Architecture Styles

Arnav Gupta

December 10, 2024

Contents

1 Architectures

No good way to characterize architectural structures, but best to make common use of architectural principles when designing software. Principles represent rules of thumb or emerging patterns, or industry standards.

2 Architecture Styles of Software Systems

Architectural Style: family of systems in terms of a pattern of structural organization

Architectural style determines:

- **vocabulary** of components and connectors that can be used in instances of that style
- **constraints** on how they can be combined
  - constraints of topology of descriptions or execution semantics

To determine an architectural style, determine:

- structural pattern (components, connectors, constraints)
- underlying computational model
- essential invariants
- examples of use
- advantages and disadvantages of use
- common specializations of style

System architecture is collection of computational components and description of interactions between components (connectors).

Software architectures are represented as graphs where nodes represent components: procedures, modules, processes, tools, and DBs; and edges represent connectors: procedure calls, event broadcasts. DB queries, and pipes.

2.1 Repository Style

Suitable for applications in which central issue is establishing, augmenting, and maintaining a complex central body of info.

Typically, info must be manipulated in a variety of ways, so long-term persistence is required.

Components:

- central data structure representing current system state
- collection of independent components that operate on central data structure

Connectors:

- procedure calls or direct memory access

Changes to data structure trigger computation.

Data structure in memory and on disk.

Concurrent computations and data accesses.

Multiple clients collaborate, accessing data, processing it, and updating it for others to process further.

Examples include programming environments, info systems, graphical editors, and AI knowledge bases.

Advantages:

- efficient way to store large amounts of data
- sharing model published as the repository schema
- centralized management: backup, security, concurrency control

Disadvantages:

- must agree on data model beforehand
- difficult to distribute data
- data evolution is expensive

2.2 Pipe and Filter Style

Suitable for applications that require defined series of independent computations to be performed on data.

Component reads streams of data as input and produces streams of data as output.

Components:

- **filters**: apply local transformations to input streams and compute incrementally so output begins before all input consumed

Connectors:

- **pipes**: conduits for streams, transmitting outputs of 1 filter to inputs of another filter

Filters do not share state with other filters. Filters do not know identity of their upstream or downstream filters.

Advantages:

- easy to understand overall input/output behaviour of system as composition of behaviours of individual filters
- support reuse since any 2 filters can be hooked together provided they agree on data transmitted between them
- easy to maintain and enhance (add new filters and replace old filters easily)
- permit specialized analysis (throughput and deadlock analysis)
- support concurrent execution

Disadvantages:

- bad for interactive systems due to transformational character
- excessive parsing and unparsing leads to loss of performance and increased complexity in writing filters

2.2.1 Specializations

Pipelines: restricts topologies to linear sequences of filters

Batch Sequential: degenerate case of pipeline architecture where each filter processes all input data before producing any output

Compilers started with sequential process and moved towards repository as intermediate representation became more important.

2.3 Object-Oriented

Suitable for applications in which a central issue is identifying and protecting related bodies of info (data).

Data representations and associated operations are encapsulated in abstract data types.

Components: objects

Connectors: function and procedure invocations (methods)

Objects are responsible for preserving integrity (invariants) of data representation.

Data representation is hidden from other objects.

Specializations include distributed objects and objects with multiple interfaces.

Advantages:

- object hides data representation from clients, so possible to change implementation without affecting clients
- can design systems as collections of autonomous interacting agents

Disadvantages:

- for 1 object to interact with another object (via method invocation), first object must know identity of second object
  - when identity changes, must modify all objects that invoke it
- objects cause side effect problems, especially when object used by multiple different objects

2.4 Layered

Suitable for applications that involve distinct classes of services that can be organized hierarchically.

Each layer provides service to the layer above and serves as client to layer below.

Only carefully selected procedures from inner layers are made available (exported) to adjacent outer layer.

Components: collections of procedures

Connectors: procedure calls under restricted visibility

Tier is physical partition of system across servers, layer is logical partition of related functionality, typically at different layers of abstraction.

Advantages:

- design based on increasing levels of abstraction
- enhancement easy since changes to one layer affects at most 2 other layers
- allows reuse since different implementations of same layer can be used interchangeably

Disadvantages:

- not all systems are easily structured in layered fashion
- performance requirements may force coupling of high-level functions to lower-level implementations

2.4.1 Specializations

Exceptions are made to permit non-adjacent layers to communicate directly, done for efficiency reasons.

2.5 Interpreter

Suitable for applications in which the most appropriate language or machine.

Components: include one state machine for the execution engine and 3 memories:

- current state of the execution engine
- program being interpreted
- current state of the program being interpreted

Connectors:

- procedure calls
- direct memory accesses

Advantages:

- simulation of non-implemented hardware
- facilitates portability of application or languages across a variety of platforms

Disadvantages:

- defining, implementing, and testing interpreter components is non-trivial
- extra level of indirection slows down execution:

2.6 Process-Control

Suitable for applications whose purpose is to maintain specified properties of the outputs of the process sufficiently near given reference values.

Components:

- process definition includes mechanisms for manipulating process variables
- control algorithm for deciding how to manipulate process variables

Connectors: data flow relations for

- **process variables**: controlled variables whose value the system is intended to control
  - input variable that measures process input
  - manipulated variable whose value can be changed by the controller
- **set point**: desired value for a controlled variable
- **sensors**: obtain values of process variables to control

2.6.1 Feedback Control System

Controlled variable is measured and result is used to manipulate 1+ process variables.

2.6.2 Open-Loop Control System

Given pure input materials, fully defined process, and completely repeatable process, process can run without surveillance (no feedback).

Info about process variables is not used to adjust system.

2.7 Client-Server

Suitable for applications that involve distributed data and processing across a range of components.

Components:

- **server**: standalone component that provides specific services
- **client**: component that call on the services provided by servers

Connector: network, which allows clients to access remote servers

Advantages:

- straightforward distribution of data
- transparency of location
- mix/match heterogeneous platforms
- easy to add new servers or upgrade existing servers

Disadvantages:

- system performance depends on network performance
- tricky to design and implement client/server systems
- hard to find out what services are available

2.8 Peer-to-Peer

Suitable for applications that partition tasks/workloads between peers without centralized control.

Components:

- peers: standalone components that provide specific services to each other (act as both client and server)

Connector: some kind of network that allows peers to access each other directly without a central server

Connections can be transient/ad hoc.

Advantages:

- no single point of failure (since no central server)
- resilient against network issues or failing peer by establishing new links to other peers
- scalable (since no central server)
- more privacy (in theory)
- easy to add new peers or upgrade existing peers

Disadvantages:

- cliques of peers could be disconnected from each other if no overlapping peer
- requires continuous exchange of lists of known peers between each peer
- could require central directory server to be able to locate each peer
- no guaranteed response from peers (could be down, delayed, hostile)