Software Architecture

David S. Rosenblum
ICS 221
Winter 2001

Software Architecture (Perry & Wolf 92)

"*Architecture* is concerned with the selection of architectural elements, their interactions, and the constraints on those elements and their interactions necessary to provide a framework in which to satisfy the requirements and serve as a basis for the design."

"*Design* is concerned with the modularization and detailed interfaces of the design elements, their algorithms and procedures, and the data types needed to support the architecture and to satisfy the requirements."

Software Architecture (Garlan & Shaw 93)

"*Software architecture* is a level of design that goes 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."

Software Architecture (Shaw & Garlan 96)

"The *architecture of a software system* defines that system in terms of computational components and interactions among those components. … In addition to specifying the structure and topology of the system, the architecture shows the correspondence between the requirements and elements of the constructed system, thereby providing some rationale for the design decisions."

Analogies with Civil Architecture

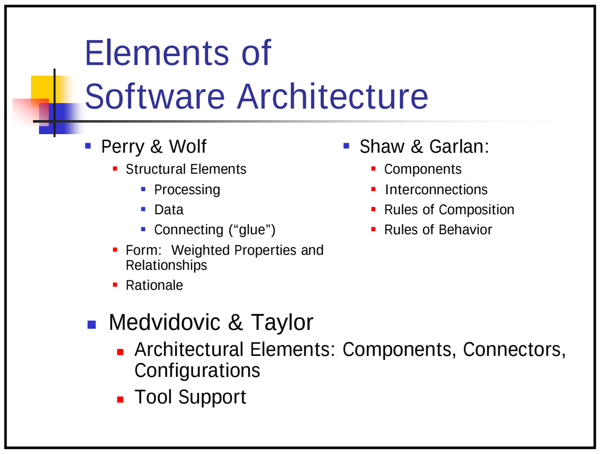
Civil Engineering and Civil Architecture are concerned with the engineering and design of civic structures (roads, buildings, bridges, etc.)

- Multiple views
  - Civil: Artist renderings, elevations, floor plans, blueprints
  - Software: Code, object design, boxes-and-arrows, GUI
- Architectural styles
  - Civil: Classical, Romanesque, Gothic, Renaissance, Baroque, Art Deco
  - Software: Pipe-and-filter, client/server, layered system
- Influence of style on engineering principle
- Influence of style on choice of materials

Differences Between Civil and Software Architecture

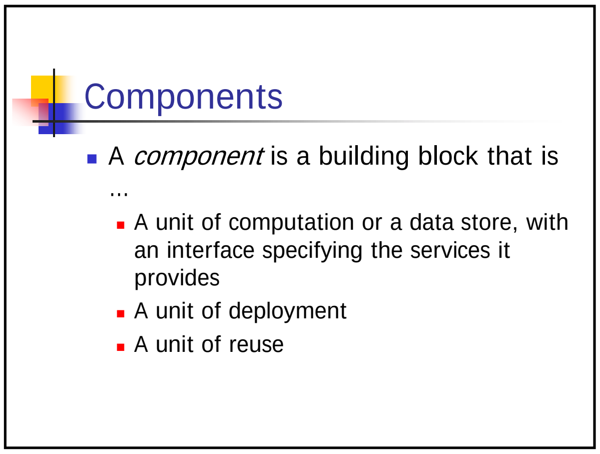
"Software systems are like cathedrals—first we build them and then we pray."
— Sam Redwine

- Physical vs. conceptual
- Static vs. dynamic
- Little evolution vs. frequent evolution
- Different mathematical and scientific bases



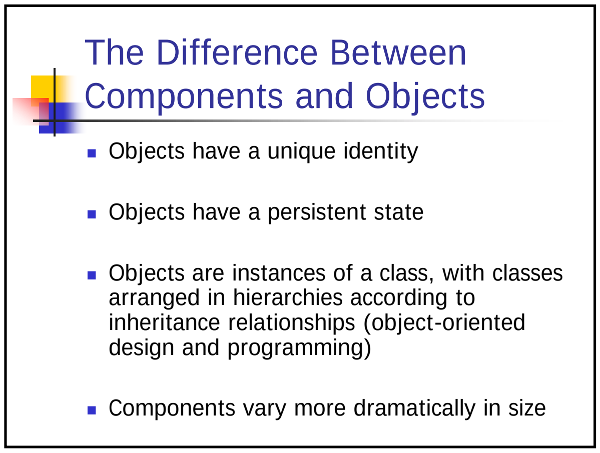
Elements of Software Architecture

- Perry & Wolf
  - Structural Elements
    - Processing
    - Data
    - Connecting ("glue")
  - Form: Weighted Properties and Relationships
  - Rationale
- Shaw & Garlan:
  - Components
  - Interconnections
  - Rules of Composition
  - Rules of Behavior
- Medvidovic & Taylor
  - Architectural Elements: Components, Connectors, Configurations
  - Tool Support



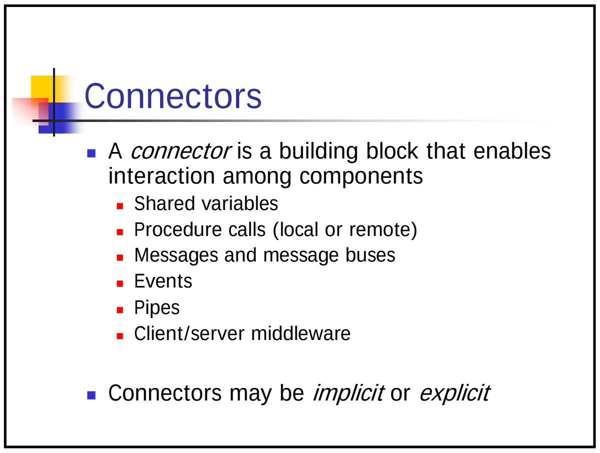
Components

- A *component* is a building block that is ...
  - A unit of computation or a data store, with an interface specifying the services it provides
  - A unit of deployment
  - A unit of reuse



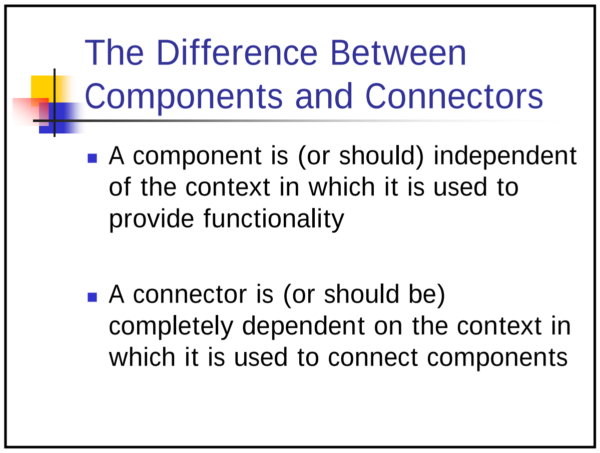
The Difference Between Components and Objects

- Objects have a unique identity
- Objects have a persistent state
- Objects are instances of a class, with classes arranged in hierarchies according to inheritance relationships (object-oriented design and programming)
- Components vary more dramatically in size



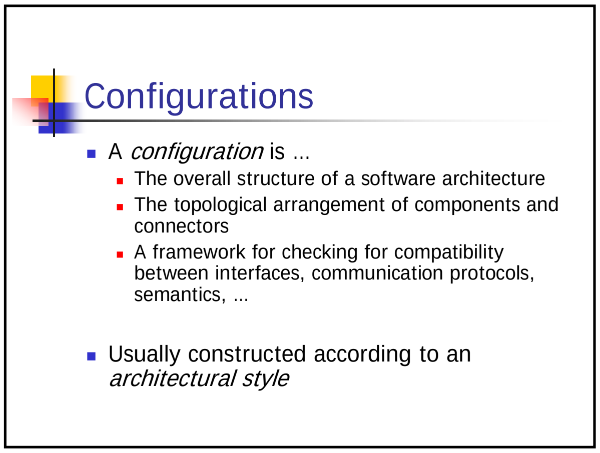
Connectors

- A *connector* is a building block that enables interaction among components
  - Shared variables
  - Procedure calls (local or remote)
  - Messages and message buses
  - Events
  - Pipes
  - Client/server middleware
- Connectors may be *implicit* or *explicit*



The Difference Between Components and Connectors

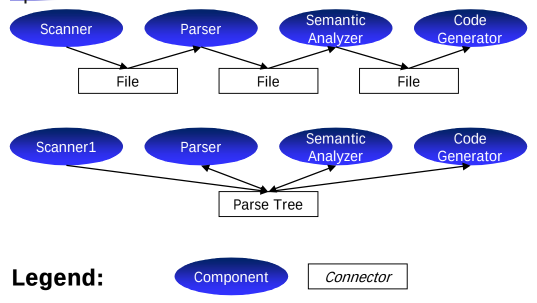
- A component is (or should) independent of the context in which it is used to provide functionality
- A connector is (or should be) completely dependent on the context in which it is used to connect components



Configurations

- A *configuration* is ...
  - The overall structure of a software architecture
  - The topological arrangement of components and connectors
  - A framework for checking for compatibility between interfaces, communication protocols, semantics, ...
- Usually constructed according to an *architectural style*

Example: Architectures for a Compiler



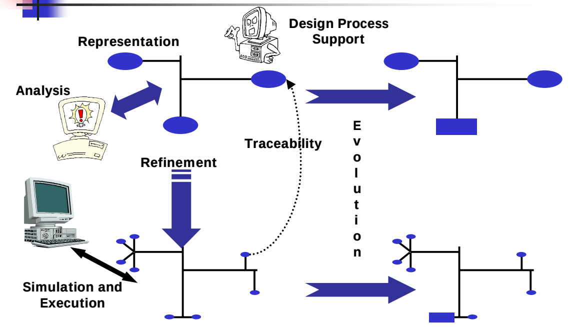
Architecture Description Languages

- An *architecture description language* (or *architecture definition language*, or *ADL*) is a formal notation for describing the structure and behavior of a software architecture
- ADLs provide
  - a concrete syntax
  - a formal semantics
  - a conceptual framework
  - for explicitly modeling the *conceptual architecture* of a system
- A *programming language* is used to define the *implementation architecture* of a system

Why ADLs?

- The problem of software architectural design can be viewed as a language problem
- Informal notations (boxes and arrows) are ambiguous, imprecise, unanalyzable, …

Putting ADLs to Use



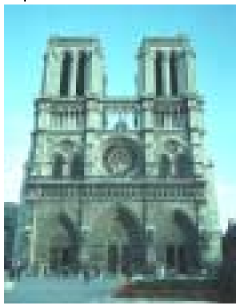
Architectural Styles

- An *architectural style* is …
  - A set of rules for arranging the components and connectors of a system
  - A family of architectures sharing a common pattern of structural organization

The Classical Style of Civil Architecture

The Pantheon
Rome, Italy

The Gothic Style



Nôtre-Dame Cathedral
Paris, France

The Mediterranean Style

Irvine, California

Common Software Architectural Styles

- Dataflow Systems
  - Batch sequential
  - Pipes and filters
- Call-and-Return Systems
  - Main program and subroutines
  - Object-oriented systems
  - Hierarchical layers (onion layers)
- Independent Components
  - Communicating processes (client/server and peer-to-peer)
  - Event systems
  - Implicit invocation

Common Software Architectural Styles (Cont.)

- Virtual Machines
  - Interpreters
  - Rule-based systems
- Data-Centered Systems (Repositories)
  - Databases
  - Hypertext systems
  - Blackboards

The Vision: Architecture-Based Composition & Reuse

- A framework for design and implementation of large-scale software systems
- A basis for early analysis of software system properties
- A framework for selection and composition of reusable off-the-shelf components
- A basis for controlled evolution of software
- A basis for runtime evolution of software

The Reality: Architectural Mismatch

- *Architectural mismatch* refers to a mismatch between assumptions made by different components about the structure of the system and the nature of the environment in which they operate

Assumptions Leading to Architectural Mismatch (I)

- Assumptions about the nature of the components
  - substrate on which component is built
  - control model
  - data model
- Assumptions about the nature of the connectors
  - protocols
  - data model

Assumptions Leading to Architectural Mismatch (II)

- Assumptions about the global configuration
  - topology
  - presence of certain components or connectors
  - absence of certain components or connectors
- Assumptions about the system construction process
  - order in which elements are instantiated
  - order in which elements are combined

Standards: *The Solution?*

- Standards define a set of “assumptions” that all components must adhere to
  - Component interface standards (e.g., JavaBeans, ActiveX, Netscape Plug-in API)
  - Component interoperability standards (e.g., CORBA, DCOM, Java RMI)
  - Standard component frameworks (e.g., Microsoft Foundation Classes)
  - Domain-Specific Software Architectures (DSSAs)
- But standards also reduce design flexibility