Software Architecture Design

Chapter 12

Part of Design Analysis

Designing Concurrent, Distributed, and Real-Time Applications with UML
Hassan Gomaa (2001)

Software Architecture Design

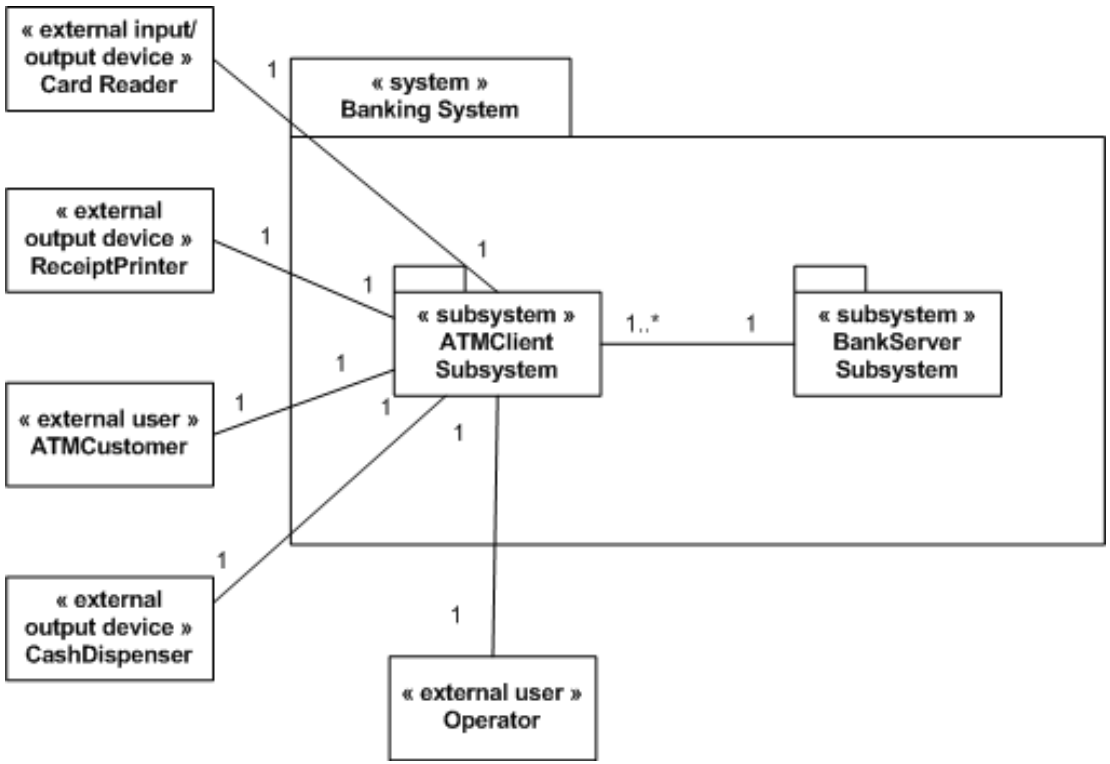
- Software Architecture refers to the decomposition of a system into subsystems
- This is necessary for large-scale and complex software systems.

Software Architectural Styles

- Software Architectural styles are recurring architectures used in a variety of applications.
  - Client/Server Architecture
  - Layers of Abstraction Architecture
  - Communicating Tasks Architecture
- The styles may be blended as necessary. They are not mutually exclusive!

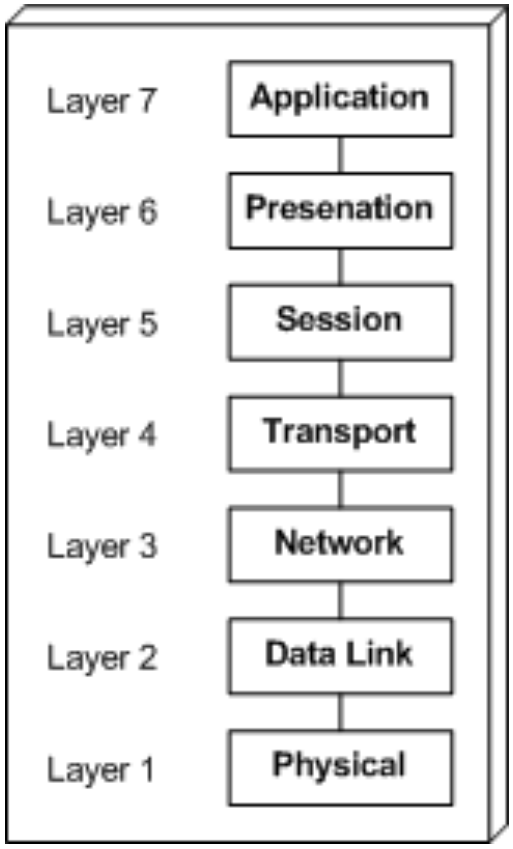
Client/Server Architecture

- Sever provides services
- Client consumes services
- Widely used in distributed applications.



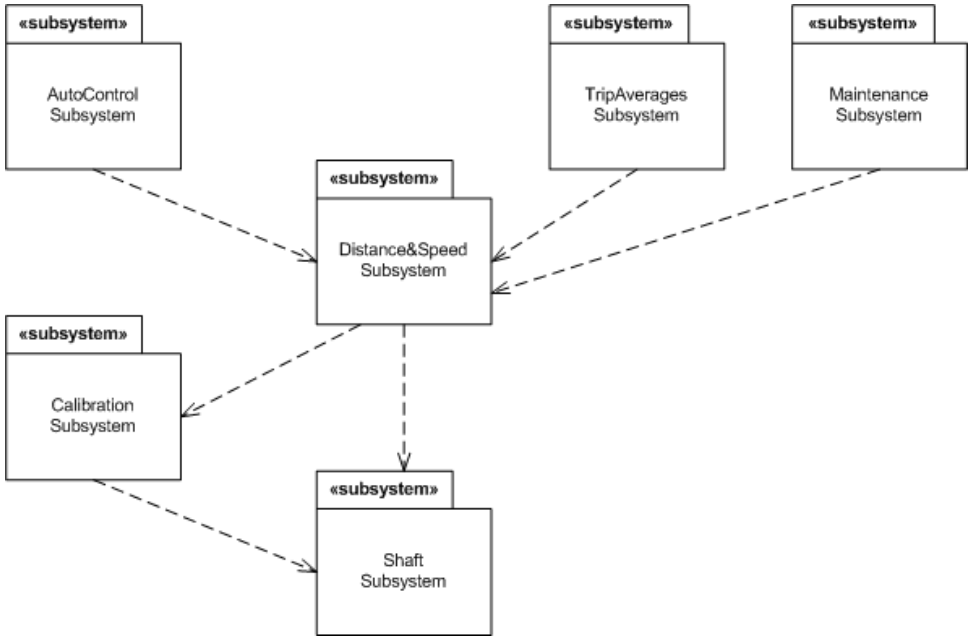
Layers of Abstraction Architecture

- Each layer communicates only with the layers directly above and below itself.
  - Each layer solves only part of a given problem.
- E.g. Operating Systems, Database Management systems, and Network Systems



Communicating Tasks Architecture

- Network of Concurrent tasks with separate threads of control.
- With or without shared memory.
- May or may not be on the same computational node.
- Common in real-time systems.



System Decomposition Issues and Guidelines

- Subsystems should be externally lowly-coupled and internally highly-coupled.
  - This can help to determine which objects belong in which subsystem.
- Separation of Concerns is the fundamental principle guiding decomposition.
- Subsystems provide lower-resolution information hiding than objects.
- If no obvious decompositions appear, consider the Use Cases.

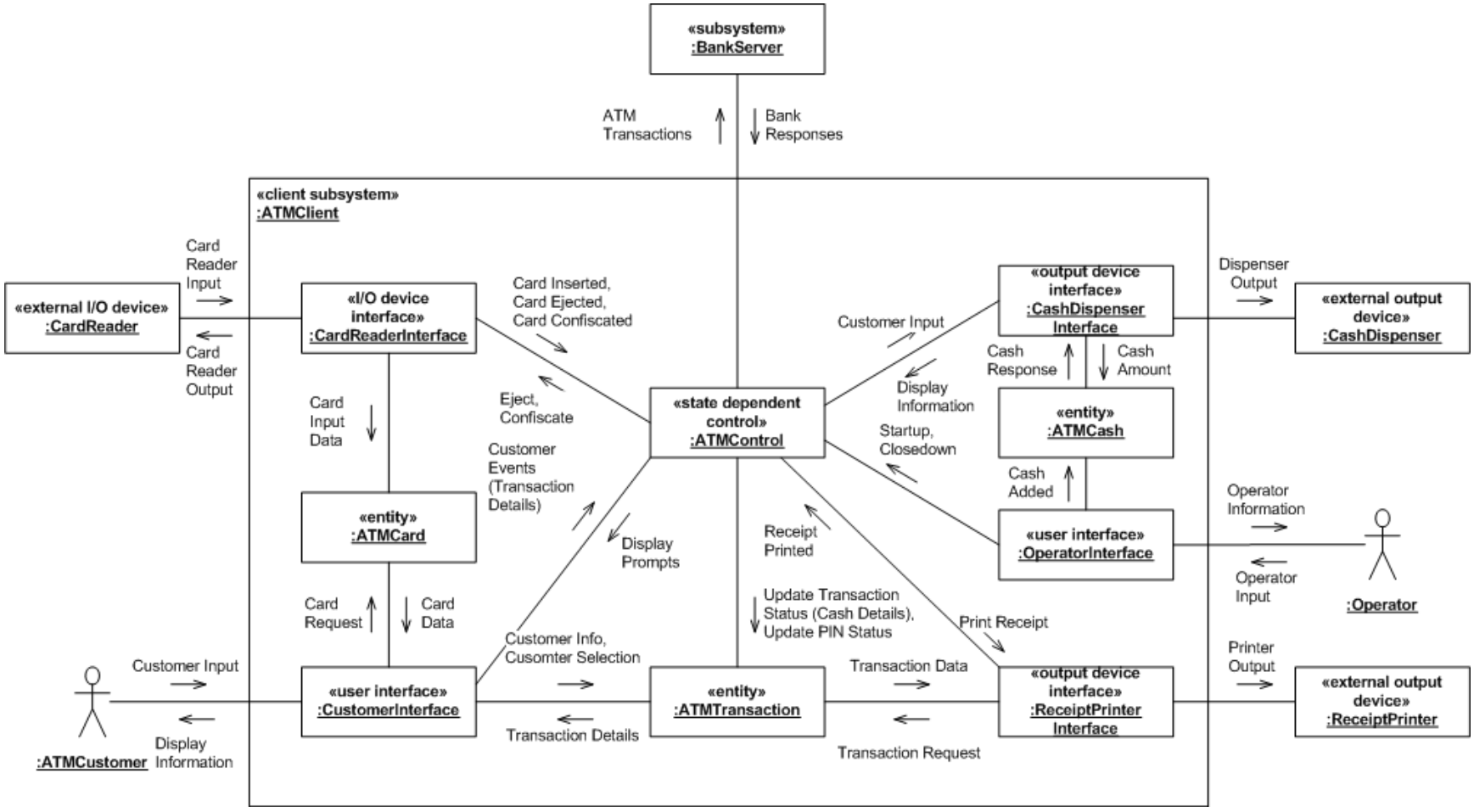
Subsystem Decomposition Issues and Guidelines (cont)

- Things to consider during separation of concern decomposing:
  - Aggregate/Composite objects
  - Geographical locations
  - Clients and servers
  - User interface
  - External object interfaces
  - Scope of control
  - Entity objects

Consolidated Collaboration Diagrams

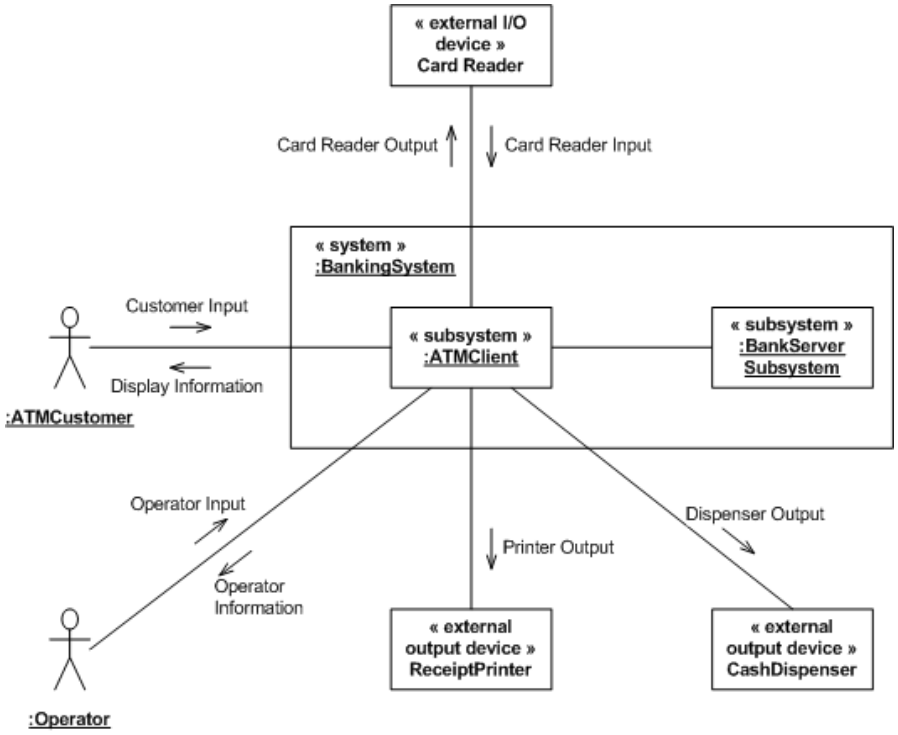
- To begin the system decomposition process, and to transition from analysis to design, construct a consolidated collaboration diagram.
- The consolidated collaboration diagram synthesizes all the collaboration diagrams used to support the use cases
  - Message sequencing numbers are omitted to reduce clutter
  - If the diagram is still too cluttered, multiple messages can be aggregated into one name.

Consolidate Collaboration Diagram (example)



Subsystem Software Architecture

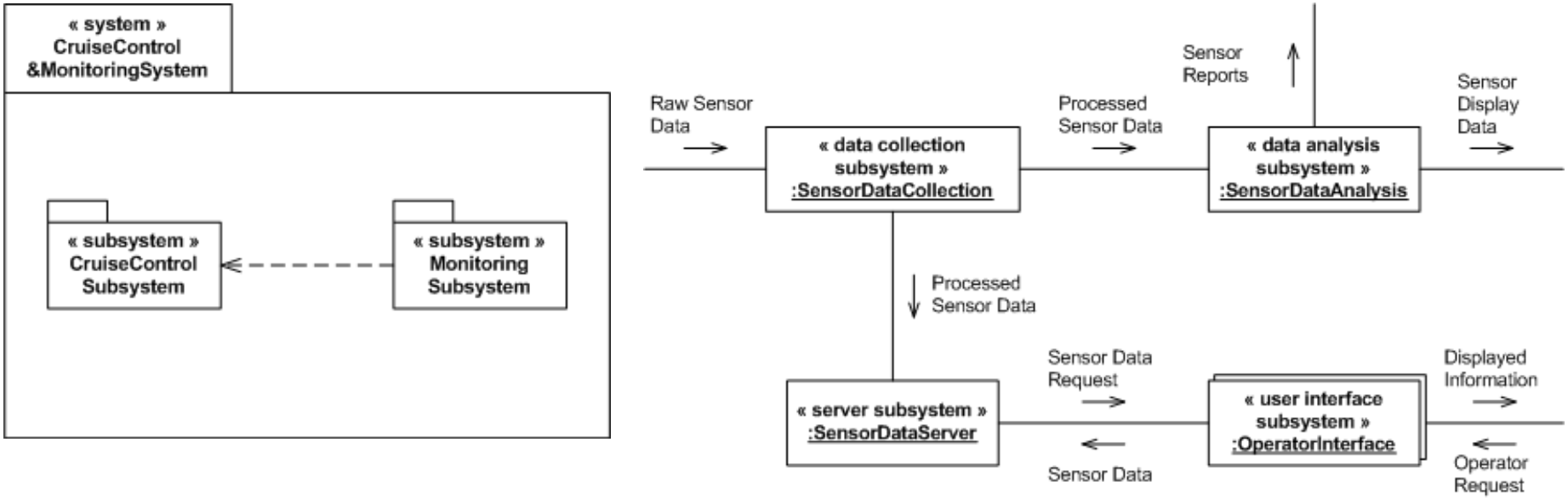
- Subsystems may also be shown on subsystem collaboration diagrams.
- This provides a high-level view of the collaborations within a system.

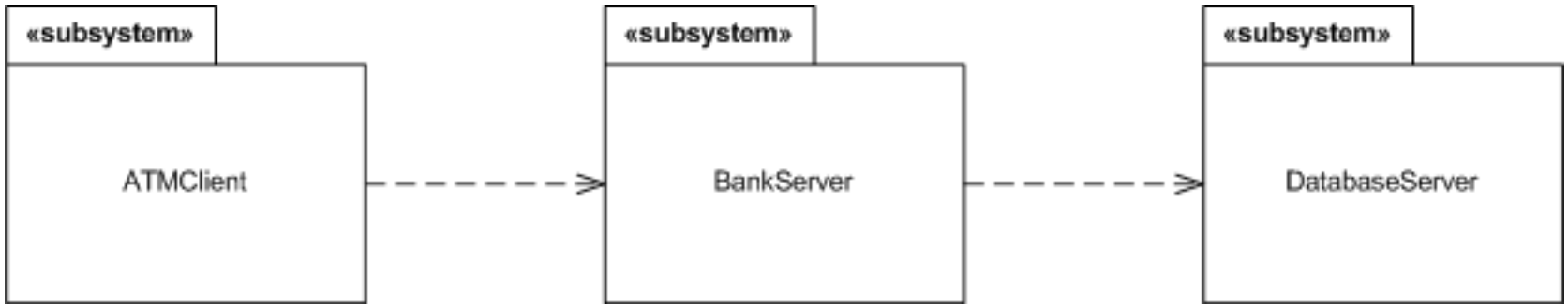


Subsystems in Concurrent, Real-time, and Distributed Applications

- Concurrent, real-time, and distributed applications frequently have the following types of subsystems:
  - Control
  - Coordinator
  - Data Collection
  - Data Analysis
  - Server
  - User Interface
  - I/O Subsystem
  - System Services

Subsystems in Concurrent, Real-time, and Distributed Applications (examples)





Design Analysis Static Modeling

- A more detailed model is developed as part of solution design.
- This static model may either refine the conceptual static model developed in the analysis modeling phase, or may be developed from the consolidated collaboration diagrams.
- Further refinement considers the directional nature of class relationships.

Summary

- Software Architectural Styles may be blended
  - Client/Server
  - Layered
  - Communicating Tasks
- Complex systems may be decomposed into subsystems with loose external coupling and high internal coupling.
- Multiple criteria exist for subsystem decomposition.