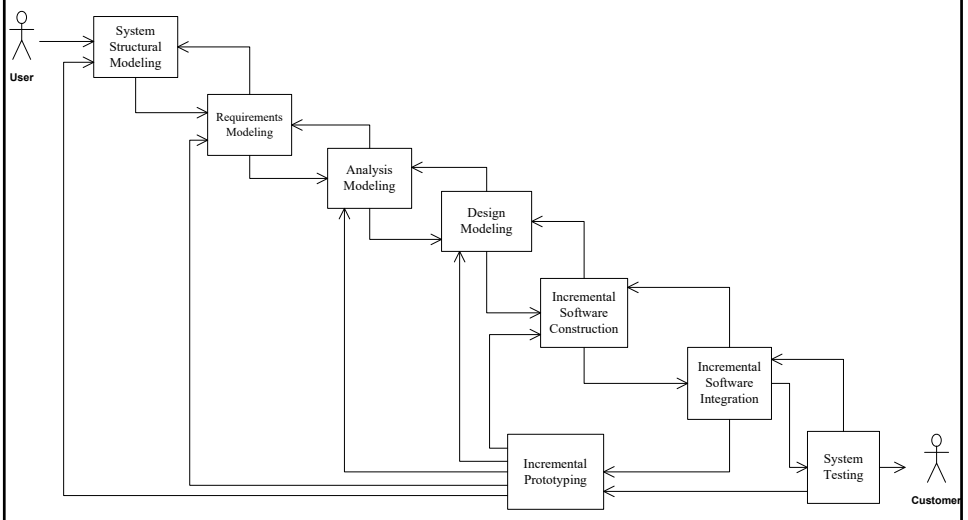
SWE 760

Lecture 7:
Software Architecture for
Real-Time Embedded Systems

Reference:
H. Gomaa, Chapter 10 - *Real-Time Software Design for Embedded Systems*, Cambridge University Press, 2016

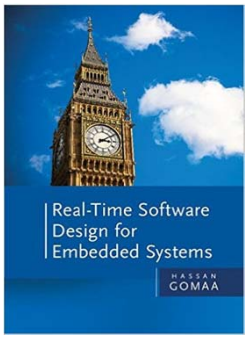
Hassan Gomaa
Dept of Computer Science
George Mason University
Fairfax, VA

Figure 4.1 COMET/RTE life cycle model



Software Modeling and Design for Real-Time Embedded Systems

- COMET/RTE method
  - Software Design for RT embedded systems
  - From Use Case Models to Software Architecture
    - Uses UML, SysML and MARTE notations
  - Requirements and Analysis Modeling
    - Use case modeling
    - Static and Dynamic modeling
  - Design modeling
    - Concurrent, distributed, real-time embedded systems
  - H. Gomaa, *Real-Time Software Design for Embedded Systems*, Cambridge University Press, 2016



Software Modeling for RT Embedded Systems

1 Develop RT Software Requirements Model
  - Develop Use Case Model

2 Develop RT Software Analysis Model
  - Develop state machines for state dependent objects
  - Structure software system into objects
  - Develop object interaction diagrams for each use case

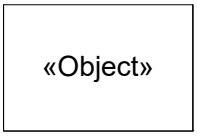
3 Develop RT Software Design Model
  - Design of Software Architecture for RT Embedded Systems
  - Apply RT Software Architectural Design Patterns
  - Design of Component-Based RT Software Architecture
  - Design Concurrent RT Tasks
  - Develop Detailed RT Software Design
  - Analyze Performance of Real-Time Software Designs

Design of RT Software Architecture

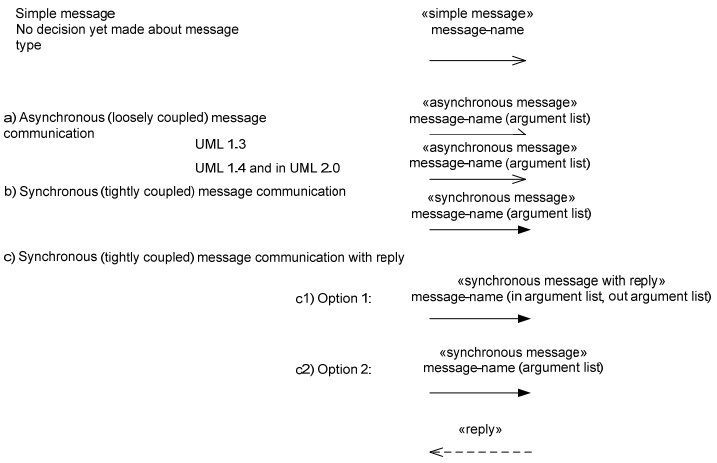
- Software Architecture
  - Structure of software system
    - Software elements (e.g., subsystems or components)
    - Externally visible properties of elements (e.g., component interfaces)
    - Relationships among elements (e.g., connectors)
- Develop initial software architecture
  - Synthesize from communication diagrams
  - Structure system into subsystems
- Subsystems determined using subsystem structuring criteria
  - Use stereotypes for subsystem structuring criteria
    - E.g., <<control>>, <<coordinator>>
  - Depict subsystems on subsystem communication diagrams

Active and Passive Objects

- Objects may be **active** or **passive**
- **Active object**
  - **Concurrent task or component**
  - Has thread of control
- **Passive object**
  - a.k.a. **Information Hiding Object**
  - Has no thread of control
  - Operations of passive object are executed by task

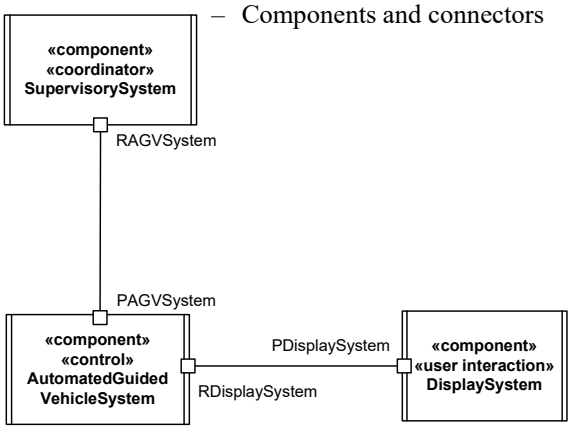


UML notation for messages



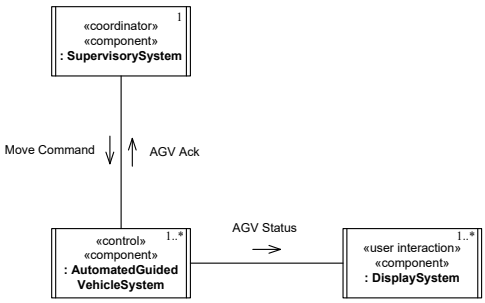
Multiple Views of RT Software Architecture

- Structural view
  - Subsystem structured class diagram
  - Components and connectors



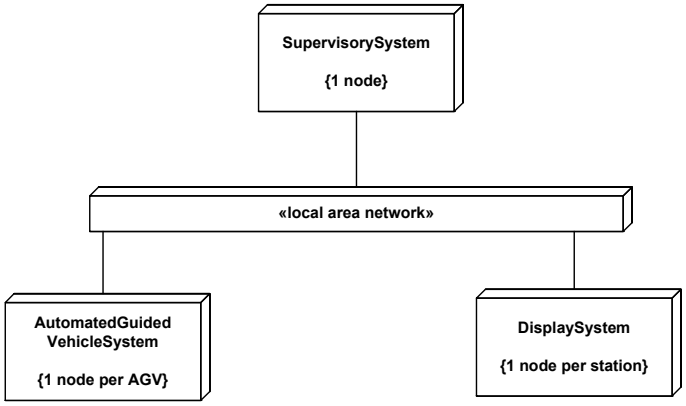
Multiple Views of Software Architecture

- Dynamic view
  - Subsystem communication diagram



Multiple Views of Software Architecture

- Deployment view
  - Physical configuration on deployment diagram



Transition from Analysis to Design: Develop initial software architecture (high level design)

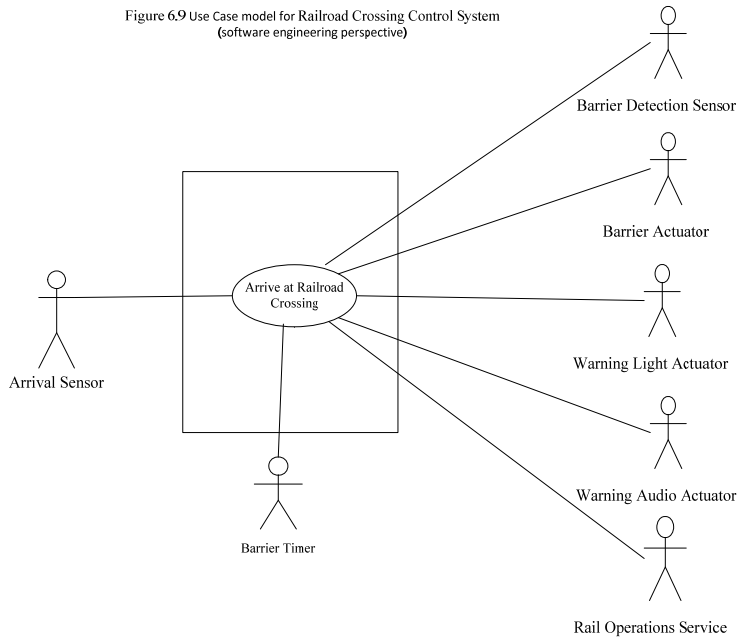
- Start with dynamic interaction model
  - Use case based interaction diagrams
    - Sequence diagrams
    - Communication diagrams
- Integrate use case based interaction diagrams
  - Initial version of software architecture
- Structure system into subsystems
  - Subsystem contains multiple objects
- Depict subystems on subsystem communication diagram
  - High-level communication diagram
  - Shows subsystems and their interactions

Transition from Analysis to Design: Integration of Communication Diagrams

- Used to determine overall structure of system
- Merger of communication diagrams
  - Start with first communication diagram
  - Superimpose other communication diagrams
    - Add new objects and new message interactions from each subsequent diagram
    - Objects and interactions that appear on multiple diagrams are only shown once
    - Consider alternative scenarios for each use case
- Integrated communication diagram
  - Shows all objects and their interactions

Example of Railroad Crossing Control System

Figure 6.9 Use Case model for Railroad Crossing Control System
(software engineering perspective)



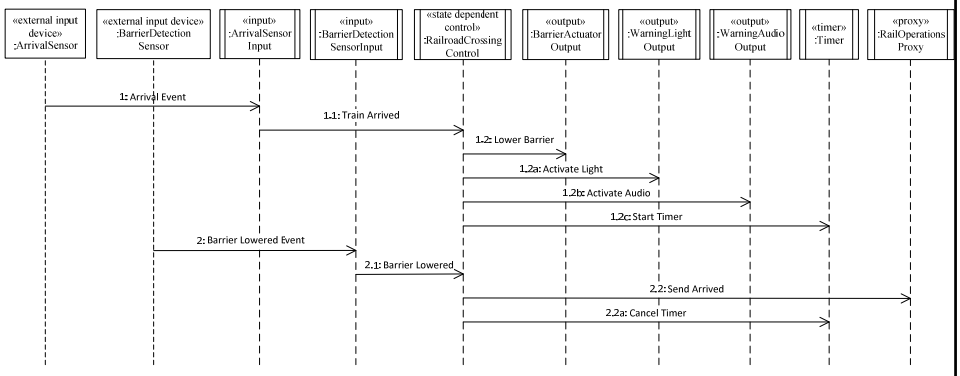


Figure 20.9 Sequence Diagram for Arrive at Railroad Crossing Use Case (software objects)

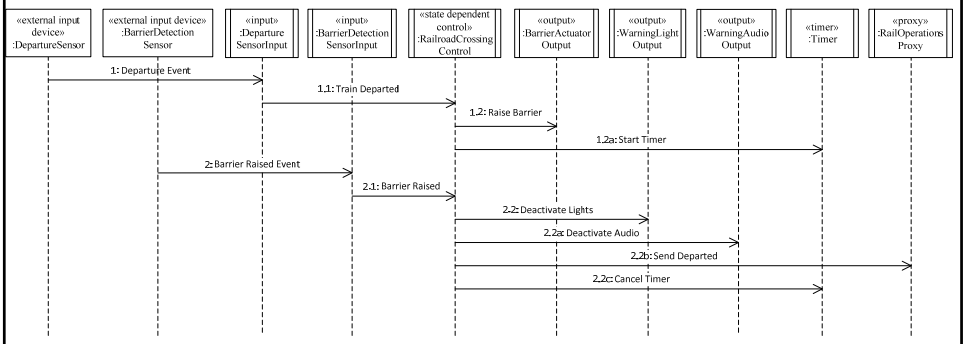


Figure 20.11 Sequence Diagram for Depart from Railroad Crossing Use Case (software objects)

15

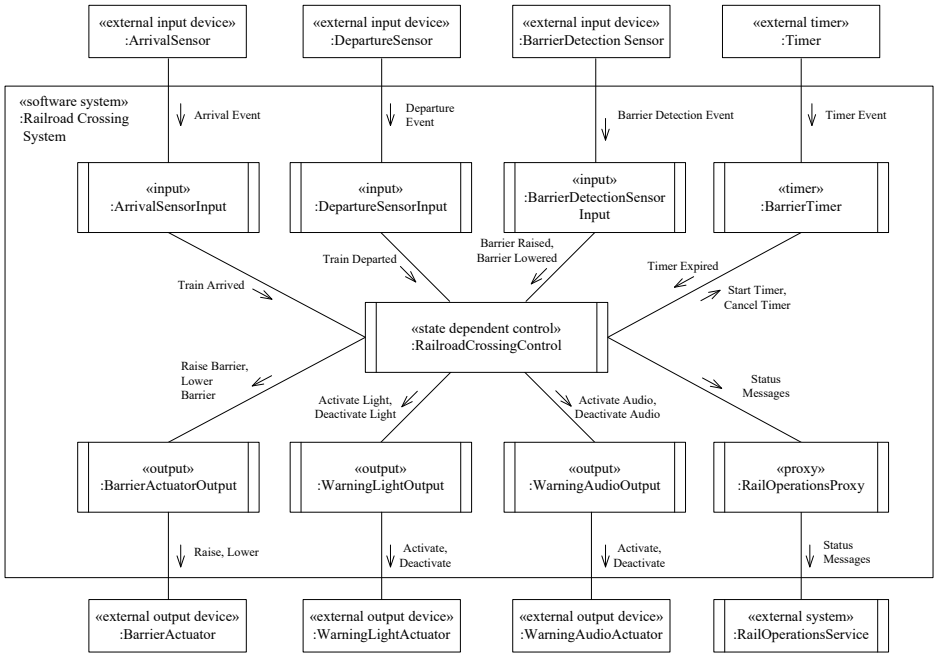


Fig. 10.5 Integrated Communication Diagram for Railroad Crossing System

16

Design of Software Architecture

- Software Architecture
  - Define overall structure of system
    - Component interfaces and interconnections
  - Separately from component internals
- Each subsystem performs major service
  - Contains highly coupled objects
  - Relatively independent of other subsystems
  - May be decomposed further into smaller subsystems
  - Subsystem is aggregate or composite object
- Apply subsystem structuring criteria
  - Subsystems can be designed as components
  - Distributed Component-based software architectures

Separation of Subsystem Concerns

- **Aggregate/composite object**.
  - Objects that are part of aggregate/composite object
  - Structure in same subsystem (e.g., Fig. 10.6)
- **Interface to external objects**
  - External real-world object should interface to 1 subsystem (e.g., Fig. 10.8, 21.9)
- **Scope of Control**
  - Control object & objects it controls are in same subsystem (e.g., Fig. 10.5)
- **Geographical location**
  - Objects at different locations are in separate subsystems (e.g., Fig. 10.4, 10.9)
- **Clients and Services**
  - Place in separate subsystems (e.g., Fig. 10.4, 10.11, 10.12)
- **User Interaction**
  - Separate client subsystem (e.g., Fig. 10.4, 10.7, 10.10)

Figure 10.6 Example of a composite class

Figure 10.6a Composite class

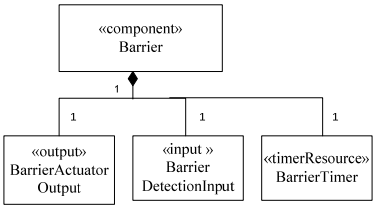
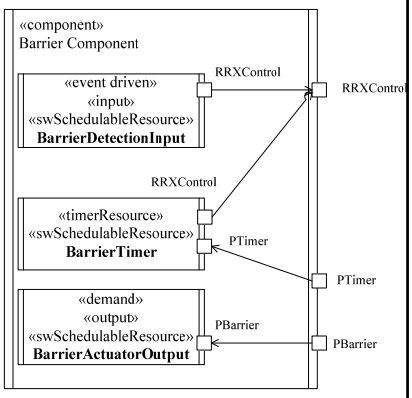


Figure 10.6b Composite component



19

Example of Interfacing to External Classes (Fig 10.8)

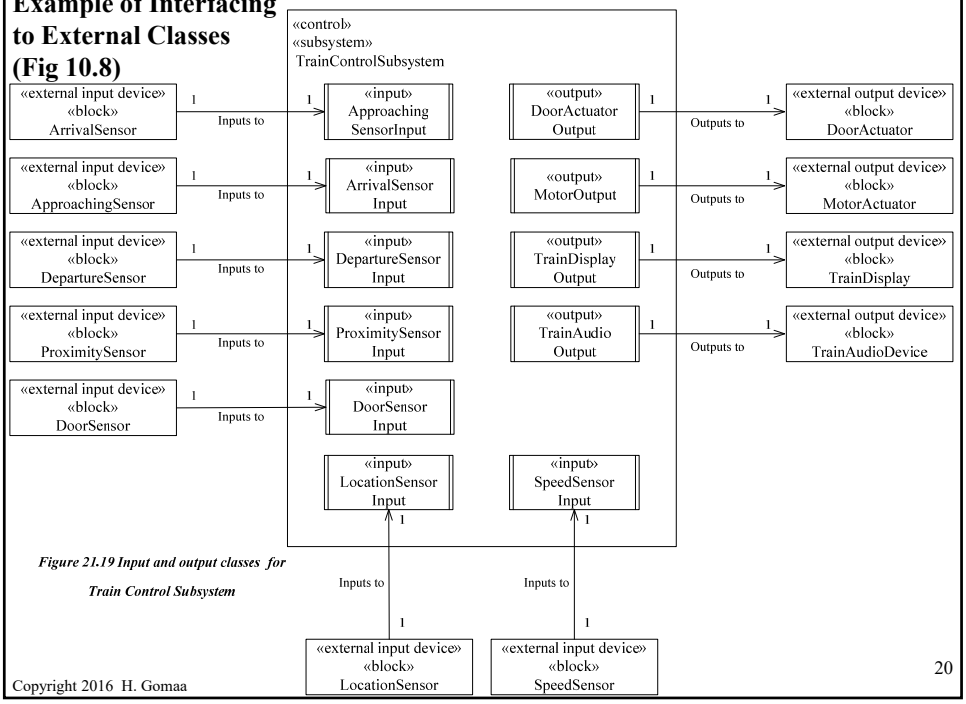
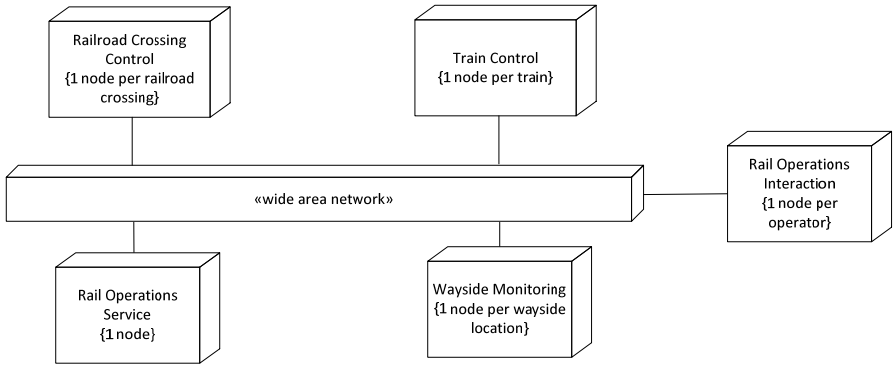


Figure 21.19 Input and output classes for Train Control Subsystem

20

Examples of Separation of Subsystem Concerns

- Example from Distributed Light Rail Embedded System (Fig. 10.4)
- Physically deploy hardware and software components to
  - Geographically Distributed platforms
- Client and Service Subsystems



Subsystem Structuring Criteria

- Control
  - Subsystem controls given part of system (e.g., Fig. 10.9)
- Coordinator
  - Coordinates several control subsystems (e.g., Fig. 10.9)
- Input / Output
  - Performs I/O operations for other subsystems (e.g., Fig. 10.12, 21.30)
- User Interaction
  - Collection of objects supporting needs of user (e.g., Fig. 10.7, 10.10)
- Service
  - Provides service for client subsystems (e.g., Fig. 10.10, 10.12)
- Data Collection
  - Collects data from external environment (e.g., Fig. 10.11)
- Data analysis
  - Provides reports and/or displays (e.g., Fig. 10.11)

Example of Control and Coordinator Subsystems (Fig 10.9)

- Multiple instances of control subsystem
- One coordinator subsystem

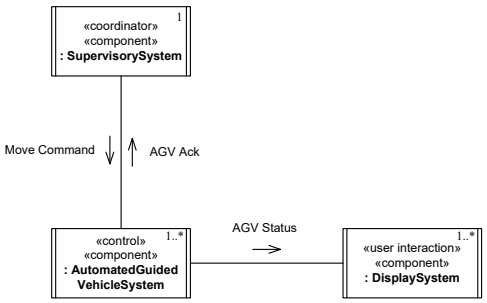


Figure 10.12 Example of client and service subsystems in Emergency Monitoring System

Example of Input, User Interaction, Client and Service Subsystems

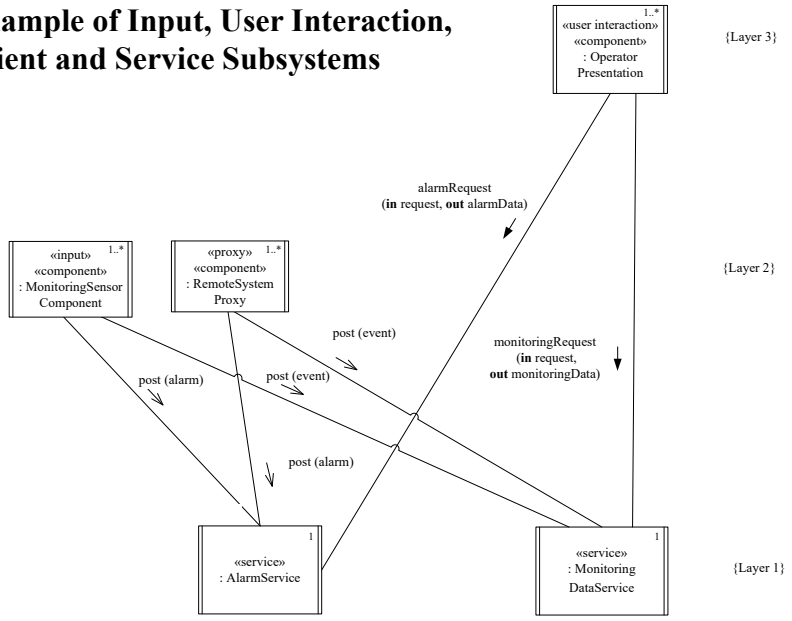


Figure 10.7 Example of User Interaction Subsystem: Operator Presentation component

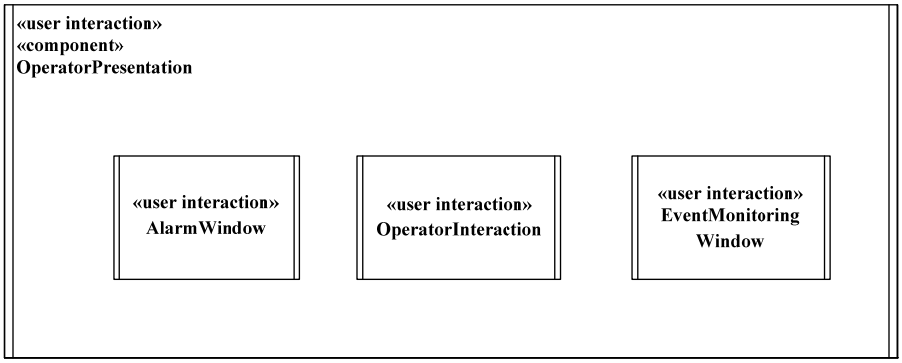


Figure 10.10 Example of user interaction subsystem with multiple windows

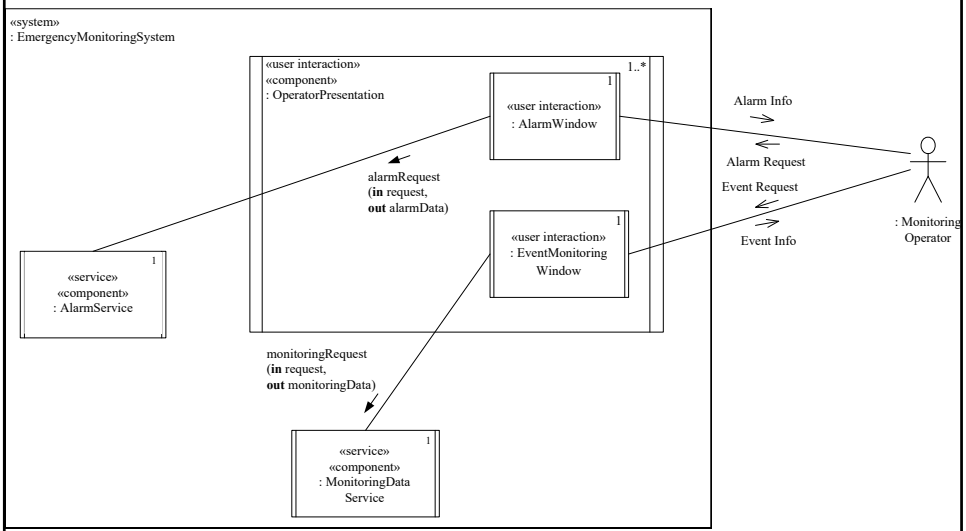
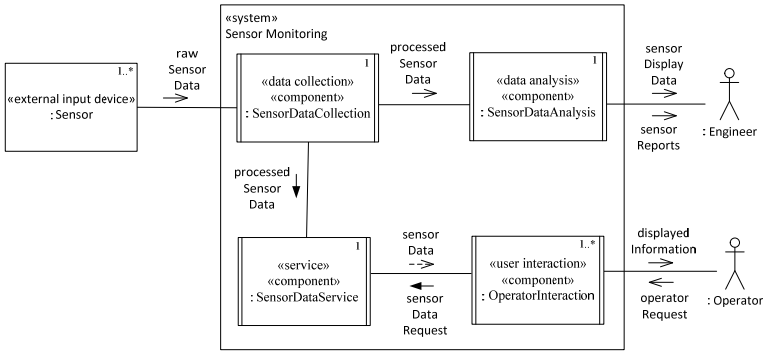


Figure 10.11: Examples of data collection and data analysis subsystems



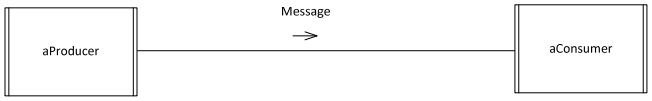
Design Message Communication Between Subsystems

- Message Communication between distributed subsystems
  - Asynchronous message communication
    - Peer to peer communication
  - Synchronous message communication
    - Client / Service message communication
  - Group Message Communication
    - Broadcast message communication
    - Multicast message communication
  - Brokered Communication
    - Uses Object Broker
- Also referred to as message communication patterns
  - See lecture 8

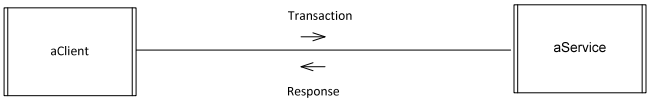
Design Message Communication Between Subsystems

Figure 10.13a Analysis Model – before decisions about type of message communication

(1) Unidirectional message communication between producer and consumer



(2) Bidirectional message communication between clients and service

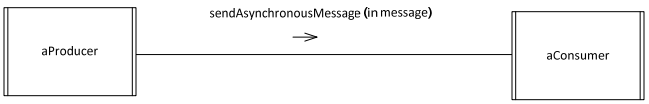


- During Object Structuring
  - Assume all objects are concurrent EXCEPT entity objects
  - Assume all communication between concurrent objects is asynchronous
- These initial decisions can be changed during RT design

Design Message Communication Between Subsystems

Figure 10.13b Design Model – after decisions about type of message communication

(3) Asynchronous message communication between producer and consumer



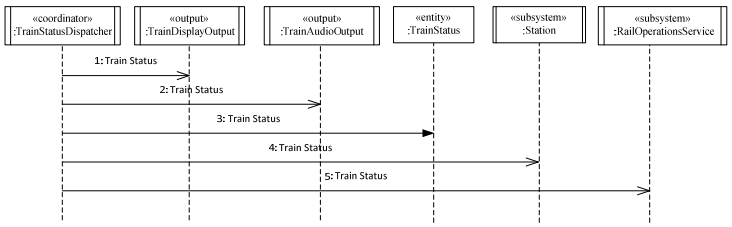
(4) Synchronous message communication between clients and service

Example of RT Design of Software Architecture - Light Rail Control System

- Automated driverless trains
- Travel between stations along a track in both directions with a circular loop at each end.
- Trains stop at each station.
- Approaching, arrival, and departure sensors
- Sensors to detect location and speed of train
- Actuators to control train motor and doors
- Proximity sensor detects hazards ahead
- Automated acceleration, deceleration, speed control
- Train display and audio system
- Each station has display and audio system
- External systems send status to LRCS
- See Lecture 6 notes
  - Figs 21.4 – 21.6 for use case model,
  - Figs 21.18-21.20 for static structural model
  - Figs 21.21-21.28 for Dynamic Interaction Model

Dynamic Interaction Model (cont'd)

Figure 21.23 Sequence diagram for Train Status Dispatcher



Design Modeling
Integration of Communication Diagrams

- Integrated communication diagram
  - If there are too many objects for one integrated communication diagram
  - Develop subsystem communication diagram
  - Develop integrated communication diagram for each subsystem
- Subsystem communication diagram
  - High-level communication diagram
  - Shows subsystems and their interactions

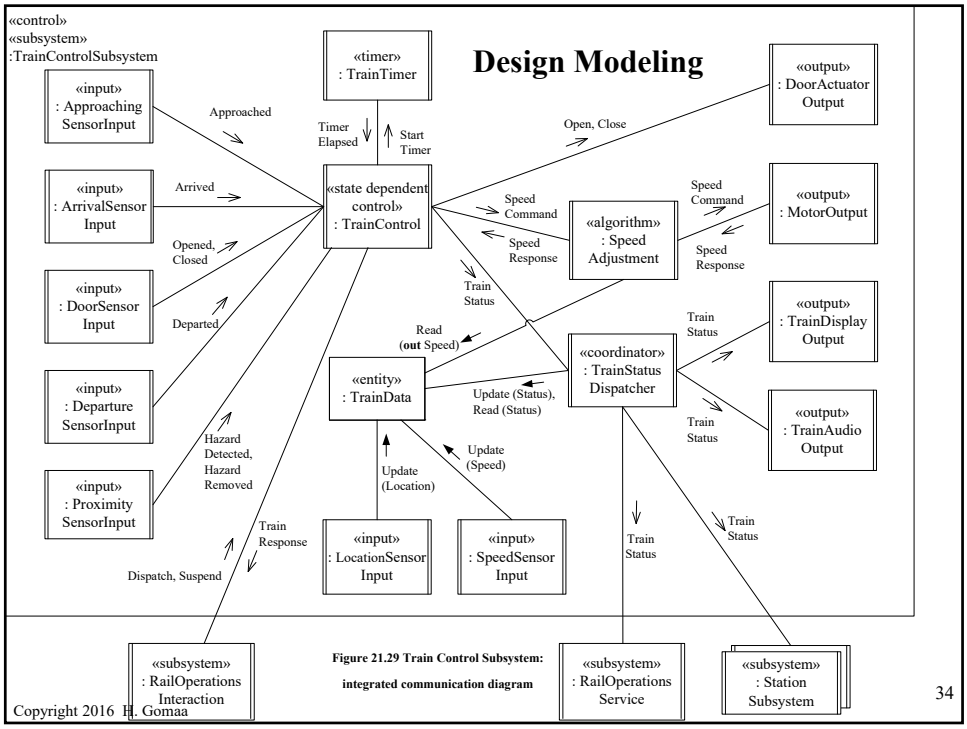


Figure 21.29 Train Control Subsystem: integrated communication diagram

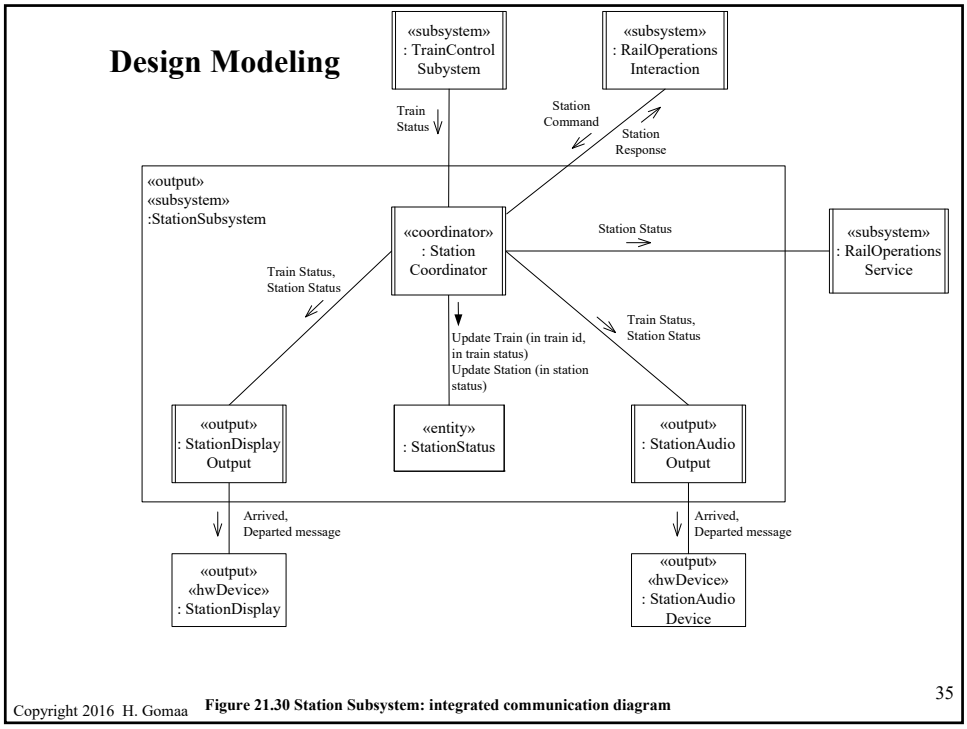
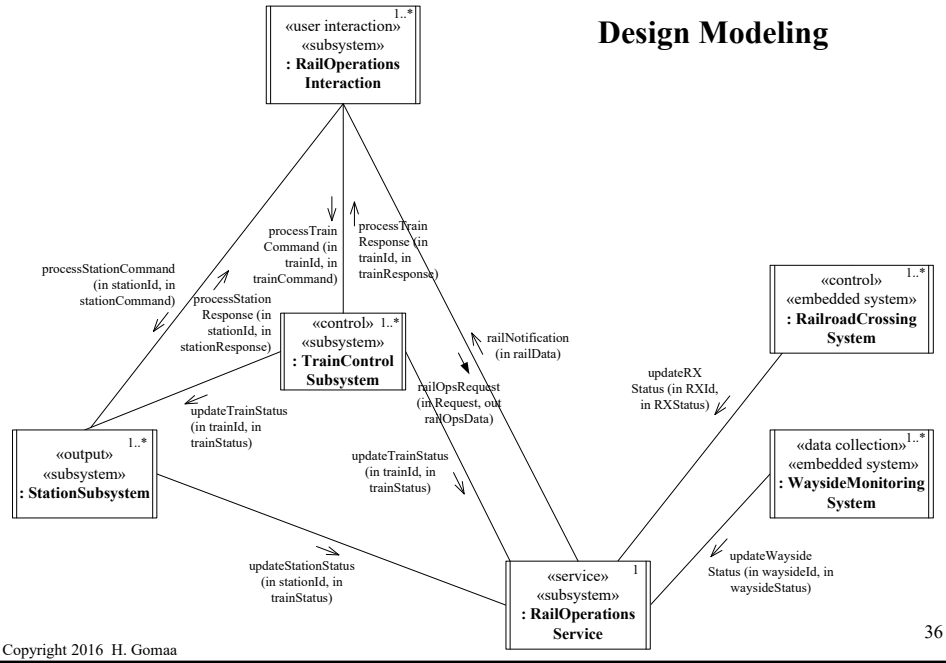


Figure 21.30 Station Subsystem: integrated communication diagram

Figure 21.32 Distributed software architecture for Light Rail System



Summary
Design of RT Software Architecture

- Software Architecture
  - Structure of software system
    - Software elements (e.g., subsystems or components)
    - Externally visible properties of elements (e.g., component interfaces)
    - Relationships among elements (e.g., connectors)
- Develop initial software architecture
  - Synthesize from communication diagrams
  - Structure system into subsystems
- Subsystems determined using subsystem structuring criteria
  - Use stereotypes for subsystem structuring criteria
    - E.g., <<control>>, <<coordinator>>
  - Depict subsystems on subsystem communication diagrams

Figure 4.1 COMET/RTE life cycle model