Architectural Structures

CSE-474 Software Design & Architecture

Dr. Awais Majeed

February 21, 2011

MCS

Outline

Software Architecture

- SA as a Transferable, re-usable model

Architectural Structures

- Basics
- Module Structures
- Component & Connector Structures
- Allocation

Conclusion

Importance of Software Architecture

Communication among stakeholders

- SA presents a common abstraction of a system
- Customers are concerned that Software is reliable and available Architecture can be implemented on schedule and within budget SA allows teams to work independently

MCS

SA for early design decisions I

Implementation constraints

- Implementation must follow the structural design decisions
- Describe prescribe elements, their interaction and the responsibilities
- Resource allocations as constraints on implementation

Organizational structure

- SA defines the highest level system decomposition
- This is used for the work breakdown
- Planning, budgeting, scheduling, teams, communication channels, configuration management etc.

SA for early design decisions II

SA as system quality enabler

- Architecture defines whether a system will inhibit the desired/required qualities or not
- High performance – manage time base behavior of the elements Inter-element communication ?
  - Modifiability – responsibilities to elements
  - Security – inter-element communication and access control
  - Scalability – localization of resources
  - Incremental release – manage inter-component usage
  - Re-usability – restrict inter-element coupling

SA for early design decisions III

Manage Change

- 80% of the cost is after initial deployment
- Change can be partitioned into 3 groups:
    - Local: modify a single element
    - Non-local: multi-element change
    - Architectural: change the way elements interact with each other
- Deep understanding of the relationships, performance and behavior of the system is required to decide on such changes

SA for early design decisions IV

Evolutionary prototyping

- An executable system in the early DLC

Cost and schedule estimation

- Team/work distribution more clear
- Identify challenging parts
- Identify required resources/competencies

Software Product Lines

SPL is a set of software-intensive systems sharing a common, managed set of features that satisfy the specific needs of a particular market segment. These are developed from a common set of core assets in a prescribed way

- Architecture: important asset
- Architecture defines what is fixed for all members of a product line/family and what is variable
- Architecture a core asset of the developing organization

Large, externally developed elements

- Architecture based development focuses on composing or assembling of independent elements
- Architecture defines what type of elements can be incorporated into the systems along with any constraints (protocols, data types, procedures, interaction with external/internal environment etc.)
- Interchangeability – challenging in software systems due to architectural mismatch

Restrict design alternatives

- Restrict to a limited no. of choice for program cooperation and interaction
  - Minimise design complexity
- Architectural pattern should help in arbitration of confliction design constraints and improve the implementation of the resulting design solution

Structure and View

- A **Structure** is the set of elements itself, as they exist in software or hardware.
- A **View** is a representation of a coherent set of architectural elements as written by and read by system stakeholders
  - A module structure is the set of the system's module and their organisation
  - A module view is the representation of that structure as documented by some system stakeholder
- Both terms are used interchangeably

Types of Architectural Structures I

Module structure

- The elements are modules – units of implementation
- Assigned areas of functional responsibilities
    - What is the primary functional responsibility assigned to each module?
    - What other software elements is a module allowed to use?

Types of Architectural Structures II

Component-and-Connector structure

- Elements are runtime components (which are the principal units of computation) and
- Connectors (which are the communication vehicles among components)
- Can answer questions as:
  - What are the major executing components and how do they interact?
  - What are the major shared data stores?
  - Which parts of the system are replicated?
  - How does data progress through the system?

Types of Architectural Structures III

- What parts of the system can run in parallel?
- How can the system's structure change as it executes?

Types of Architectural Structures IV

Allocation structures

- Allocation structures show the relationship between the software elements and the elements in one or more external environments
- Can answer questions like:
  - What processor does each software element execute on?
  - In what files is each element stored during development, testing, and system building?
  - What is the assignment of software elements to development teams?

Module Structure

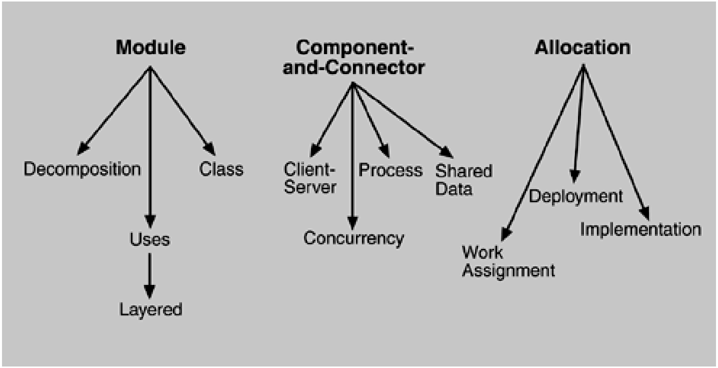


Figure: Module Structures

Decomposition & Uses Structures

Decomposition

- Relationship between modules is “is a sub-module of”
- The decomposition structure enhances system’s modifiability
  - changes fall in a small number of modules
  - The modules have associated products (interface specifications, code, test plans etc)

Uses

- The units are modules or procedures or resources on the interfaces of modules.
- One unit uses another if the correctness of the first requires the presence of a correct version of the second.
- Extend or abstract (extract) functionality

Layered

- A system of layers emerges when *uses* relationship is controlled in a particular fashion in which a layer is a coherent set of related functionality.
- In strict sense : layer n may only use the services of layer n – *1* .
- Layers are often designed as abstractions (virtual machines) that hide implementation specifics below from the layers above – ***portability***
  Think of an example?

Class or Generalisation

- The module units in this structure are called classes
- The relation is **"inherits-from"** or **"is-an-instance-of"**
- This view supports reasoning about collections of similar behavior or capability

Concurrency & Repository

Concurrency

- A structure which allows the architect to determine opportunities for parallelism and the locations where resource contention may occur
- The units are components and the connectors are "logical threads."
- A logical thread is a sequence of computation that can be allocated to a separate physical thread later in the design process.

Repository or Shared Data

- These are the structures that create, store, and access persistent data

Client Server

- A structure of components and connectors for a system built as a group of cooperating clients and servers
- The components are the clients and servers
- The connectors are protocols and messages they share to carry out the system's work
- Useful for:
  - separation of concerns
  - physical distribution, and
  - load balancing (supporting runtime performance)

MCS

Deployment

- The deployment structure shows how software is assigned to hardware-processing and communication elements
- The elements are software (usually a process from a component-and-connector view), hardware entities (processors), and communication pathways
- Relations are "allocated-to"
  - which physical units the software elements reside
  - "migrates-to" if the allocation is dynamic

Work Assignment

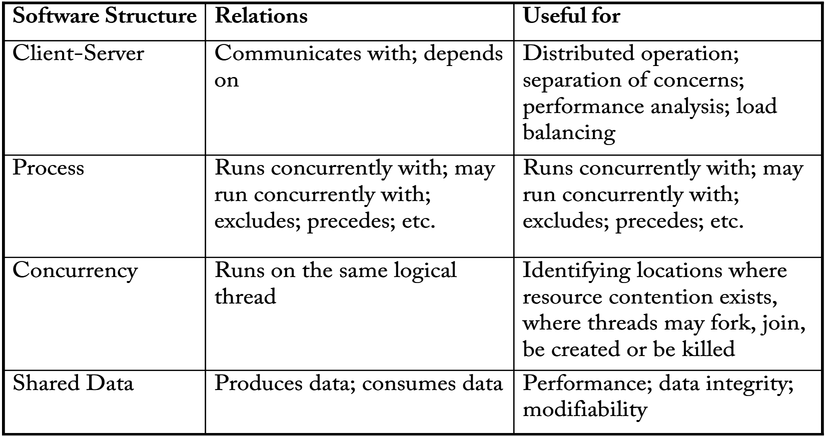
- This structure assigns responsibility for implementing and integrating the modules to the appropriate development teams
- Work assignment structure has architectural as well as management implications

Conclusion-I

Software Structure	Relations	Useful for
Decomposition	Is a submodule of; shares secret with	Resource allocation and project structuring and planning; information hiding, encapsulation; configuration control
Uses	Requires the correct presence of	Engineering subsets; engineering extensions
Layered	Requires the correct presence of; uses the services of; provides abstraction to	Incremental development; implementing systems on top of "virtual machines" portability
Class	Is an instance of; shares access methods of	In object-oriented design systems, producing rapid almost-alike implementations from a common template

Figure 2.1 (a): Module Structures

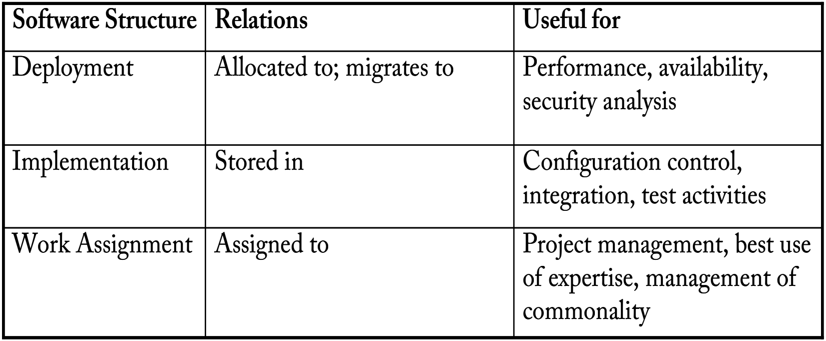
Conclusion-II



Software Structure	Relations	Useful for
Client-Server	Communicates with; depends on	Distributed operation; separation of concerns; performance analysis; load balancing
Process	Runs concurrently with; may run concurrently with; excludes; precedes; etc.	Runs concurrently with; may run concurrently with; excludes; precedes; etc.
Concurrency	Runs on the same logical thread	Identifying locations where resource contention exists, where threads may fork, join, be created or be killed
Shared Data	Produces data; consumes data	Performance; data integrity; modifiability

Figure 2.1 (b): Component & Connector Structures

Conclusion-III



Software Structure	Relations	Useful for
Deployment	Allocated to; migrates to	Performance, availability, security analysis
Implementation	Stored in	Configuration control, integration, test activities
Work Assignment	Assigned to	Project management, best use of expertise, management of commonality

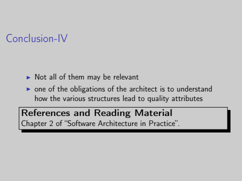
Figure 2.1 (c): Allocation structures

Conclusion-IV

- Not all of them may be relevant
- one of the obligations of the architect is to understand how the various structures lead to quality attributes

References and Reading Material

Chapter 2 of “Software Architecture in Practice”.



Conclusion-IV

- Not all of them may be relevant
- one of the obligations of the architect is to understand how the various structures lead to quality attributes

References and Reading Material
Chapter 2 of "Software Architecture in Practice".

1. Depending upon the system, there might be more than one structures involved.