The major forms of architectural style

1 Call and return

Characterised by:

- Order of computation (sequencing of control)
- Only a single thread of control (no concurrency)
- Structures organised around computational tasks

Type of reasoning:

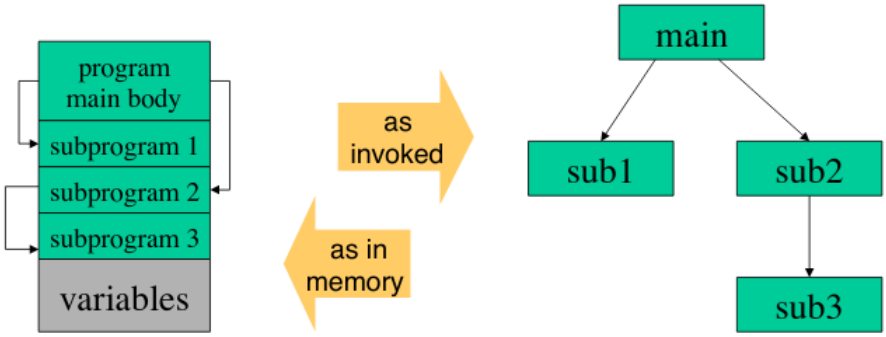
- Hierarchical, organised around control being passed from ”higher” to ”lower” items
- Task performed by a set of sub-tasks
- Explicit control linkages between the subtasks and the algorithm of the program to determine calling order

1.1 Example 1

Main program/sub-programs:

- Components are subprograms
- Connectors are the invocation links (including parameters)

This is the form that is embodied in most non-OO imperative programming languages

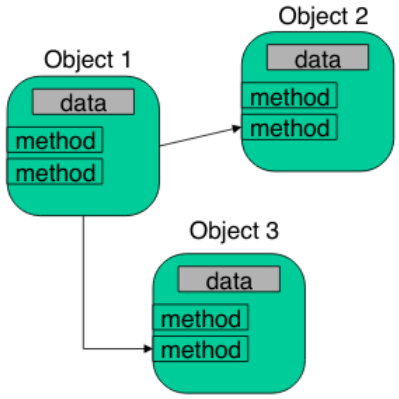


1.2 Example 2

Classical objects - here the methods are part of the objects:

- Components are both the objects and also their methods/data
- Connectors are the method calls and parameters

Supported by object-based and object-oriented programming languages. May use run-time bindings



2 Interacting Processes

Characterised by:

- Communication patterns among independent, usually concurrent, processes
- Independent elements can be objects, lightweight processes, services etc
- Context is more complex as emphasis is upon ”interaction”

Type of reasoning

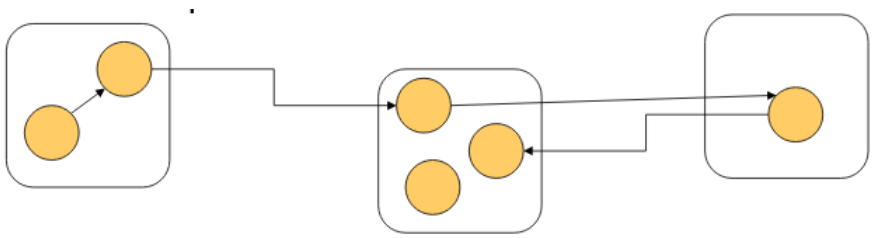
- Non-deterministic - scheduling of system elements is performed by separate, independent computers
- Design based around these independent actions and needs to specifically address any requirements that impose the need for particular sequences or interactions

2.1 Communicating processes

Based around a network of processes running on different machines and linked via Remote Procedure Calls (RPC)

- Components are the processes
- Connectors are the message protocols via RPCs

Binding time can be at each construction or when the system is started. Each system runs in its own address space - no direct access to shared data

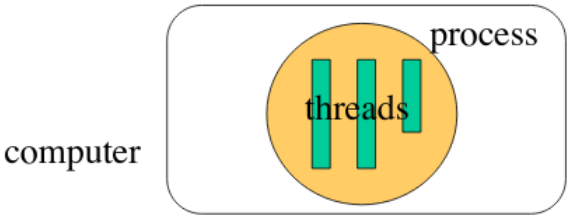


2.2 Lightweight processes (threads)

A thread is a block of code that can be executed independently of other threads within the program

- Components are the threads
- Connectors are provided largely by using shared data

Because threads are part of a single process, they have a shared data space, needing care with synchronisation of access



3 Data-Centred Repository

Characterised by:

- A dominant central data store that is manipulated by independent communications
- Centred around the issues relating to data access
- Data is at the focus

Type of reasoning:

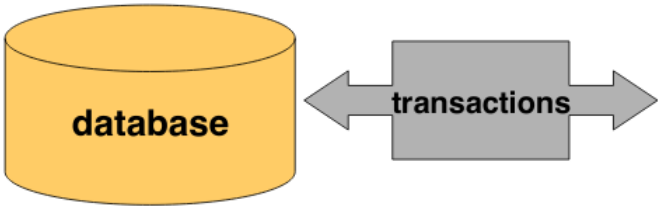
- Can depend on the form of the system
- For databases it is the ACID properties
- Includes modern compilers

3.1 Transactional Database

The classical idea of the data-centred repository

- Components - Memory and computations within the database
- Connectors - Transaction streams (queries)

Style is not concerned with the internal form of the database, only with its role



3.2 Client-Server

A ”distributed” form using a ”thin” client process to interact with the end-users. While the server often incorporates a database, this is not a necessity

- Components - Data managers and computations
- Connectors - Transaction operations with history

The history element is important, systems that lack this are sometimes termed ”naive client-server” forms

4 Data-Sharing

Characterised by:

- Direct sharing of data between the components
- Data may be distributed between many elements, but these share a common form of ”representation” for using and sharing this data

Type of reasoning:

- Based upon the data representation itself and ensuring that this is common to all elements
- Describes both programs, but also documents

4.1 Hypertext

This is a ”document” form rather than a program form, but still an important architectural style since it affects the way that programs are structured in order to use it

- Components - Documents in HTML format
- Connectors - Hyperlinks and internal anchors

Websites can incorporate both static pages and also pages that are generated dynamically, so this example is quite an important one.

5 Mixed styles

An example of a mixed style is a compiler, it started out as pipe and filter, but are now usually more of a data-centred repository.

6 Shaw and Clement's classification

- Context is elaborated by being sub-divided into three aspects (control, data and control/data interaction) which are further sub-divided
- In addition, they consider the form of reasoning that might be used with a particular architectural style

Control issues are classified in terms of:

- Topology (geometric form)
- Synchronicity (interdependence of actions and states)
- Binding time (when is the identify of a partner established)

Data issues are classified using:

- Topology - same concept as for control
- Continuity - e.g. continuous flow of data or sporadic flow
- Mode - whether data is copied, shared etc
- Binding time (as for control)

Control/data interaction captures some further relationships, and is classified in terms of

- Share (are control and data flow topologies isomorphic?)
- Directionality (if the shapes are the same, does control flow in the same direction as data, or in the opposite direction)

7 Architectural Mismatch

Definition: Architectural Mismatch

Mismatched assumptions a reusable part makes about the system it is to be a part of

Basic cause was in the detail (of components, connectors and context). Each element had a slightly different model of how control and event passing was organised, reinforcing the point that all of the different aspects of architectural style are relevant

8 Security

- The way this is handled will depend on architectural style
- Each style will exhibit its own vulnerabilities
  - How does a component check or confirm that is it connected to a valid component
  - How might it be attacked
  - How easily can we achieve ”information hiding” within a particular architectural style
- Increasingly, this adds a new dimension to those that the designer has to think of

9 Choosing a style

Some guidelines on choice of style, where this is not already determined by other factors:

- For strongly sequential tasks: batch sequential or pipe-and-filter
- For tasks which involve transformation on continuous streams of data, use pipe-and-filter
- If a central issue in your system is understanding the data used in the application, consider a repository style
- For flexibility, loose coupling between tasks and reactive tasks, consider interacting processes