The architecture may be reusable across a range of systems.

Architecture and system characteristics

- Performance
  - Localise critical operations and minimise communications. Use large rather than fine-grain components.
- Security
  - Use a layered architecture with critical assets in the inner layers.
- Safety
  - Localise safety-critical features in a small number of sub-systems.
- Availability
  - Include redundant components and mechanisms for fault tolerance.
- Maintainability
  - Use fine-grain, replaceable components.

Packing robot control system



Box and line diagrams

- Very abstract - they do not show the nature of component relationships nor the externally visible properties of the sub-systems.
- However, useful for communication with stakeholders and for project planning.

1. Architectural Design of Software
2. Architecture Models

Architecture Models

7.2

Architecture Models

- Repository
- Pipe and Filter
- Event-driven / Asynchronous

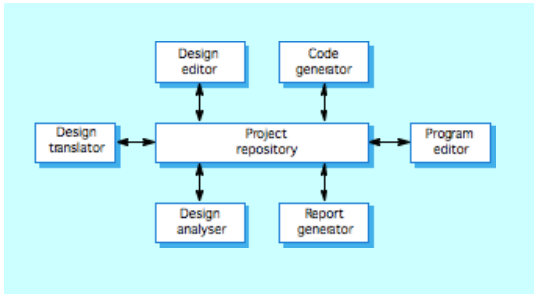
- Client Server
- P2P

- Layered
- MVC
- Service Oriented
- Microservices

The repository model

- Sub-systems must exchange data. This may be done in two ways:
  - Shared data is held in a central database or repository and may be accessed by all sub-systems: ***the repository model***;
  - Each sub-system maintains its own database and passes data explicitly to other sub-systems.
- When large amounts of data are to be shared, the repository model of sharing is most commonly used.

CASE toolset architecture



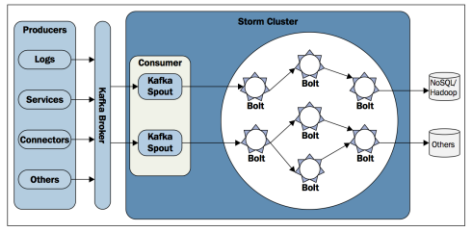
Repository model characteristics

- Advantages
  - Efficient way to share large amounts of data;
  - Sub-systems need not be concerned with how data is produced;
  - Centralised management e.g. backup, security, etc.
  - Sharing model is published as the repository schema: ***easy integration***;
- Disadvantages
  - Sub-systems must agree on a repository data model. Inevitably a compromise;
  - Data evolution is difficult and expensive: ***e.g., changing the data model is expensive or even impossible***;
  - No scope for specific management policies: ***sub-systems may have differente requirements for security, backup, etc. ...***
  - Difficult to distribute efficiently.

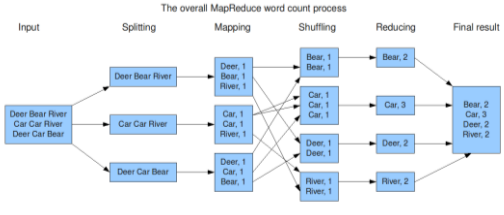
Pipe and Filter model

- This approach lays emphasis on the incremental transformation of data by successive component.
- The connections between modules are data stream which is first-in/first-out buffer that can be stream of bytes, characters, or any other type of such kind
- A filter is an independent data stream transformer or stream transducers. It transforms the data of the input data stream, processes it, and writes the transformed data stream over a pipe for the next filter to process.
- Pipes are stateless and they carry binary or character stream which exist between two filters. It can move a data stream from one filter to another

Apache Storm Architecture



Map Reduce Architecture



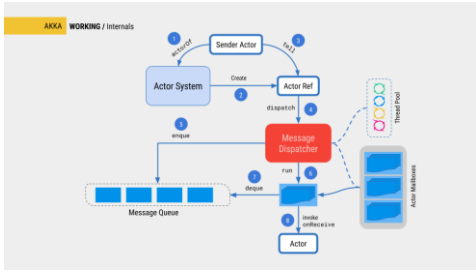
Pipe and Filter model characteristics

- Advantages
  - Provides concurrency and high throughput for excessive data processing.
  - Provides reusability and simplifies system maintenance.
  - Provides modifiability and low coupling between filters.
  - Provides simplicity by offering clear divisions between any two filters connected by pipe.
  - Provides flexibility by supporting both sequential and parallel execution.
- Disadvantages
  - Not suitable for dynamic interactions.
  - A low common denominator is needed for transmission of data in ASCII formats.
  - Overhead of data transformation between filters.
  - Does not provide a way for filters to cooperatively interact to solve a problem.
  - Difficult to configure this architecture dynamically.

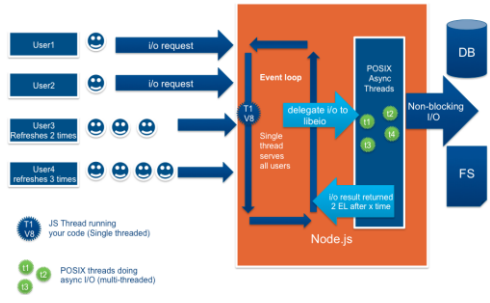
Event-driven / Asynchronous model

- The event-driven architecture is made up of highly decoupled, single-purpose event processing components that asynchronously receive and process events.
- The event-driven architecture pattern consists of two main topologies, the mediator and the broker.
  - The mediator topology is commonly used when you need to orchestrate multiple steps within an event through a central mediator
  - The broker topology is used when you want to chain events together without the use of a central mediator.

Akka architecture



Node.js architecture



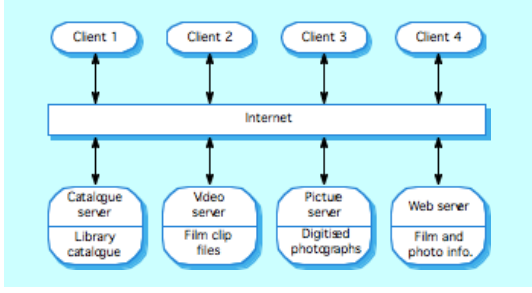
Event-driven model characteristics

- Advantages
  - Relatively easy to deploy due to the decoupled nature of the event-processor components
  - The pattern achieves high performance through its asynchronous capabilities
  - Scalability is naturally achieved in this pattern through highly independent and decoupled event processors.
- Disadvantages
  - While individual unit testing is not overly difficult, it does require some sort of specialized testing client or testing tool to generate events.
  - Development can be somewhat complicated due to the asynchronous nature of the pattern as well as contract creation and the need for more advanced error handling

Client-server model

- Distributed system model which shows how data and processing is distributed across a range of components.
- Set of stand-alone ***servers*** which provide specific services such as printing, data management, etc.
- Set of ***clients*** which call on these services.
- ***Network*** which allows clients to access servers.

Film and picture library



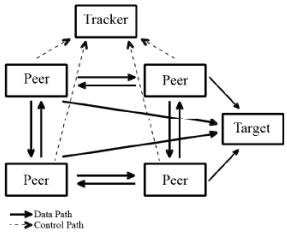
Client-server characteristics

- Advantages
  - Distribution of data is straightforward;
  - Makes effective use of networked systems. May require cheaper hardware;
  - Easy to add new servers or upgrade existing servers.
- Disadvantages
  - No shared data model so sub-systems use different data organisation. Data interchange may be inefficient;
  - Redundant management in each server;
  - No central register of names and services - it may be hard to find out what servers and services are available.

Peer-to-peer model

- Peers may function both as a client, requesting services from other peers, and as a server, providing services to other peers.
- Peers acting as a server may inform peers acting as a client of certain events. Multiple clients may have to be informed, for instance using an event-bus.
- When a peer receives a query, first the query is evaluated against its local data collection and thereafter, if necessary other peers are contacted through its neighbors. Query messages are forwarded only between open connections, i.e., neighboring peers.

Bittorrent



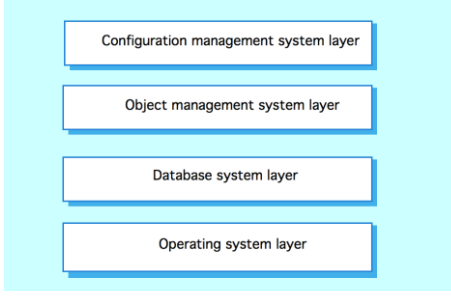
Peer to peer characteristics

- Advantages
  - Nodes may use the capacity of the whole, while bringing in only their own capacity.
  - Administrative overhead is low, because peer-to-peer networks are self-organizing
  - Scalable, and resilient to failure of individual peers.
  - The configuration of a system may change dynamically: peers may come and go while the system is running
- Disadvantages
  - There is no guarantee about quality of service, as nodes cooperate voluntarily.
  - Security is difficult to guarantee
  - Performance may be low when there are few nodes

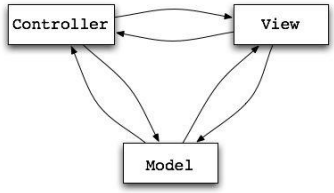
Layered model

- Used to model the interfacing of sub-systems.
- Organises the system into a set of layers (or abstract machines) each of which provide a set of services.
- Supports the incremental development of sub-systems in different layers. When a layer interface changes, only the adjacent layer is affected.
- However, often artificial to structure systems in this way.

Version management system

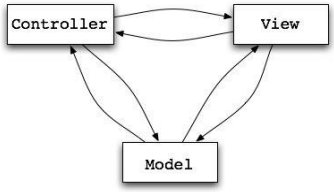


Model View Controller

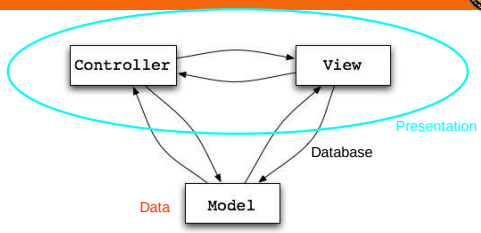


- Architectural Pattern from Smalltalk (1979)
- Decouples data and presentation
- Eases the development

Model View Controller

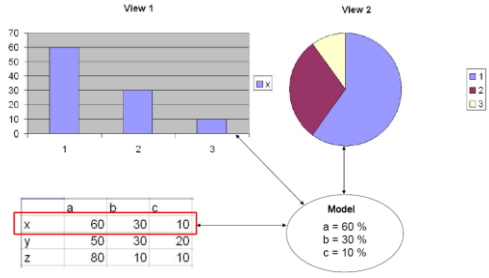


- Tier 1: View (Client)
- Tier 2: Controller (Server)
- Tier 3: Model (Database)



- Presentation:
  - View is the user interface (e.g. button)
  - Controller is the code (e.g. callback for button)
- Data:
  - Model is the database

Example MVC

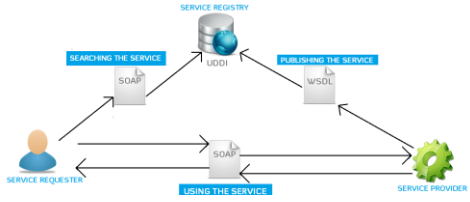


SOA model

A service-oriented architecture provides the following features

- **Distributed Deployment:** Expose enterprise data and business logic as loosely, coupled, discoverable, structured, standard-based, coarse-grained, stateless units of functionality called services.
- **Composability:** Assemble new processes from existing services that are exposed at a desired granularity through well defined, published, and standard complaint interfaces.
- **Interoperability:** Share capabilities and reuse shared services across a network irrespective of underlying protocols or implementation technology.
- **Reusability**: Choose a service provider and access to existing resources exposed as services.

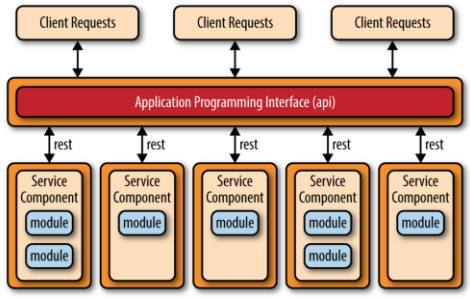
Web Services



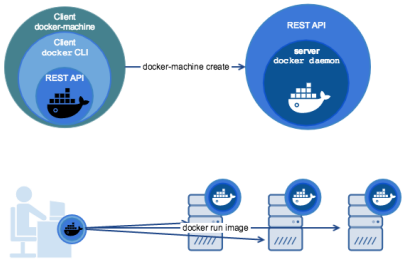
Microservice model

- Each component of the microservices architecture is deployed as a separate unit
- Service components contain one or more modules (e.g., Java classes) that represent either a single-purpose function (e.g., providing the weather for a specific city or town) or an independent portion of a large business application (e.g., stock trade placement or determining auto-insurance rates)
- All the components within the architecture are fully decoupled from one other and accessed through some sort of remote access protocol (e.g. REST)

REST Architecture



REST through Docker



Microservice characteristics

- Advantages
  - Overall this pattern is relatively easy to deploy due to the decoupled nature of the event-processor components.
  - Due to the separation and isolation of business functionality into independent applications, testing can be scoped, allowing for more targeted testing efforts
  - Because the application is split into separately deployed units, each service component can be individually scaled, allowing for fine-tuned scaling of the application
- Disadvantages
  - This pattern does not naturally lend itself to high-performance applications due to the distributed nature of the microservices architecture pattern

Wrap-up

This week we present

- Architectural Models
  - Different architectural models may be produced during the design process
  - Each model presents different perspectives on the architecture

Next Week

- We will be covering *Software Design Engineering*!!!