Architectural Design

- Establishing the overall structure of a software system

Topics covered

- **Software architecture**
- Common architecture styles
- Wrap-up

Software architecture

- Garlan and Shaw, 1993:
  - ...beyond the algorithms and data structures of the computation; designing and specifying the overall system structure emerges as a new kind of problem. Structural issues include gross organization and global control structure; protocols for communication, synchronization, and data access; assignment of functionality to design elements; physical distribution; composition of design elements; scaling and performance; and selection among design alternatives."
- The design process for identifying the sub-systems making up a system and the framework for sub-system control and communication is *architectural design*

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

Topics covered

- Software architecture
- **Common architecture styles**
- Wrap-up

Architectural Styles

- Informal phrases you may have read:
  - The system is based on the *client-server model* and uses remote procedure calls…..
  - *Abstraction layering* provides the appearance of system uniformity to clients ….
  - We have chosen a *distributed, object-oriented approach* ….
  - The easiest way to make the canonical sequential compiler into a concurrent compiler is to *pipeline* the execution of the compiler phases ….
- We can think of these as examples of *"architectural styles"*

Architectural Styles

- Framework
  - Components - modules
  - Connectors - interactions between modules
  - Constraints - rules for combining components and connectors (topological, execution semantics, etc.)
- Questions about architectural styles
  - What is the structural pattern?
  - What is the underlying computational model?
  - What are the essential invariants of the style?
  - What are some common examples of its use?
  - What are the advantages and disadvantages of using it?
  - What are some common specializations of it?

Example Styles

- Pipes and filters
- Data abstraction and Object-Oriented Organization
- Event-based implicit invocation
- Layered systems
- Repositories
- Interpreters
- Other

Pipes and Filters

- Component
  - has a set of inputs and a set of outputs
  - reads input data stream, transforms it, produces an output stream
  - called a filter (because of the transformation it does)
- Connectors
  - streams connecting output of one component to input of another
- Constraints
  - filters are independent (don't share state)
  - components are anonymous (don't know up- and downstream neighbors)
  - order of processing among components does not affect output correctness
- Example
  - Unix shell scripts. Q: How many computers are on the 128.8.122 subnet?
    - » cat /etc/hosts | grep "128\.8\.122\.[0-9]*" | wc -l

Pipes and Filters

- Specializations
  - pipelines - linear sequence of filters
  - bounded pipes - restrict amount of data on pipe
  - typed pipes - input and output streams have well-defined types
- Advantages
  - system behavior is composable/decomposable
  - supports reuse
  - easy to maintain/enhanced
  - can sometimes be analyzed
  - support concurrency
- Disadvantages
  - lead to batch processing
  - not good for interactive applications
  - tend to force a lowest common denominator on data transmission
    - » causing each component to parse/unparse data

Data Abstraction and Object-Oriented Organization

- Component
  - encapsulated as objects
- Connectors
  - function and procedure invocation
- Constraints
  - objects are manage integrity of data representation
  - representation is hidden from other objects
- Example
  - typical C++ program

Data Abstraction and Object-Oriented Organization

- Specializations
  - active objects
  - multiple interfaces
- Advantages
  - data encapsulation
  - polymorphism
- Disadvantages
  - for one object to interact with another (via procedure call), it must know the identity of that other object
  - When an object changes, other objects that invoke it must be modified
  - contrast with pipe and filter

Event-based Implicit Invocation

- Also called
  - reactive integration or selective broadcast
- Component
  - traditional objects with event sets
  - can broadcast events
  - can register procedures to be called when certain events occur
- Connectors
  - procedure call
  - bindings between events and procedures
- Constraints
  - components don't know who is affected by the events they broadcast

Event-based Implicit Invocation

- Example
  - GUIs
  - spreadsheets
- Advantages
  - strong support for reuse
  - eases system evolution
- Disadvantages
  - components relinquish control over system computation
  - data exchange can be costly
  - difficult to reason about correctness

Layered Systems

- Components
  - traditional objects
  - each object placed in a layer
- Connectors
  - protocols that define how objects interact
- Constraints
  - topological constraints such as limiting interactions to adjacent layers

Layered Systems

- Example
  - Java virtual machine
  - layered communications protocols
  - most operating systems
- Advantages
  - supports design based on levels of abstraction
  - supports enhancement - can add layers without affecting lower-level layers
  - supports reuse - different implementations of a layer should be interchangeable
- Disadvantages
  - not all systems easily structured as layers
  - can be slow if many layers exist

Repositories

- Component
  - a central data structure representing the current state
  - other components that operate on the central data structure
- Connectors
  - can vary significantly among different types of systems
    - » command invocation - e.g., database
    - » state-based triggers - e.g., blackboard
- Constraints
  - Can vary

Repositories

- Example
  - agent-based systems - e.g., some web-crawlers
  - workflow systems
- 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.
- Disadvantages
  - Sub-systems must agree on a repository data model
  - Data evolution is difficult and expensive
  - Difficult to distribute efficiently

Interpreters

- Component
  - interpretation engine
  - psuedocode
  - control state representation for interpretation engine
  - current state representation for psuedocode
- Connectors
  - data accesses
  - read/write
- Constraints
  - defined by language semantics

Interpreters

- Example
  - postscript
- Advantages
  - well-understood technology
  - allows for analysis and optimization
- Disadvantages
  - end

Other

- Distributed processes
  - client-server
  - distributed objects - e.g., Corba
- Main program / subroutine organizations
- State transition systems
- Process control systems - environment, sensors, actuators, control loop

Topics covered

- Software architecture
- Common architecture styles
- **Wrap-up**

Architecture attributes and heuristics

- Performance
  - Localize operations to minimize sub-system communication
- Security
  - Use a layered architecture with critical assets in inner layers
- Safety
  - Isolate safety-critical components
- Availability
  - Include redundant components in the architecture
- Maintainability
  - Use fine-grained, self-contained components

Final points

- An awareness of architectural styles can simplify the problem of defining system architectures
- Most large systems are heterogeneous and do not follow a single architectural style
- A wide variety of architectural styles can be used to tackle programming problems