Lecture 8

Architectural Design

Outline

- Why Architectural Design?
- What is architecture?
- What makes a good architecture?
- Architectural patterns
- Modularity

Architecture

Text Book, **Software Architecture in Practice**, Chapter 1, 2, 7 and 10

Architectural Design vs Detailed Design

Why Architectural Design?

- Architectural design:
  - Identifies the major components and the relationships between them
  - Supports large-scale reuse
  - Supports system evolution
- Why Architectural design:
  - Understanding and management of commercial power distribution (C.P.D.) system requires understanding of its major components and how they interact
  - Understanding of the C.P.D. system architecture allows us to identify and use the types of work assignments for programming teams
  - Major categories of software architecture
  - Pipe and filter
  - Client/server
  - Subsystem and Controller

Architecture

Module Structures:

- Module structures are concerned with properties mostly implementation units, which we call modules.
- Modules are units of implementation, computational responsibility, and are the basis of work assignments for programming teams.
- The major properties, many elements (modules) are subdivided into smaller units (submodules).
- Modules are elements such as classes, layers, functionality, ...

Architecture

Component and connector structures:

- Module structures allow us to answer questions about what software will be built and what software will be modified.
- Component and connector structures is a complex structure and it is used to answer questions about how the system will run.
- Major properties, many elements are connected by many connections, such as call/return, process synchronization, data flow, ...
- Other modules are related to other modules by dependencies, inheritance, or special-purpose, architecture-dependent.

Architecture

Component and connector structures are known as answer questions about the system's runtime. Such questions are:

- What are the major units of work while the system is running?
- What are the major units of data while the system is running?
- What parts of the system can run in parallel?
- What is the relationship between Vistage or Calculator and C.P.D. Task?

Architecture

Allocation Structures:

- Allocation structures describe the mapping from software structures to hardware structures.
- Allocation structures:
  - Deployment structure
  - Implementation structure
  - Work assignment structure

For example:

- Deployment structure concerns in the structure for representation, reliability, and security.
- Work assignment structure is used to relate to assure

Architecture

Allocation Structures:

- Allocation structures describe the mapping from software structures to hardware structures.
- Allocation structures:
  - Deployment structure
  - Implementation structure
  - Work assignment structure

For example:

- Deployment structure concerns in the structure for representation, reliability, and security.
- Work assignment structure is used to relate to assure

Good Architecture

Structural rules

Architecture should follow rules:

- Structural rules are necessary to make a system understandable by such a structure, and to make it easy to modify and maintain.
- Structural rules are necessary to make a system understandable by such a structure.
- Structural rules prevent data from be separate from modules into components.
- Structural rules are necessary to make a system understandable by such a structure, and to make it easy to modify and maintain.
- Structural rules are necessary to make a system understandable by such a structure, and to make it easy to modify and maintain.

Structural rules

- Structural rules are necessary to make a system understandable by such a structure, and to make it easy to modify and maintain.
- Structural rules are necessary to make a system understandable by such a structure, and to make it easy to modify and maintain.
- Structural rules are necessary to make a system understandable by such a structure, and to make it easy to modify and maintain.

Architectural patterns

Architectural Patterns

- Architectural patterns are guidelines, a relationship between components, and a set of constraints on how components can be used in a system.
- Architectural patterns, approaches, governance that refer to the structure of a system.
- Architectural patterns are:
  - Layered Patterns
  - Client-server Patterns
  - Pipe and filter Patterns
  - Hierarchical control Patterns
  - Blackboard Patterns
  - Interpreter Patterns
  - Process control Patterns
  - Rule based Patterns
  - Model view controller Patterns
  - Object oriented Patterns
  - Service oriented Patterns
  - Publish subscribe Patterns
  - Asynchronous messaging Patterns
  - Event driven Patterns
  - Shared memory Patterns
  - Tuple space Patterns
  - Pipes and filters Patterns
  - Blackboard Patterns
  - Interpreter Patterns

Architectural Patterns

Patterns can be categorized by the dominant type of elements that they show.

- Module Patterns
  - Layered Patterns
- Component and connector (C&C) patterns:
  - Client-Server Patterns
  - Shared Data Patterns
  - Publish-Subscribe Patterns
  - Peer-to-Peer Patterns
  - Pipe-and-Filter Patterns
  - Model-View-Controller Patterns
  - Service-Oriented Architecture Patterns
  - Map-Reduce Patterns
- Allocation Patterns
  - Deployment Patterns
  - Work Assignment Patterns

Module Patterns – Layered Pattern

Context:

- Complex software systems requires the ability to develop and maintain portions of the system independently and to understand the system in terms of its components without needing to understand all of the system's components. In large systems, it is not feasible, and in many cases, it is not possible to understand the entire system.

Problem:

- Minimizing the effort needed to make changes to the system, supporting portability, and hiding the details of how to access services from the rest of the system.

Solution:

- The software system is broken into a number of layers, each of which has a well-defined set of responsibilities. The design decision which assigns responsibilities to each layer is the most critical and difficult part of this pattern.

Module Patterns – Layered Pattern

Figure 10 Layers Uses Relation for Layered

Component and connector (C&C) Patterns – Client-Server Pattern

Context: There are several clients and servers that need to communicate to accomplish work of the system, and the number of clients and servers may change at runtime.

Problem: Supporting scaling of shared resources and services, so that the system can continue to run with acceptable performance and availability when the number of clients and servers increases, and supporting the ability to move resources and services, while distributing the execution of services across the network, so that the system can be deployed in a way that meets performance requirements.

Solution: Clients interact with servers to request services, and servers may be distributed across the network so that they can be located on separate machines from the clients.

Component and connector (C&C) Patterns – Client-Server Pattern

Consists: Clients and servers of a given type, and a connector that allows clients to find servers and communicate with them.

Relations: Clients are associated with servers, meaning clients rely on servers to provide services, and servers may be associated with clients.

Attributes: Clients are independent from number of server components, and servers are independent from number of client components. Servers may be stateless or stateful, and may be replicated. Clients may be stateless or stateful. Clients and servers may be deployed on different machines or share a single machine.

Topology: Clients and servers may be deployed on a single machine or on many machines, connected by a variety of networks of different speeds and bandwidths.

Variants: Three-tier client-server, n-tier client-server, peer-to-peer, client-server with proxy, client-server with broker, client-server with replication, mobile code, and client-server with heartbeat.

Reasoning: Supports modifiability of the number of clients and servers, supports performance by allowing servers to be replicated on multiple machines, supports availability by allowing servers to be replicated on multiple machines.

Component and connector (C&C) Patterns – Client-Server Pattern

The Client and Server Components in a Client-Server Pattern

Component and connector (C&C) Patterns – Broker

Context: Many services are distributed between different servers, and clients need to be able to find these services. Clients should be able to access services without knowing where they are located, and should be able to access services from different servers without having to know how to communicate with each server individually.

Problem: Time and effort required to develop and maintain the client's access to the servers, and the ability to add, remove, and update the servers available to the clients in a way that minimizes the impact on the clients.

Solution: The broker pattern separates out the servers's location from the client's access to the servers. Clients and servers connect to the broker, and the broker forwards requests from the clients to the servers, and returns the results of the services back to the clients. The broker may also be responsible for other tasks, such as authorization, but this is not an essential part of the pattern.

Component and connector (C&C) Patterns – Broker

The Broker Component in a Broker Pattern

Component and connector (C&C) Patterns – Broker

Consists: The broker component, client proxies, server proxies, and a name-to-address table. The proxies are only found in broker-based systems, and are optional components.

Relations: Clients are associated with servers, meaning clients rely on servers to provide services, and servers may be associated with clients. The broker is associated with clients and servers, and the broker's name-to-address table associates the broker with the servers.

Attributes: Clients are independent from number of server components, and servers are independent from number of client components. Servers may be stateless or stateful, and may be replicated. Clients may be stateless or stateful. Clients and servers may be deployed on different machines or share a single machine.

Topology: Clients and servers may be deployed on a single machine or on many machines, connected by a variety of networks of different speeds and bandwidths.

Variants: Broker with client proxy, broker with server proxy, broker with client proxy and server proxy, broker with name-to-address table, broker with client-server heartbeat, broker with broker-broker heartbeat, broker with broker-broker replication, broker with broker-broker hierarchy, broker with broker-broker peer-to-peer, broker with broker-broker publish-subscribe, broker with broker-broker shared memory, broker with broker-broker tuple space, broker with broker-broker pipes and filters, broker with broker-broker blackboard, broker with broker-broker interpreter.

Reasoning: Supports modifiability of the number of clients and servers, supports performance by allowing servers to be replicated on multiple machines, supports availability by allowing servers to be replicated on multiple machines.

Component and connector (C&C) Patterns – Broker Pattern

Broker Pattern Components and Connectors

Component and connector (C&C) Patterns – Model-View-Controller Pattern

Context: Large interactive software is typically the most frequently changed part of a system, and it is important to be able to develop and test the interface independently from the rest of the system.

Problem: Minimizing the effort needed to make changes to the interface, supporting portability of the interface, and supporting the ability to develop and test the interface independently from the rest of the system.

Solution: The model-view-controller pattern separates the model and the view into separate components, and the controller mediates between the model and the view.

Component and connector (C&C) Patterns – Model-View-Controller Pattern

Model-View-Controller Components and Connectors

Component and connector (C&C) Patterns – Model-View-Controller Pattern

Consists: The model, one or more views, and the controller.

Relations: The model is associated with the views and the controller, the views are associated with the controller, and the controller is associated with the model and the views.

Attributes: The model is independent from the number of views and the controller, the views are dependent on the model and the controller, and the controller is dependent on the model and the views.

Topology: The model, views, and controller may be deployed on a single machine or on many machines, connected by a variety of networks of different speeds and bandwidths.

Variants: Model-view-controller with client-server, model-view-controller with broker, model-view-controller with publish-subscribe, model-view-controller with shared memory, model-view-controller with tuple space, model-view-controller with pipes and filters, model-view-controller with blackboard, model-view-controller with interpreter.

Reasoning: Supports modifiability of the number of views, supports portability of the interface, supports the ability to develop and test the interface independently from the rest of the system.

Modifiability

Modifiability

Change Factors:

- Adding a new feature
- Modifying an existing feature
- Deleting a feature
- Porting software to a new platform
- Restructure system to improve future maintenance
- Integrate system with other systems
- Change the system to run on new platforms
- Change the system to run on new platforms
- Change the system to run on new platforms