Introduction to Software Architectures

Nenad Medvidovic

neno@usc.edu
http://sunset.usc.edu/~neno/

CSCI 577A

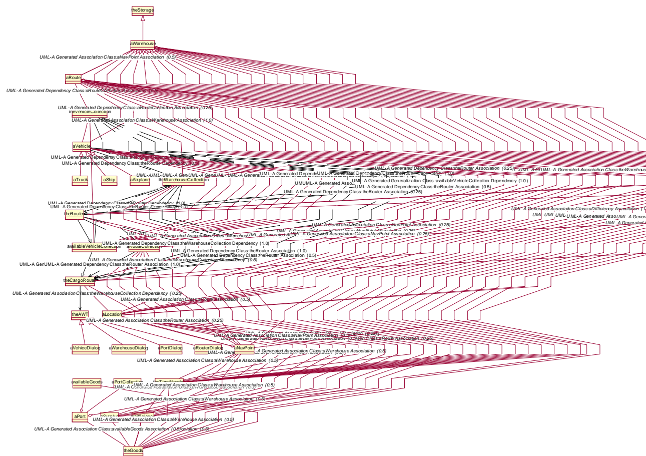
October 20, 2000

Motivation

- Software systems are rapidly and continuously growing in size and complexity
- Techniques and tools for developing and maintaining such systems typically play catch-up
- To deal with this problem, many approaches exploit *abstraction*
  - ignore all but the details of the system most relevant to a task (e.g., developing the user interface or system-level testing)
  - this greatly simplifies the model of the system
  - apply techniques and tools on the simplified model
  - incrementally reintroduce information to complete the “picture”
- Software architecture is such an approach
  - applicable to the task of software design

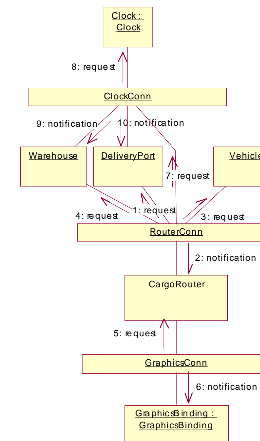
Software is Complex

- A ***simple*** software system



Architecture Removes Complexity

- Architectural view of the system



What is Architecture?

- A high-level model of a *thing*
  - describes critical aspects of the *thing*
  - understandable to many stakeholders
    - architects, engineers, workers, managers, customers
  - allows evaluation of the *thing*'s properties before it is built
  - provides well understood tools and techniques for constructing the *thing* from its blueprint
- Which aspects of a software system are architecturally relevant?
- How should they be represented most effectively to enable stakeholders to understand, reason, and communicate about a system before it is built?
- What tools and techniques are useful for implementing an architecture in a manner that preserves its properties?

What is *Software* Architecture?

- A software system's blueprint
  - its components
  - their interactions
  - their interconnections
- Informal descriptions
  - boxes and lines
  - informal prose
- A shared, semantically rich vocabulary
  - RPC
  - client-server
  - pipe and filter
  - layered
  - distributed
  - OO

From Requirements to Architecture

- Problem Definition ➔ Requirements Specification
  - determine exactly what the customer and user want
  - specifies *what* the software product is to do
- Requirements Specification ➔ Architecture
  - decompose software into modules with interfaces
  - specify high-level behavior, interactions, and non-functional properties
  - maintain a record of design decisions and traceability
  - specifies *how* the software product is to do its tasks

Focus of Software Architectures

- Two primary foci
  - system structure
  - correspondence between requirements and implementation
    - components + rules of composition + rules of behavior
- A framework for understanding system-level concerns
  - global rates of flow
  - communication patterns
  - execution control structure
  - scalability
  - paths of system evolution
  - capacity
  - throughput
  - consistency
  - component compatibility

Definitions of Software Architecture

- Perry and Wolf
  - Software Architecture = { Elements, Form, Rationale }
    - *WHAT* (Elements), *HOW* (Form), *WHY* (Rationale)
- Shaw and Garlan
  - Software architecture [is a level of design that] involves
    - the description of elements from which systems are built,
    - interactions among those elements,
    - patterns that guide their composition,
    - and constraints on these patterns.
- Kruchten
  - Software architecture deals with the design and implementation of the high-level structure of software.
  - Architecture deals with abstraction, decomposition, composition, style, and aesthetics.

Why Software Architecture?

- A key to reducing development costs
  - component-based development philosophy
  - explicit system structure
  - separation of concerns
- A natural evolution of design abstractions
  - structure and interaction details overshadow the choice of algorithms and data structures in large/complex systems
- Benefits of explicit architectures
  - a framework for satisfying requirements
  - technical basis for design
  - managerial basis for cost estimation & process management
  - effective basis for reuse
  - basis for consistency and dependency analysis

Components

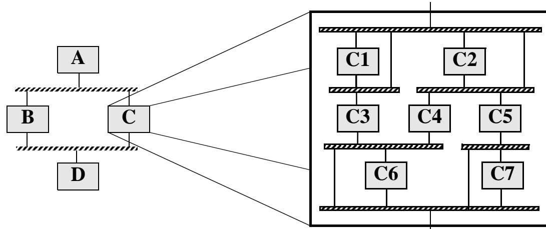
- A ***component*** is a unit of computation or a data store
  - Perry & Wolf's processing and data elements
- Components are loci of computation and state
  - clients
  - servers
  - databases
  - filters
  - layers
  - ADTs
- A component may be simple or composite
  - composite components describe a system

Connectors

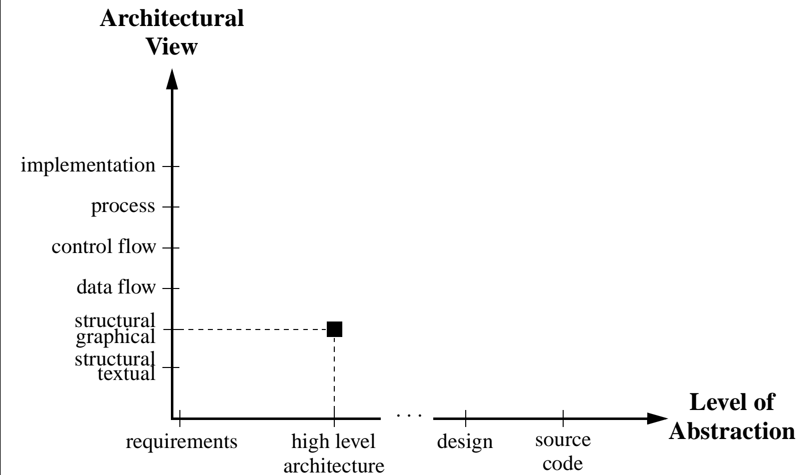
- A ***connector*** is an architectural element that models
  - interactions among components
  - rules that govern those interactions
- Simple interactions
  - procedure calls
  - shared variable access
- Complex and semantically rich interactions
  - client-server protocols
  - database access protocols
  - asynchronous event multicast
  - piped data streams

Configurations/Topologies

- An ***architectural configuration*** or ***topology*** is a connected graph of components and connectors which describes architectural structure.
  - proper connectivity
  - concurrent and distributed properties
  - adherence to design heuristics and style rules
- Composite components are configurations



Architectural Perspectives

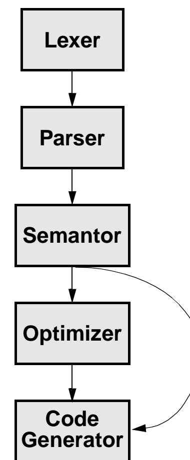


Scope of Software Architectures

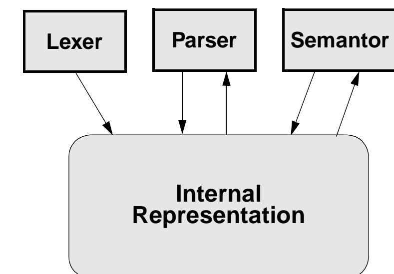
- Every system has an architecture
- Details of the architecture are a reflection of system requirements and trade-offs made to satisfy them
- Possible decision factors
  - performance
  - compatibility with legacy software
  - planning for reuse
  - distribution profile
    - current and future
  - safety, security, fault tolerance
  - evolvability
    - changes to processing algorithms
    - changes to data representation
    - modifications to the structure/functionality

Compiler Architecture

Sequential



Parallel



Compiler Architecture Pros and Cons

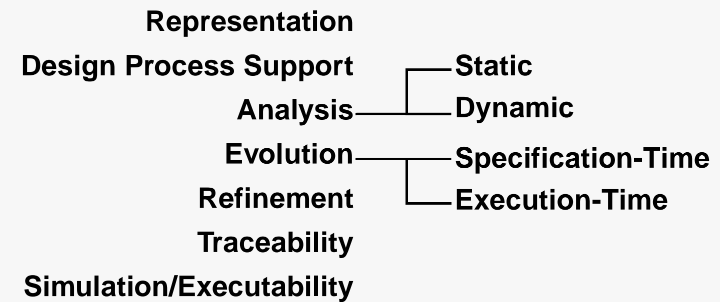
Sequential

+ Conceptual simplicity
+ Architecture reflects control flow
- Performance

Parallel

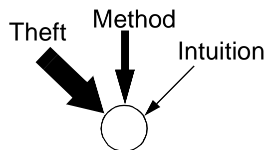
+ Performance
+ Adaptability
- Synchronization
- Coordination
  - analysis and testing

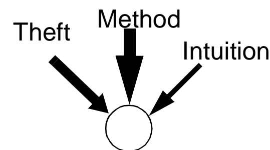
What Are Software Architectures Used for?



Sources of Architecture (1)

- Architecture comes from “black magic, people having ‘architectural visions’”
- 3 + 1 main sources of architecture
  - intuition
  - method
  - theft
  - blind luck
- Their ratio varies according to
  - architects’ experience
  - system’s novelty





Sources of Architecture (2)

- Theft
  - from previous similar systems
  - from literature
- Method
  - systematic and conscious
  - possibly documented
  - architecture is derived from requirements via transformations and heuristics
- Intuition
  - “the ability to conceive without conscious reasoning”
  - increased reliance on intuition increases the risk

Routine Design

- Method is critical
  - an architecture built with 50% theft and 50% intuition is doomed to fail*
- Standardized methods
- Similarity to previous solutions
- Theft
- Cheaper but not optimal
- Can be done by good designers
- Potential pitfall
  - over-reusing

*An architecture built with 50% theft and 50% blind luck makes you world's #1 software development organization

Innovative Design

- Raw invention
- Intuition
- Derivation from abstract principles
- More optimal
- More expensive
- Must be done by great designers
- Potential pitfall
  - reinventing the wheel

Software "Architecting"

- The "architecting" problem lies in
  - *decomposition* of a system into constituent elements
  - *composition* of (existing) elements into a system
- Two idealized approaches
  - top-down
    - decompose the large problem into sub-problems
    - implement or reuse components that solve the sub-problems
  - bottom-up
    - implement new or reuse existing stand-alone components
    - compose (a subset of) the components into a system
- A realistic approach will require both

Issues in Decomposition (1)

- How do we arrive at
  - components
  - connectors
  - their configuration
- What is the adequate component granularity level?
- What constraints on components are imposed by
  - functional requirements
  - non-functional requirements
  - envisioned evolution patterns
  - system scale
  - computing environment
  - customers/users
- What assumptions can components make about one another?

Issues in Decomposition (2)

- How do components interact?
- What are the connectors in the system?
- What is the role of connectors?
  - mediation
  - coordination
  - communication
- What is the nature of connectors?
  - type of interaction
  - degree of concurrency
  - degree of information exchange

Issues in Composition

- Where does one locate existing
  - components
  - connectors
  - configurations
- How do we determine which elements are needed?
  - both at development-time and at reuse-time
- What is the adequate element granularity level?
- How do we ensure effective composition of heterogeneous elements?
- How do we know that we have the needed system?

Example — Software Development Environment