Architecture Modeling

Instructor: Yongjie Zheng

CS 5553: Software Architecture and Design

Basic Concepts

- **Architecture model**: an artifact that captures some or all of the design decisions that comprise a system's architecture.
- **Architecture modeling**: the reification and documentation of architecture design decisions.
- **Architecture modeling notation**: a language or means of capturing design decisions.

Basic Concepts

- **Architecture component**: 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.

- **Architecture connector**: an architectural element tasked with effecting and regulating interactions among components.

- A component typically provides application-specific services, while a connector typically provides application-independent interaction facilities.

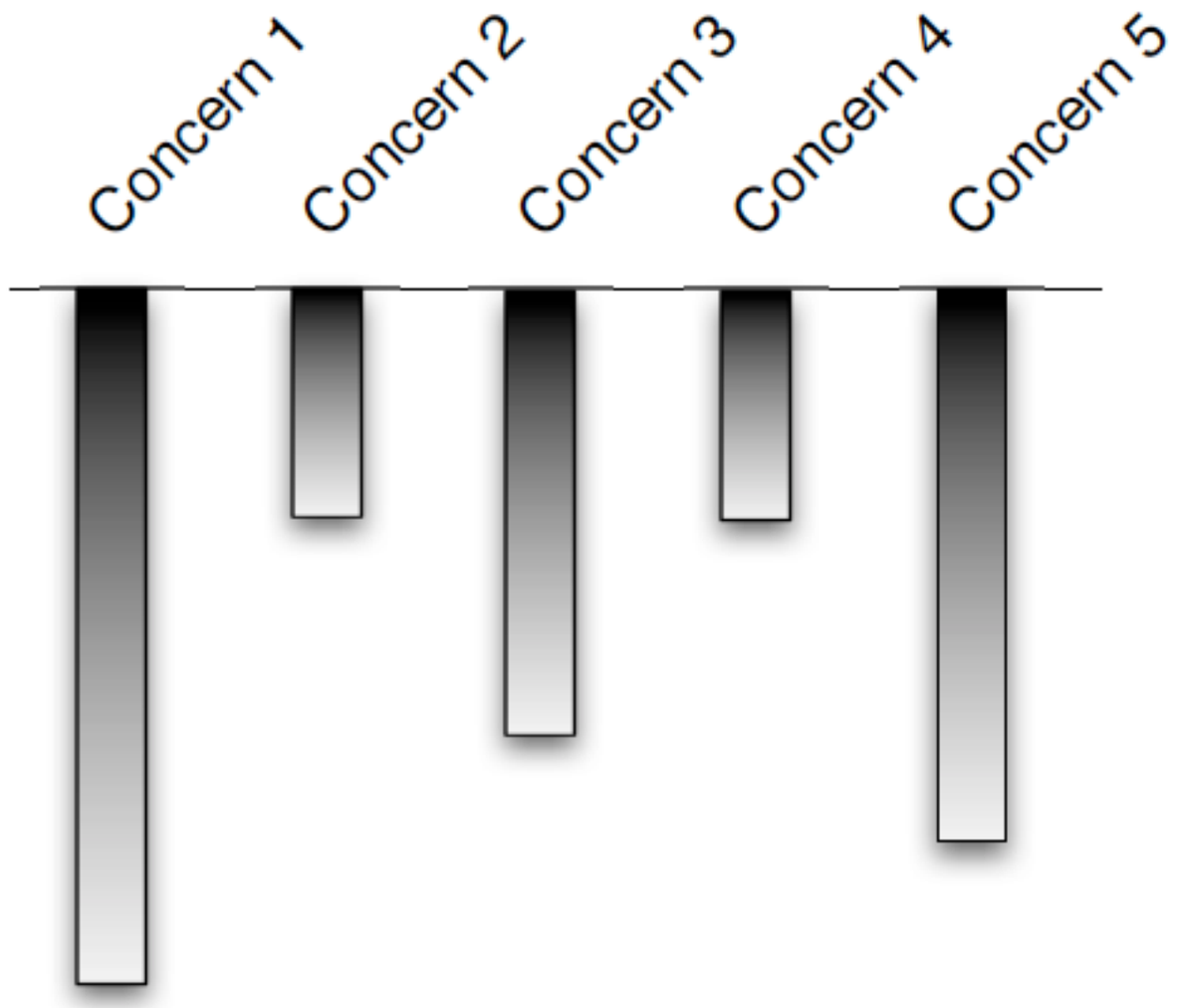
Basic Concepts

- **Architecture interface**: an interface is a specification of component behavior and properties; technically, an interface is a set of semantically related operations that can be invoked by clients.
- An architecture component normally has multiple provided interfaces and required interfaces.
  - A provided interface provides a service as a contract for other components to use.
  - A required interface represents what the external environment needs to provide for the component to function.

Modeling Considerations

- The architectural decisions and concepts that should be modeled.
  - Different architectural decisions may be of different importances.
- The level of detail.
- The amount of rigor or formality.
  - Formalization generally requires the use of a formal method.

Rule of Thumb



The most important or critical aspects of a system should be the ones that are modeled in the greatest detail with the highest degrees of rigor/formality.

Modeling Process

1. Identify relevant aspects of the software to model
2. Roughly categorize them in terms of importance.
3. Identify the goals of modeling for each aspect (communication, bug finding, quality analysis, generation of other artifacts, and so on).
4. Select modeling notations that will model the selected aspects at appropriate levels of depth to achieve the modeling goals.
5. Create the models.
6. Use the models in a manner consistent with the modeling goals.

Modeling Basic Architecture Elements

- Components, connectors, interfaces, configurations (can be modeled by simple box and arrow diagrams).
- Rationale: the information that explains why particular architectural decisions were made, and what purpose various elements serve (usually expressed using natural languages).

Modeling the Architecture Style

- Applications that are large, dynamic, and distributed may be harder to model (e.g. World Wide Web).
  - At this point, it may be more effective to model the underlying architecture style.

- Elements of an architecture style
  - Special elements: the existence of particular components, connectors, or interfaces.
  - Constraints: interaction constraints, behavioral constraints, concurrency constraints.

Modeling Static and Dynamic Aspects

- **Static aspects**: do not involve the system's behavior.
  - Component/connector topologies
  - Assignments of components/connectors to processing elements or hosts
  - Host and network configuration
- **Dynamic aspects**: involve the system's runtime behavior.
  - Behavioral models: describing the behavior of a component or connector over time.
  - Interaction models: describing the interactions between a set of components and connectors over time.
  - Data flow models: describing how data flows through an architecture over time.

We need to differentiate

- Modeling static and dynamic aspects of a system
  - A model of a dynamic aspect of a system describes how the system changes as it executes.
  - For example, a statechart models dynamic aspects of a system, but the statechart itself does not change.
- Using static or dynamic models.
  - A dynamic model changes itself as the system executes.
  - A dynamic model is not necessary to capture dynamic aspects of a system.

Modeling Functional and Non-Functional Aspects

- Functional aspects: what a system does.
  - Functional aspects of a system can be described using declarative, subject-verb sentences: e.g. *The system prints medical records*.
- Non-functional aspects: how a system performs its functions.
  - Non-functional aspects of a system can be described by adding adverbs to these sentences: e.g. *The system prints medical records quickly and confidentially*.
  - Natural languages are often used to capture non-functional aspects.

Ambiguity, Accuracy, and Precision

- **Ambiguity**: a model is ambiguous if it is open to more than one interpretation.
  - Incompleteness is a primary reason for ambiguity. Therefore, architecture models could be ambiguous.
- **Accuracy**: a model is accurate if it is correct, conforms to fact, or deviates from correctness within acceptable limits.
- **Precision**: a model is precise if it is specific, detailed, and exact.
- Stakeholders should generally choose notations that allow unambiguous, accurate, and precise modeling of the aspects of the system that are most important.

Accuracy vs. Precision

Neither accurate nor precise.

(a)

(b)

Accurate, but not precise.

Precise, but not accurate

(c)

(d)

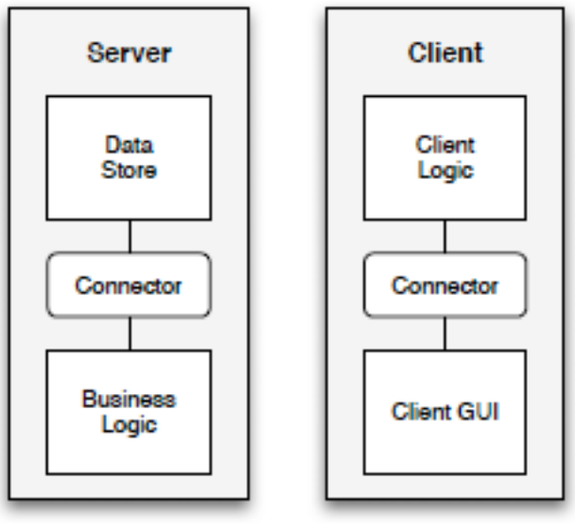
Both accurate and precise.

Consider each shot to be an assertion about a system being designed.

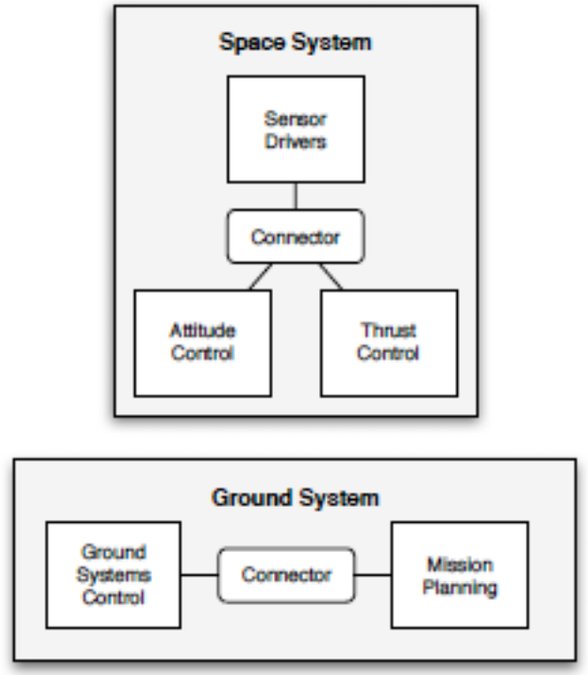
Views and Viewpoints

- Generally, it is not feasible to capture everything we want to model in a single model or document.
- So, we create several coordinate models, each capturing a subset of design decisions.
    - Generally, the subset is organized around a particular concern or other selection criteria.
- We call the subset model a “view” and the concern a “viewpoint”.

Views and Viewpoints Example



The deployment view of a client-server system.



The deployment view of a distributed space-ground system.

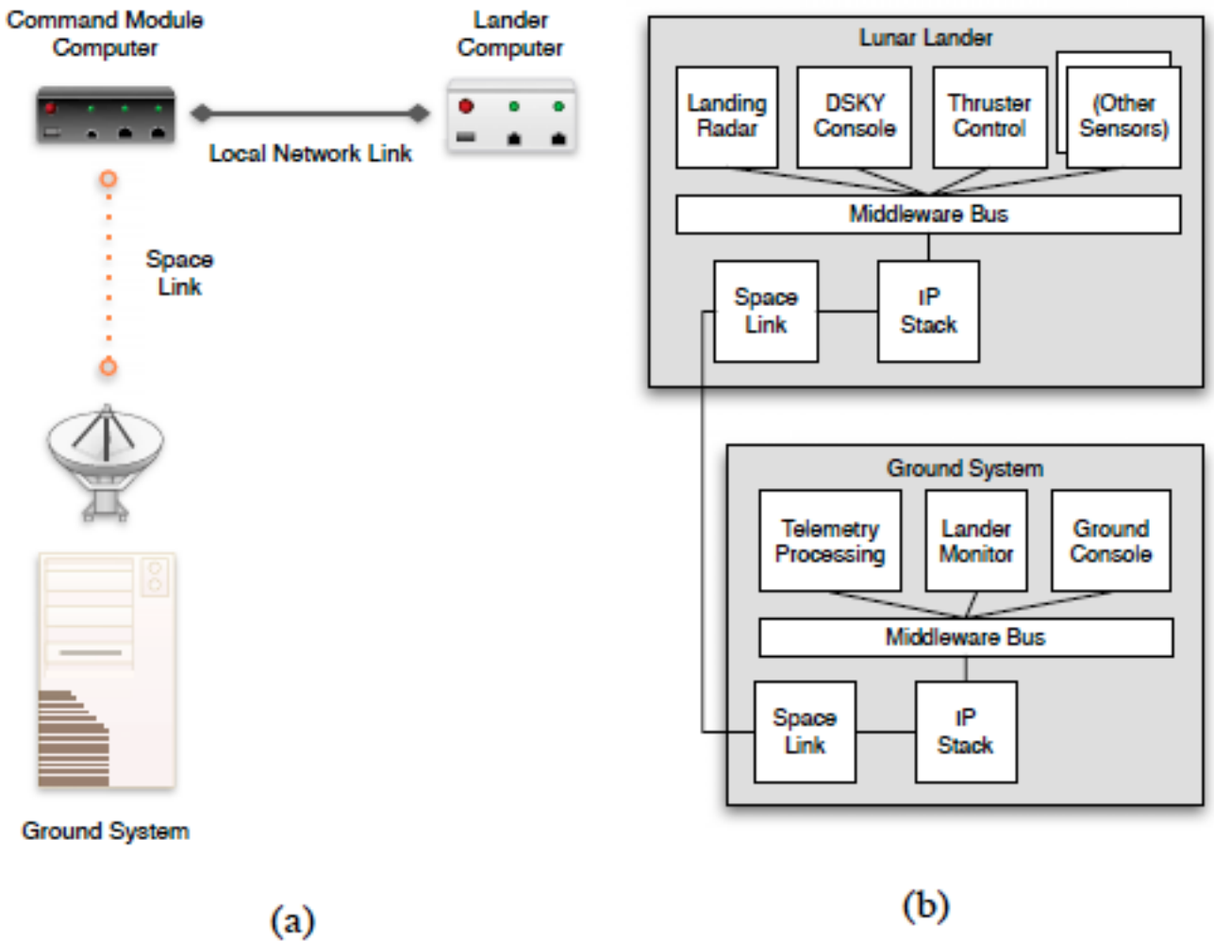
Commonly-Used Viewpoints

- **Logic Viewpoint**: captures the logical (often software) entities in a system and how they are interconnected.
- **Physical Viewpoint**: captures the physical (often hardware) entities in a system and how they are interconnected.
- **Deployment Viewpoint**: captures how logical entities are mapped onto physical entities.
- **Concurrency Viewpoint**: captures how concurrency and threading will be managed in a system.
- **Behavioral Viewpoint**: captures the expected behavior of (parts of) a system.

Consistency among Views

- Views can contain overlapping and related design decisions.
  - There is the possibility that the views can thus become inconsistent with one another.
- Views are **consistent** if the design decisions they contain are compatible.
  - Views are **inconsistent** if two views assert design decisions that cannot simultaneously be true.
- Inconsistency is usually but not always indicative of problems.
  - Temporary inconsistencies are a natural part of exploratory design.

Example of View Inconsistency



(a) The physical view of a ground system.
(b) The deployment view of the system.

Common Types of Inconsistencies

- Direct inconsistencies: two views assert directly contradictory propositions.
- Refinement inconsistencies: high-level (more abstract) and low-level (more concrete) views of the same parts of a system conflict.
- Static aspects versus dynamic aspects: dynamic aspects (e.g., behavioral specifications) conflict with static aspects (e.g., topologies).
- Dynamic aspects: different descriptions of dynamic aspects of a system conflict.
- Functional versus non-functional aspects

Evaluating Modeling Techniques

- **Scope and purpose**: What does the technique help you model? What does it not help you model?
- **Basic elements**: What are the basic elements (the 'atoms') that are modeled? How are they modeled?
- **Style**: To what extent does the approach help you model elements of the underlying architectural style? Is the technique bound to one particular style or family of styles?
- **Static and dynamic aspects**: What static and dynamic aspects of an architecture does the approach help you model?
- **Dynamic modeling**: To what extent does the approach support models that change as the system executes?
- **Non-functional aspects**: To what extent does the approach support (explicit)modeling of non-functional aspects of architecture?
- **Ambiguity**: How does the approach help you to avoid (or embrace) ambiguity?
- **Accuracy**: How does the approach help you to assess the correctness of models?
- **Precision**: At what level of detail can various aspects of the architecture be modeled?
- **Viewpoints**: Which viewpoints are supported by the approach?
- **View consistency**: How does the approach help you assess or maintain consistency among different viewpoints?