Software Architecture and Design

Instructor: Yongjie Zheng
February 10, 2020

CS 441: Software Engineering

Outline

- **What** is software architecture?
- **Why** do we need software architecture?
- **How** do we design and model software architecture?
  - Fundamental principles
  - Architecture patterns and styles
  - Function-oriented design
  - Object-oriented design: UML, design patterns

What is software architecture? (in traditional software engineering)

- *The top-level decomposition of a software system into major sub-systems together with a characterization of how these sub-systems interact is called software architecture.*
- Software architecture is top-level design or global design!
- Software architecture is the result of the initial design process.

What is software architecture? (in current research)

- *The software architecture of a program or computing system is the structure or structures of the system, which comprise software components, the externally visible properties of those components and the relationships among them.* - [Bass et al., 1998]
- *Software architecture* = {*Elements, Form, Rationale*} - [Dewayne Perry & Alex Wolf, 1992]
- This is the so-called *4C* model.
  - **C**omponent, **C**onnector, **C**onfiguration, **C**onstraint.

What is software architecture? (a new definition)

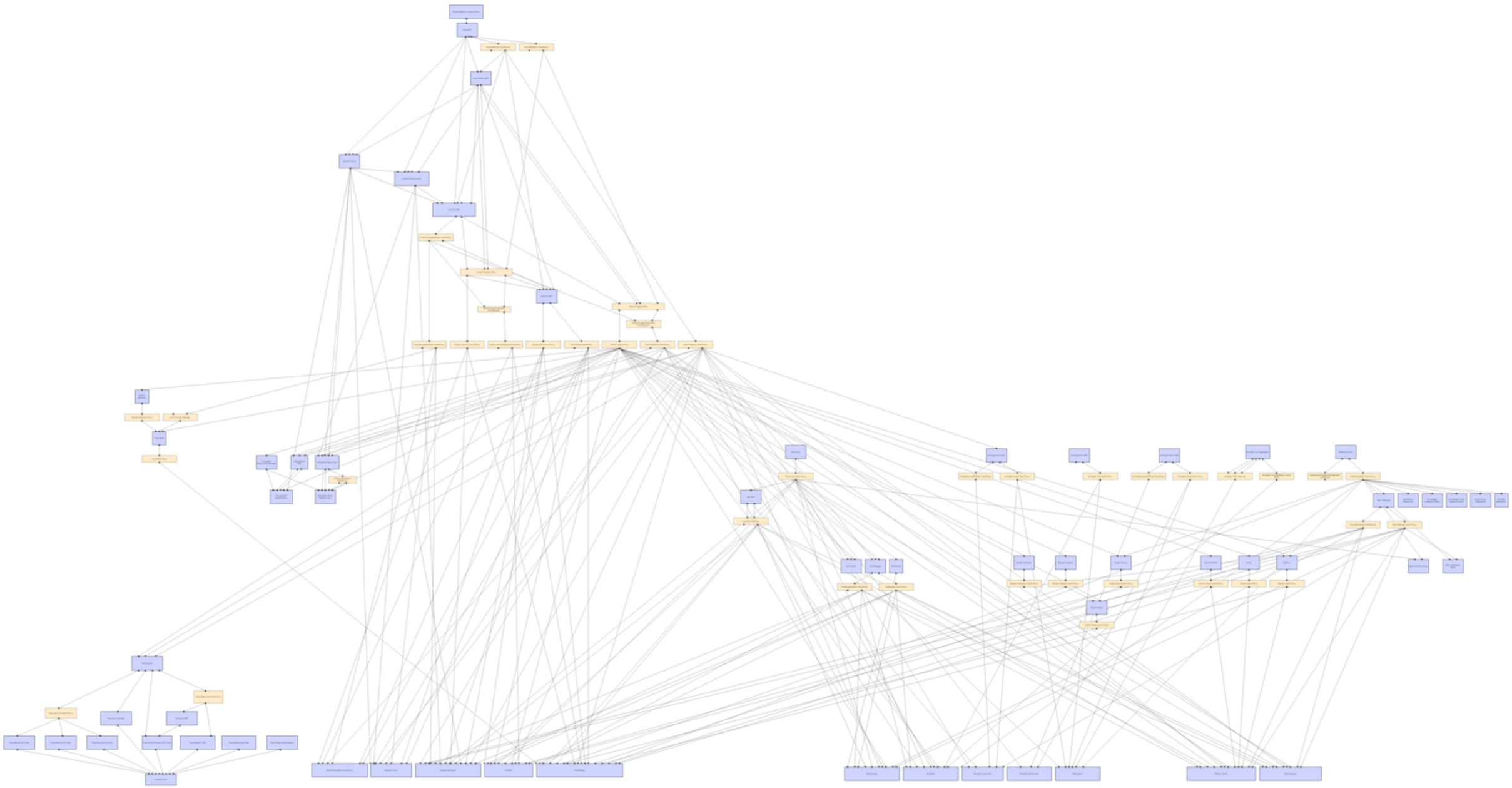
- *A software system's architecture is the set of principal design decisions made about the system.* - [Taylor & Medvidovic & Dashofy]

- This definition particularly emphasizes *extensibility* of software architecture.
  - "principal" is up to stakeholders to decide.
  - Note that the definition does not say anything about what software architecture should look like, and how it should be modeled.

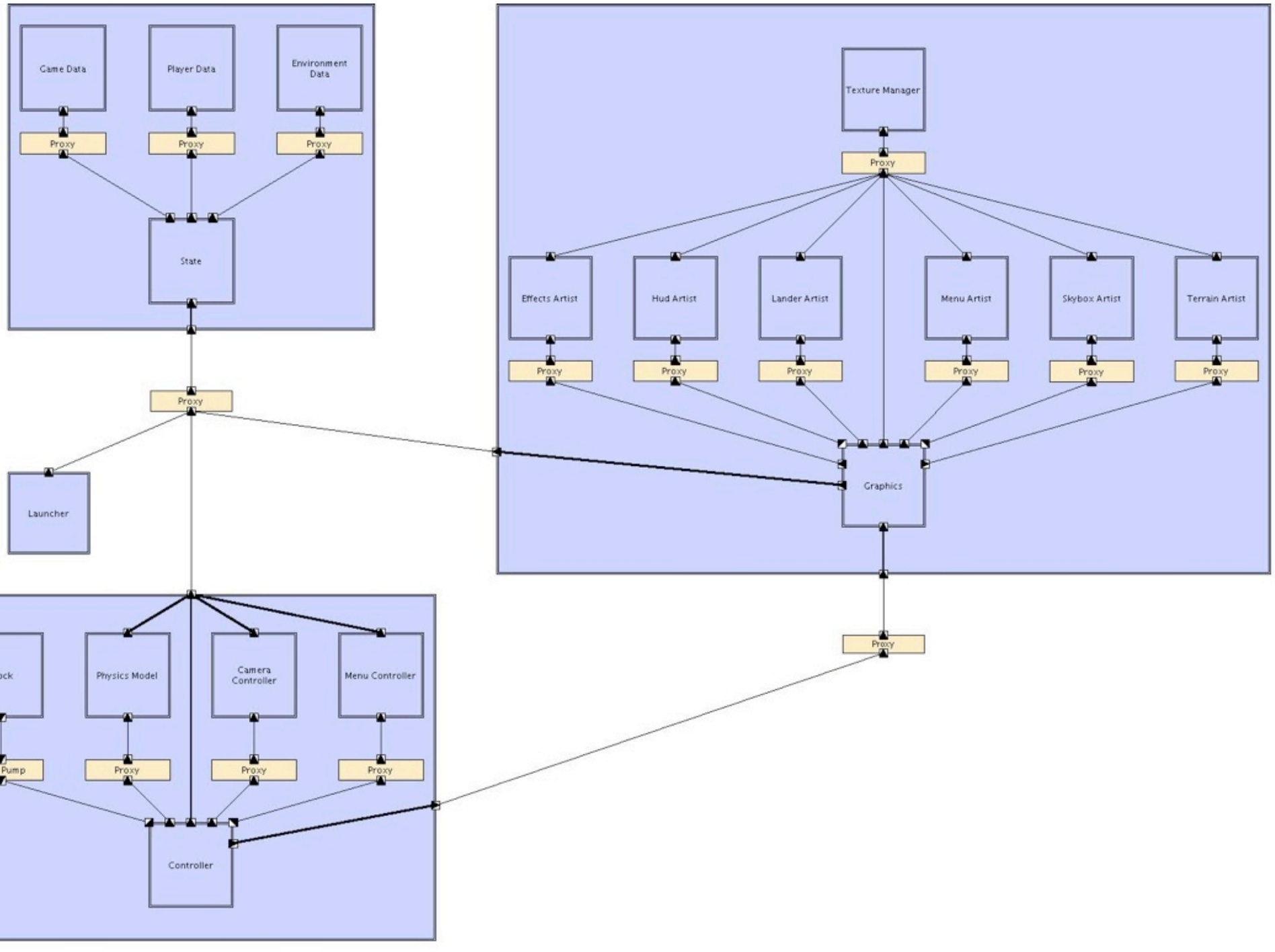
Components and Connectors

- **Component** is an architectural entity that (1) encapsulates a subset of the system's functionality and/or data, (2) restricts access to that subset via an explicitly defined interface, and (3) has explicitly defined dependencies on its required execution context.
- **Connector** is an architectural element tasked with effecting and regulating interactions among components.

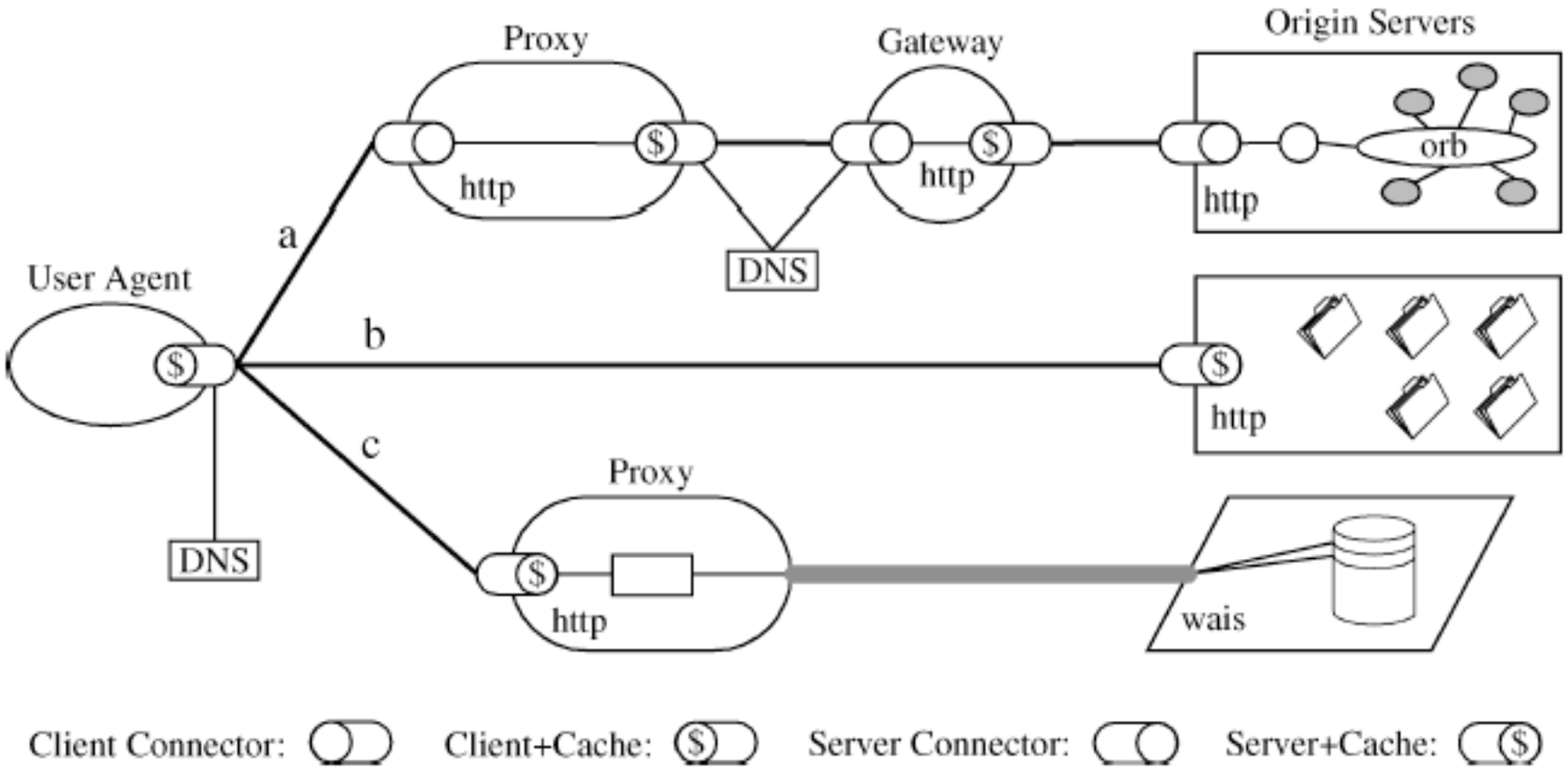
An example of software architecture



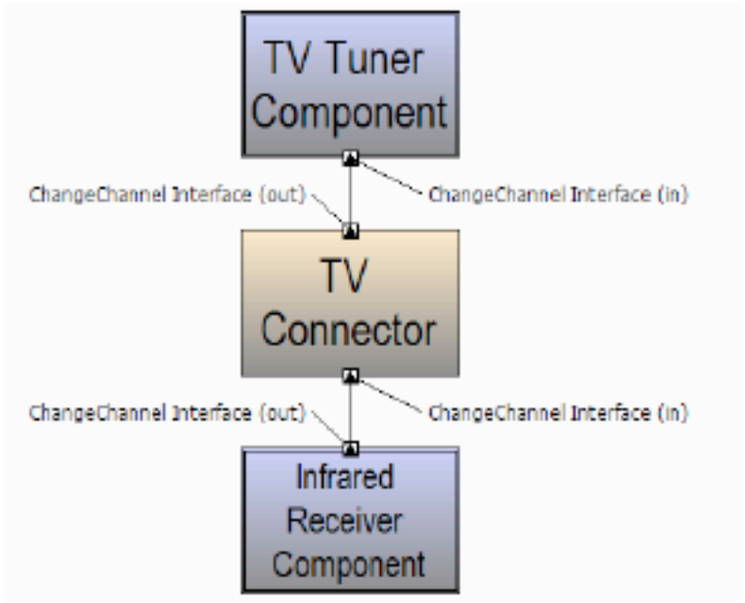
A less complicated example



Another example



```
<xArch>
  <archStructure id="tvset">
    <description>TV Set</description>
    <component id="tuner">
      <description>
        TV Tuner Component
      </description>
      <interface id="tuner.channel">
        <description>
          ChangeChannel Interface
          (in)
        </description>
        <direction>in</direction>
      </interface>
    </component>
    <component id="ir">
      <description>
        Infrared Receiver Component
      </description>
      <interface id="ir.channel">
        <description>
          ChangeChannel Interface
          (out)
        </description>
        <direction>out</direction>
      </interface>
    </component>
    <connector id="tvconn">
      <description>
        TV Connector
      </description>
      <interface id="tvconn.in">
        <description>
          ChangeChannel Interface
          (in)
        </description>
        <direction>in</direction>
      </interface>
      <interface id="tvconn.out">
        <description>
          ChangeChannel Interface
          (out)
        </description>
        <direction>out</direction>
      </interface>
    </connector>
```




```
    <link id="link1">
      <description>
        Tuner to Connector
      </description>
      <point>
        <anchor
          href="#tuner.channel"/>
      </point>
      <point>
        <anchor
          href="#tvconn.out"/>
      </point>
    </link>
    <link id="link2">
      <description>
        Connector to IR
      </description>
      <point>
        <anchor href="#tvconn.in"/>
      </point>
      <point>
        <anchor
          href="#ir.channel"/>
      </point>
    </link>
  </archStructure>
</xArch>
```

Why do we need software architecture? (architecture in context)

- Requirements
  - Existing architectures provide vocabulary for requirements.
- Design
  - Software architecture is the outcome of software design.
- Implementation
  - High-level reuse
- Testing
  - Early system analysis
- Maintenance
  - Open architecture

How do we design architecture?

- Creativity
  - This requires extensive experience, broad training, ...
- Principles, process, and methods
  - Goals, activities, and principles
  - Design methods: object-oriented design, functional design, and quality-driven design
- Reuse
  - Horizontal reuse: architecture patterns and styles
  - Vertical reuse: product-line architectures

Goals (Considerations) of Architecture Design

Conceptual Integrity

The fact that a software product presents to each of its users a coherent mental model of the application, of strategies for doing the application, and of the user-interface tactics to be used in specifying actions and parameters. The conceptual integrity of the product is the most important fact in ease of use. - [Fred Brooks]

Conceptual integrity implies that the similar or same design decisions are made to solve a collection of similar problems for the same goal.

Activities of Architecture Design

- Analyzing and refining requirements
- **Decomposing** the system into components
- Selecting protocols for communication, synchronization, and data access
- Developing global structures
- Designing component internal structures
- Selecting among design alternatives (often needs to consider non-functional properties)
- Dealing with deployment issues
- Making stakeholder related decisions

Design Principles

- **Abstraction**: we concentrate on the essential features and ignore, abstract from, details that are not relevant at the level we are currently working.
- **Modularity**: the degree (cohesion, coupling) to which a system is partitioned into components (or modules).
  - Cohesion: a measure of the mutual affinity of the elements of a component. E.g. functional cohesion, data cohesion.
  - Coupling: a measure of the strength of the inter-component connections. E.g. control coupling, data coupling.
  - High-cohesion and loose-coupling are generally preferred.

Design Principles, cont.

- **Information hiding** (i.e., encapsulation): one begins with a list of difficult design decisions or design decisions which are likely to change. Every module (component) is designed to hide such a decision from all others. Its interface or definition was chosen to reveal as little as possible about its inner workings. [Parnas]

Reference

- Richard N. Taylor, Nenad Medvidovic, and Eric M. Dashofy. Software Architecture: Foundations, Theory, and Practice. John Wiley and Sons. ISBN-10: 0470167742; ISBN-13: 978-0470167748. 2010.