Architectural Modeling

Lecture 6

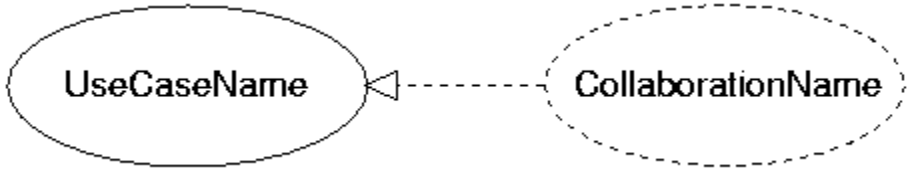
Part I

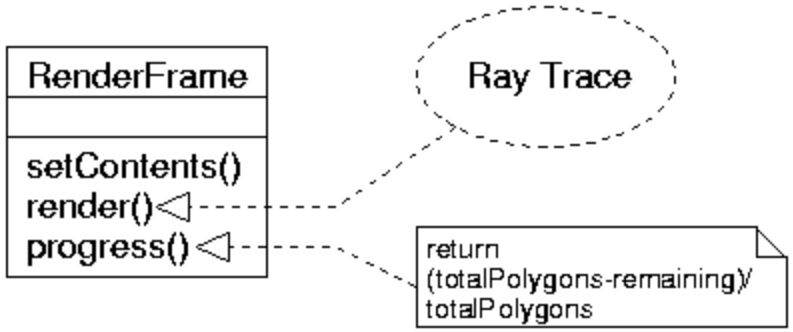
Realizations
and
Collaborations

Realization Relationship

- A semantic relationship between components in which one component specifies a contract that another component guaranties to carry out
- Semantically, a realization is a cross between dependency and generalization
- It binds a “problem” with a “solution”
- Used in two contexts: interfaces and collaborations

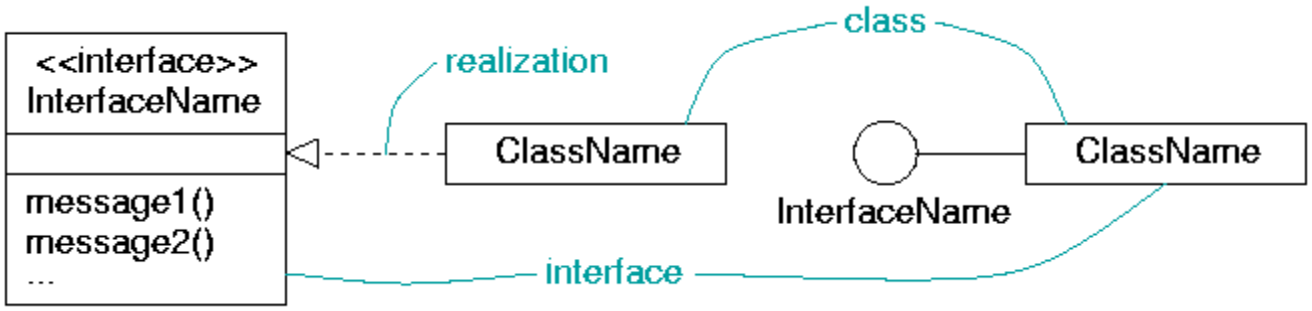
Realization Examples





Interface

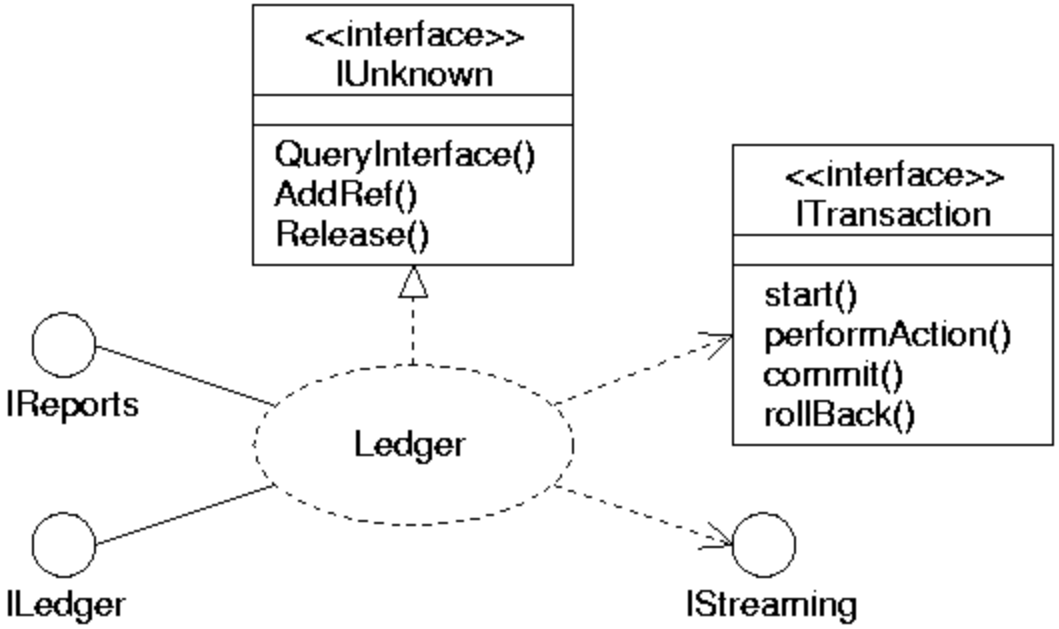
- A named collection of operations that are used to specify a service of a class or component
- Can be rendered either using a prototyped class without attributes or the “lollipop” icon
- As a convention, their name begins with an “I” to distinguish them from classes



Defining Interfaces

- Identify boundaries between groups of tightly coupled classes and component
- Components that tend to change together should be grouped as collaborations
- Identify the operations that cross the boundaries and package them in logically related sets (interfaces)
- For each collaboration in the system, identify the interfaces it defines and relies on. Interface imports are dependencies, exports are realizations
- The dynamics of each interface can be specified using pre and post conditions or state machines

Interface Example



Hints and Tips

- Well-structured interfaces are:
  - simple yet complete
  - understandable without need of inspection of its realizer component
  - extremely important in the project

Part II

Collaborations:
A general abstraction mechanism

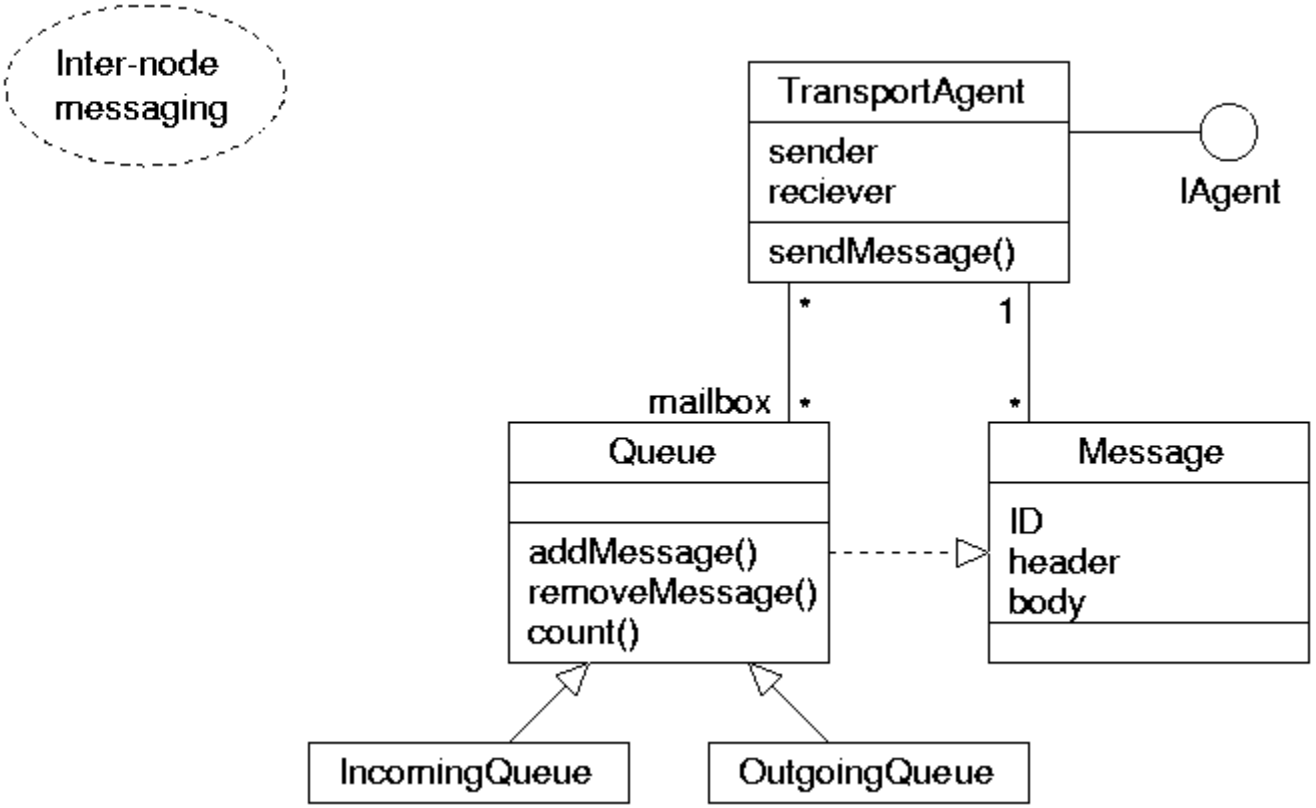
Collaborations

- A conceptual chunk that encompasses both static and dynamic aspects of a problem
- Names a society of classes, interfaces and other elements that work together to provide some cooperative behavior
- Used to specify the realization of use cases and operations and to model other architecturally significant mechanisms of the system
- Provides a general abstraction mechanism

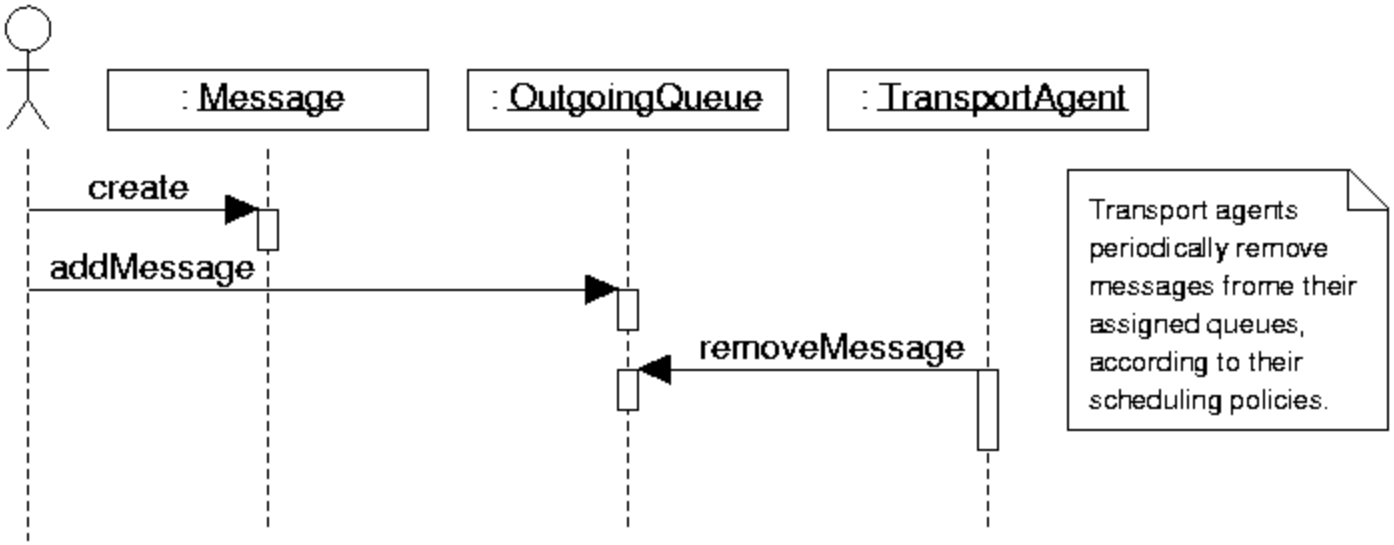
Collaboration Example

- Collaborations have two aspects:
  - a structural part that specifies the classes, interfaces, and other elements that work together to carry out the collaboration (class diagram)
  - a behavioral part that specifies the dynamics of how these elements interact (sequence diagram(s), interaction diagram)
- You can "zoom inside" a collaboration to reveal its structural and behavioral aspects
- Unlike packages and subsystems, collaborations do not own any of these elements
- It just uses and aggregates elements into conceptual chunks
- It may cross subsystem boundaries

Collaboration Example



Collaboration Example



Organizing Collaborations

- The mechanisms that shape a system represent significant design decisions
- Collaborations are the heart of a system's architecture
- You should come with a modest number of regularly sized collaborations
- Collaborations can have relationships with:
  - the thing it realizes, e.g. the relation between a use case or operation and its realizing collaboration
  - other collaborations, e.g. <<refine>> relationships similar to those defined on the corresponding use cases
- Collaborations can be grouped into larger packages, though it is only used for extremely large systems

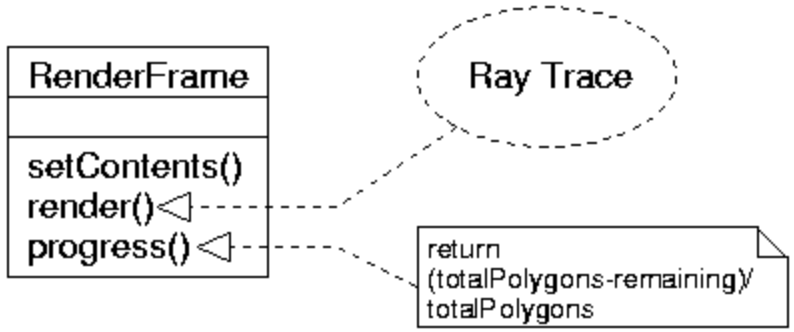
Realizing Use Cases

- Analysis is typically driven by the system's use cases
- Use cases are realized using collaborations
- The structural contents of such collaborations will eventually overlap
- Process:
  - identify the structural elements necessary to carry out the semantics of the use case
  - capture their organization in a class diagram
  - identify the set of scenarios that represent this use case
  - capture the dynamics of these scenarios in sequence and/or collaboration diagrams
  - organize these structural and behavioral elements as a collaboration and connect it to the use case with a realization

Realizing Operations

- In many simple cases, operations can be specified using straight code or an algorithm
- More complex operations can be specified using activity diagrams
- For complex operations where several classes pay a role, collaborations can be used. Procedure:
  - identify the parameters, return value, and other objects visible to the operation
  - if the operation is complex enough or otherwise requires some detailed design work, use a collaboration to represent its implementation. The collaboration's class and interaction diagrams can be used to specify the operation in detail.

Realizing Operations



Realizing Mechanisms

- A mechanism is a design pattern that can be applied to various societies of classes in different contexts, e.g. a queue mechanism
- Mechanisms can be represented as collaborations
- The use of mechanisms (design patterns) make the system:
  - **simple**: mechanisms reify common interactions
  - **understandable**: the system can be approached through its mechanisms
  - **resilient**: the whole system can be tuned by tuning its mechanisms
- Design Patterns will be discussed in more details later

Hints and Tips

- A well-structured collaboration
  - consists of both structural and behavioral aspects
  - provides a crisp abstraction of some identifiable interaction in the system
  - is rarely completely independent, and sometimes overlaps with other collaborations
  - is understandable and simple
- Not all collaborations are useful in the final documentation
- Collaborations should be organized according to the classifier or operation they represent, or in packages associated with the system as a whole

Part III

Systems and Models

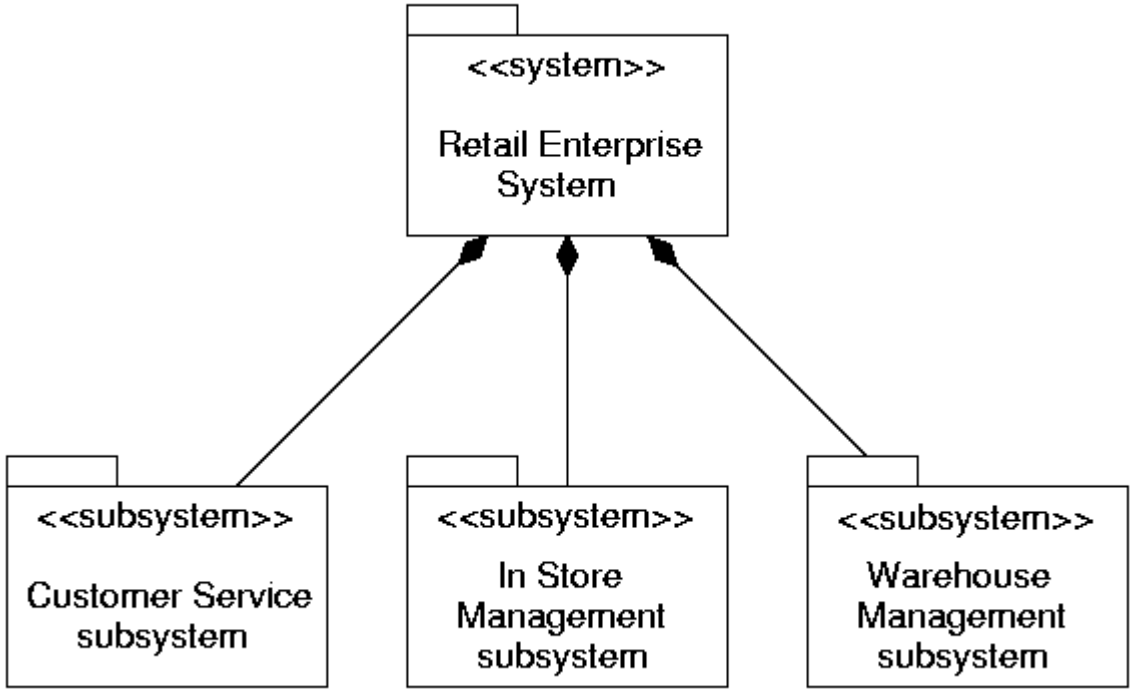
Systems and Models

- **System**: a set of elements organized to accomplish a purpose and described by a set of models describing the system from different viewpoints. A system can be decomposed into subsystems
- **Model**: a simplification of reality, an abstraction of a system created in order to better understand the system
- **View**: a projection of a model which is seen from one perspective and omits entities that are not relevant to this perspective

Systems and Subsystems

- A system represents the highest-level thing in a given context
- The subsystems that make-up a system provide a complete and non-overlapping partitioning of the system as a whole
- In the UML, a system (or subsystem) is rendered as a stereotyped package
- A package owns elements such as use cases, collaborations, interaction diagrams, etc. that specify the structure and behavior of the modeled system
- The primary relationship among systems (and subsystems) is aggregation. A system is defined in terms of subsystems. Similar systems can be generalized

Example



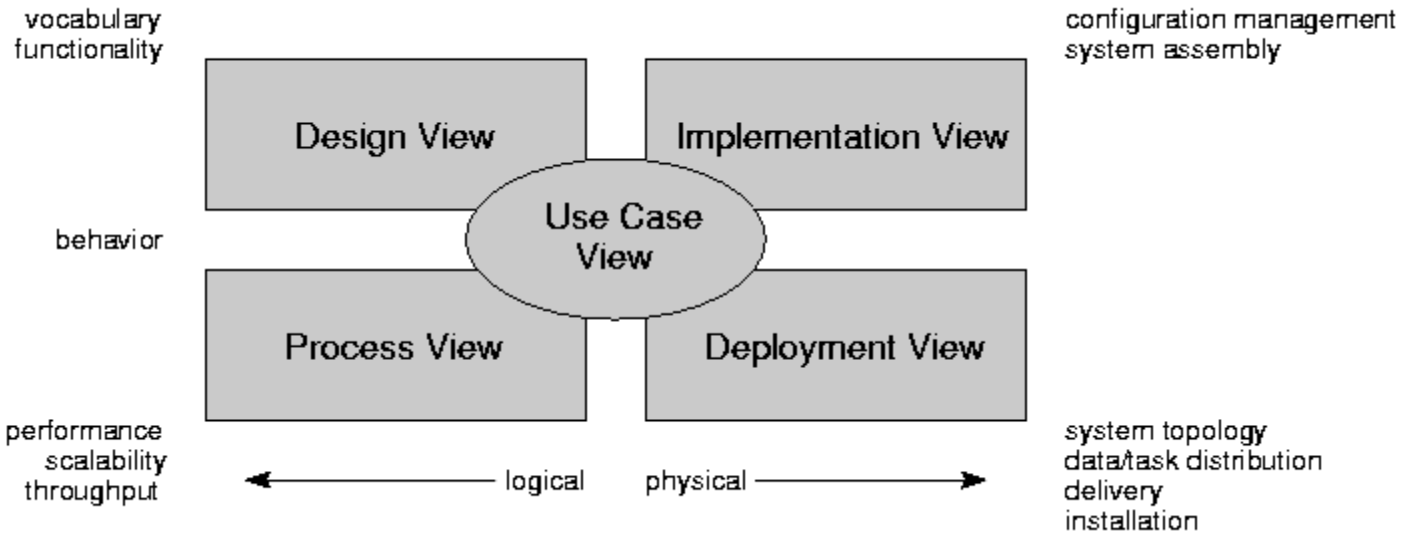
Modeling System Architecture

- Captures the decisions about the system's requirements, logical elements and physical elements
- Models both structural and behavioral aspects of the system
- Defines the seams between subsystems, and tracing between requirements and deployment

Modeling System Architecture

- Identify the views used to represent the system: use case view, design view, process view, implementation view and deployment view
- Specify the context of the system, including actors and other systems
- If necessary, define subsystems

Modeling System Architecture



Modeling System Architecture

- Specify a use case view encompassing all the use cases on the system. Apply use case, interaction, statechart and activity diagrams to specify each use case
- Specify a design view encompassing the classes, interfaces and collaborations that forms the solution. Apply class, interaction, statechart and activity diagrams to specify the design of the system and subsystems
- Specify a process view encompassing the threads and processes that form the system's mechanisms. Apply the same diagrams as for design, restricted only to active classes

Modeling System Architecture

- Specify an implementation view encompassing the physical components that are used to assemble and release the system. Apply component, interaction, statechart, and activity diagrams.
- Specify a deployment view encompassing the nodes that form the system's hardware topology on which the system executes. Apply deployment, interaction, statechart and activity diagrams.
- Model the architectural and design patterns identified in the system using collaborations.
- In many cases, some steps will not be needed, especially steps 3-4 and 5.

Modeling System Architecture

- The system's architecture is never created in one "big-bang" step
- Rather, the system model is refined incrementally using iterations (spiral model)

Hints and Tips

- A well-structured architectural model:
  - is self-contained in that it does not require other content to understand its semantics
  - collectively provides a complete statement of a system's artifacts
  - is functionally, logically and physically cohesive