SWE 760

Lecture 9:
Component-based Software Architectures for Real-Time Embedded Systems

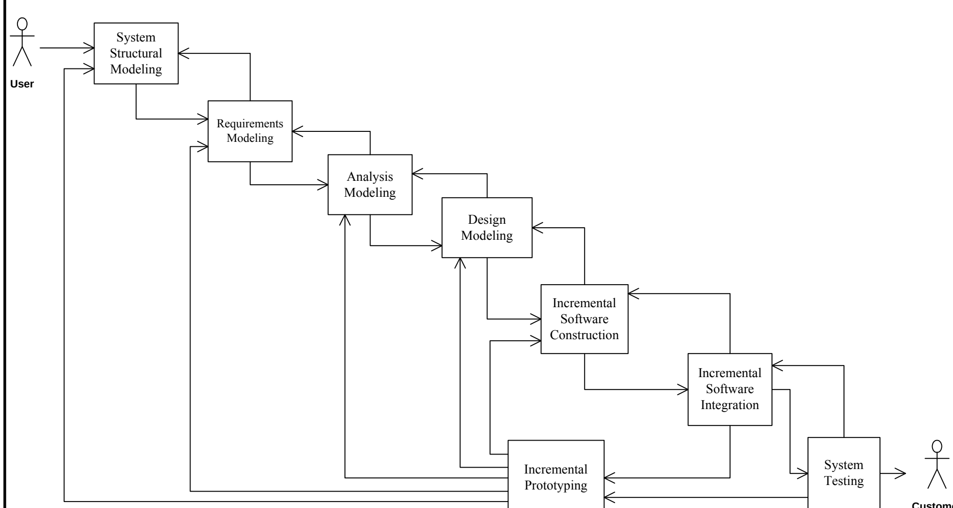
Reference:

H. Gomaa, Chapter 12 - *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

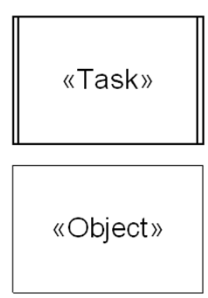


Software Modeling for RT Embedded Systems

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

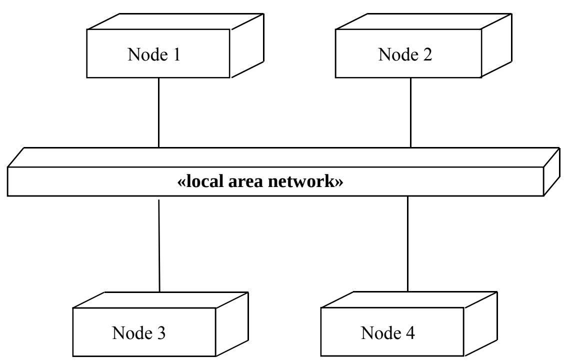
Active and Passive Objects

- 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



Architectural Design of Distributed Applications

- Distributed processing environment
  - Multiple computers communicating over network
- Typical applications
  - Distributed real-time data collection
  - Distributed real-time control
  - RT Client / Service applications
- COMET/RTE for Distributed RT Applications
  - Addresses structuring RTE application into distributed subsystems



Distributed processing environment

Characteristics of Distributed Applications

- Structure of component-based distributed application
  - Consists of one or more component-based subsystems
  - Each subsystem designed as a distributed component
  - Execute on multiple nodes in distributed configuration
- Component
  - Concurrent self-contained object with a well-defined interface, capable of being used in different applications from that for which it was originally designed
- Structure of component-based subsystem
  - Consists of one or more objects
  - Objects all execute on same node
- Communication between component-based subsystems
  - Message communication

Steps in Designing Distributed Component-based Software Architectures

- Design software architecture
  - Structure architecture into distributed subsystems
    - Each subsystem designed as composite component
  - Define message communication interfaces
- Design constituent components
  - Structure composite component into simple components
  - Simple component can execute on only one node
- Deploy software components
  - Define component instances
  - Instantiate and deploy to hardware configuration
    - Distributed physical nodes

Component-based Software Architecture

- Executes on multiple nodes in distributed configuration
  - Consists of distributed components
- Distributed component
  - Well-defined *provided* and *required* interfaces
  - Concurrent object
  - Logical unit of distribution and deployment
  - Communicates with other components using messages
  - Structure
    - Composite object consisting of other objects
    - Simple object
  - Capable of being reused

Component Interface Design

- Interface
  - Externally visible operations of a class, service, or component
- UML notation
  - Interface can be modeled separately from component
  - Two ways to depict (simple and expanded)
- Component can provide one or more interfaces
  - Use different interfaces if clients require different services
- Component can require one or more interfaces
- Component realizes an interface
  - Provides implementation of interface

Figure 2.15 UML notation for components and interfaces

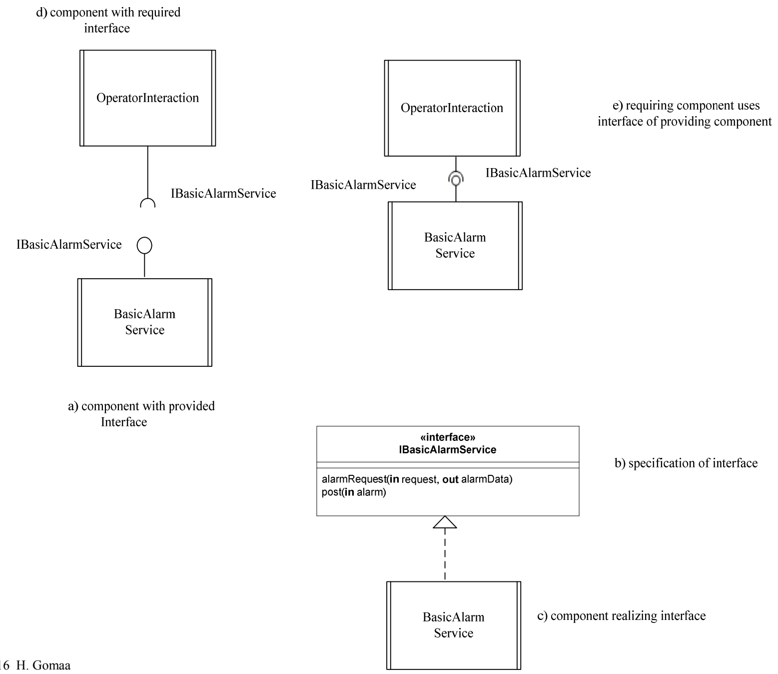
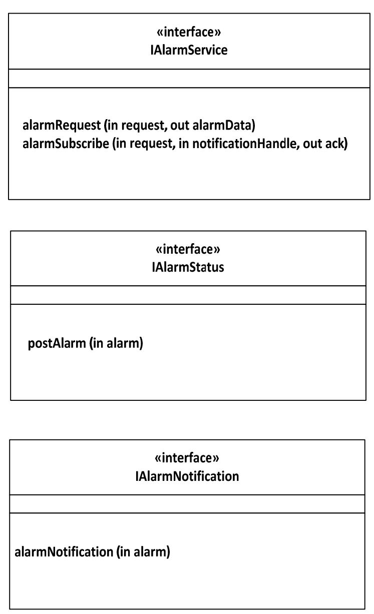


Figure 12.1: Example of component interfaces and provided operations: Emergency Monitoring System



Modeling Component Interfaces

- Component have *provided* and *required* interfaces
- **Provided interface**
  - Specifies operations that a component must fulfill
- **Required interface**
  - Specifies operations that other components provide for this component
- **Components**
  - Communicate with each other through **ports**
- **Port**
  - Consists of *provided* and/or *required* interfaces
- **Connector**
  - Joins *required* port of one component to *provided* port of another component

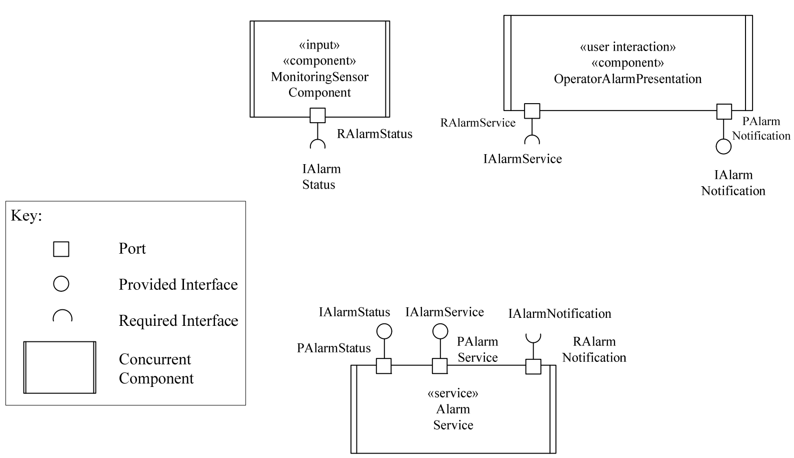
Example of Component Interfaces

- Alarm Service provides 2 interfaces, requires 1interface
  - Provided interfaces
    - **IAlarmService** interface to receive alarm requests and subscriptions
    - **IAlarmStatus** interface to receive new alarms
  - Required interface
    - **IAlarmNotification** interface to send alarm notifications
- Operator Alarm Presentation component
  - Required interface
    - **IAlarmService** interface to make alarm requests and subscriptions
  - Provided interface
    - **IAlarmNotification** interface to receive alarm notifications
- Monitoring Sensor component
  - Required interface
    - **IAlarmStatus** interface to post new alarms

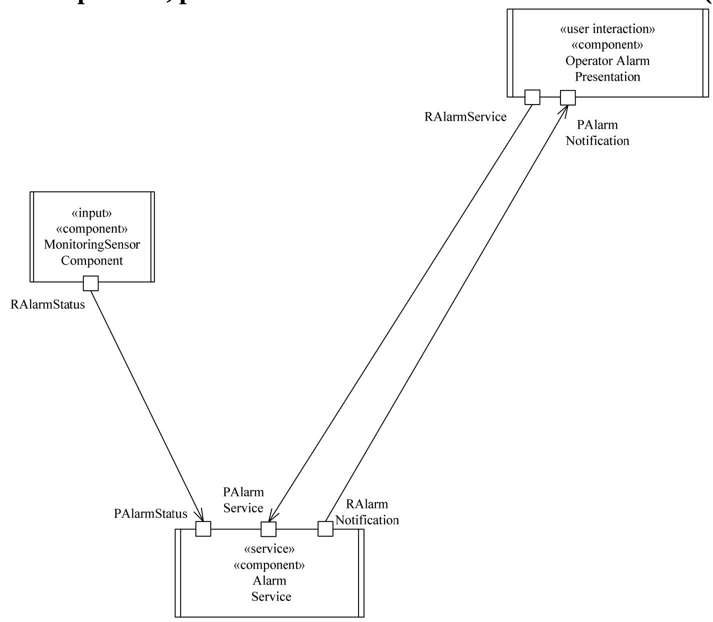
Modeling Components in UML 2

- Components
  - Modeled as UML 2 structured classes
  - Depicted on UML 2 composite structure diagrams
- Component *provided* and *required* interfaces are explicitly modeled
- **Components**
  - Communicate with each other through **ports**
- **Ports and interfaces**
  - **Provided Port** supports *provided* interface
  - **Required Port** supports *required* interface
  - **Complex Port** supports both *provided* and *required* interfaces

Example of ports, provided, and required interfaces in UML 2 (Fig. 12.2)



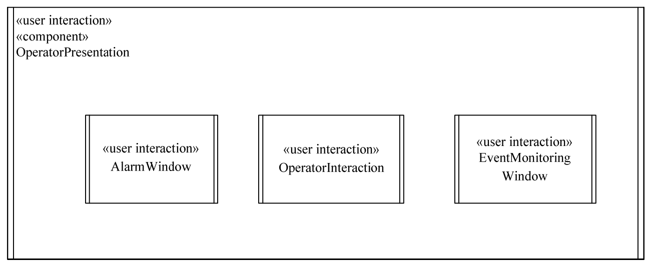
Example of components, ports and connectors in software architecture (Fig. 12.3)



Design of Composite Components

- Composite Component
  - Component that encapsulates internal components
  - Both a logical and physical container
  - Functionality provided entirely by components it contains
  - Internal components referred to as
    - Constituent, nested, or part components
- Structure of Composite Component
  - Structured into part components
  - Part components are depicted as instances
  - Can have more than one instance of part component
- Simple component
  - Component with no internal components

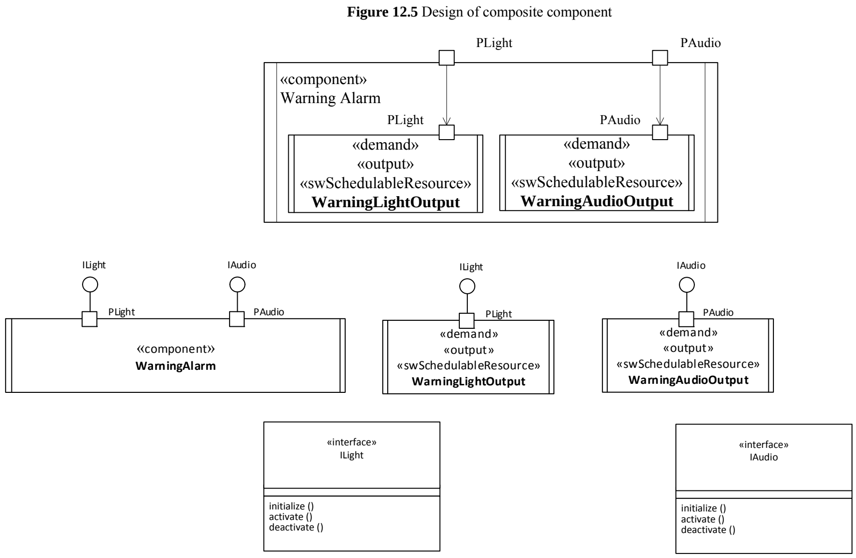
Example of Composite Component



Design of Composite Component

- Delegation connector (provided port to provided port)
  - Provided port of composite (outer) component connected to
    - Provided port of nested (inner) component
    - Both provided ports are given the same name
  - Operation of outer component calls
    - Operation of inner component
- Delegation connector (required port to required port)
  - Required port of nested (inner) component connected to
    - Required port of composite (outer) component
    - Both required ports are given the same name
  - Operation of inner component calls
    - Operation of outer component

Design of Composite Component



Component Structuring Criteria

- Proximity to source of physical data and/or component
  - Ensures fast access to physical data
  - Physically located with hardware component
  - E.g., Barrier component (Fig. 12.9)
- Localized autonomy
  - Performs specific site related function
  - Same function performed at multiple sites
  - Each instance of component resides on separate node
  - Operational if other nodes temporarily unavailable
  - E.g., Light Rail System (Fig. 12.10)

Figure 12.9 Example of component proximity to source of local data: Barrier Component

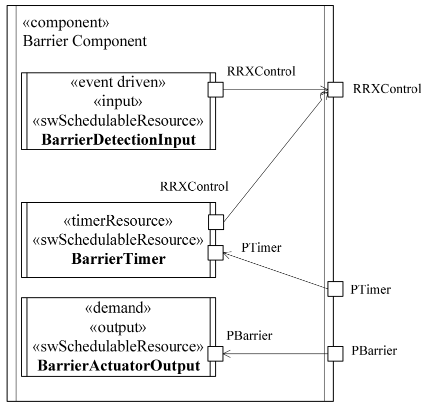
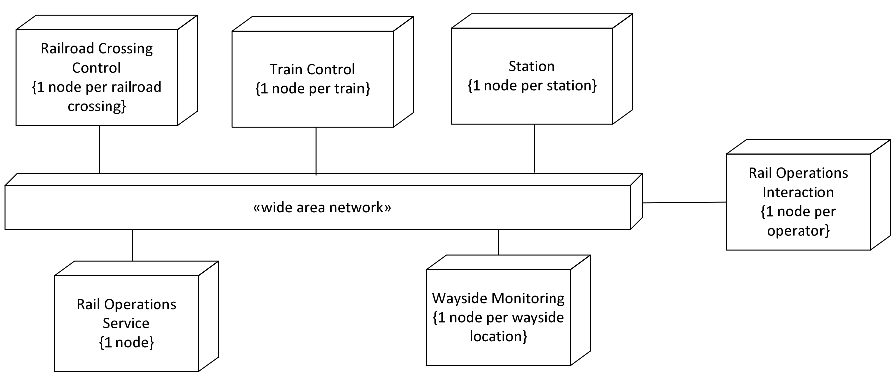


Figure 12.10 Examples of component localized autonomy and control: Deployment of Light Rail System



Subsystem Configuration Criteria

- Performance
  - Provides time critical function
  - More predictable performance,
  - E.g., Train Control (Fig. 12.10)
- Specialized Hardware
  - Node interfaces to special purpose hardware (Fig. 12.10)
  - E.g., Interface to special purpose sensors and actuators
- I/O component
  - Smart device (hardware + software)
  - Interacts with external environment
  - Input, Output, Input and Output (I/O), Network Interface
  - E.g., Barrier Component (Fig. 12.9)

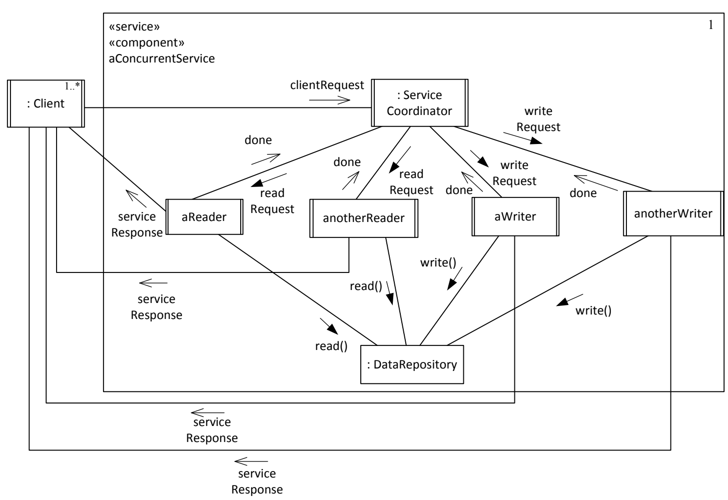
Design of Service Components - Sequential Service Component

- Receives message requests from clients
  - One message type for each service type
- Sequential Service designed as one component
  - Services client requests sequentially
  - Service completes one request before starting next
  - E.g., Rail Operations Service (Fig. 12.10)
- Service Coordinator
  - Acts as service stub
  - Unpacks incoming message
  - Invokes service operation
  - Packs response in service response message

Design of Service Components - Concurrent Service

- Service functionality shared among several concurrent objects
  - Service Coordinator coordinates activities
- Synchronization algorithm is needed
  - Mutual exclusion
    - Only one reader or writer may access data repository at any one time
  - Multiple readers and writers algorithm (Fig. 12.11)
    - Multiple readers access shared data repository concurrently
    - Only one writer can updates data repository at any one time

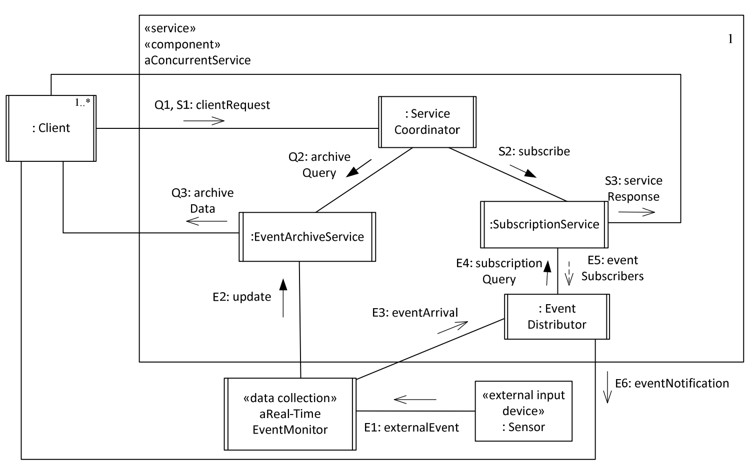
Figure 12.11: Example of concurrent component - multiple readers and writers



Design of Service Component - Subscription and Notification

- Real-Time Event Monitor receives external events
  - Records external events of interest
- Subscription service
  - Maintains subscription list of clients that wish to be notified of monitored events
- Client subscribes to Subscription service
  - Fig. 12.12, S prefix
  - Client requests to be notified of events of a given type
- When significant event occurs (Fig 12.12, E prefix)
  - Real-Time Event Monitor updates event archive
  - Sends message to Event Distributor
  - Event Distributor multicasts event notification to clients on subscription list

Figure 12.12: Example of concurrent component - subscription/notification



Software Deployment

- Define instances of component
  - Each component instance has unique name
  - Component parameters need to be defined
    - E.g., sensor names, sensor limits, alarm names
- Map component instances to physical nodes
  - Depict physical configuration of component instances on deployment diagram
- Interconnect component instances
  - One-to-one inter-component communication
  - One-to-many inter-component communication
  - Many-to-one inter-component communication
- Examples – Figs.12.10, 12.13

Figure 12.10 Examples of software deployment: Deployment of Light Rail System

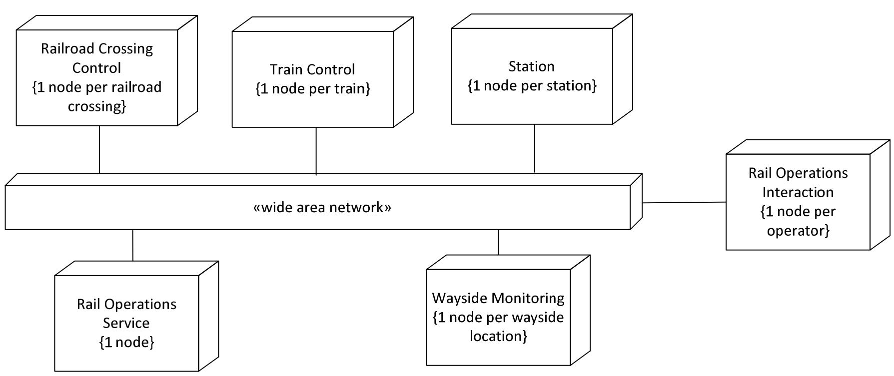
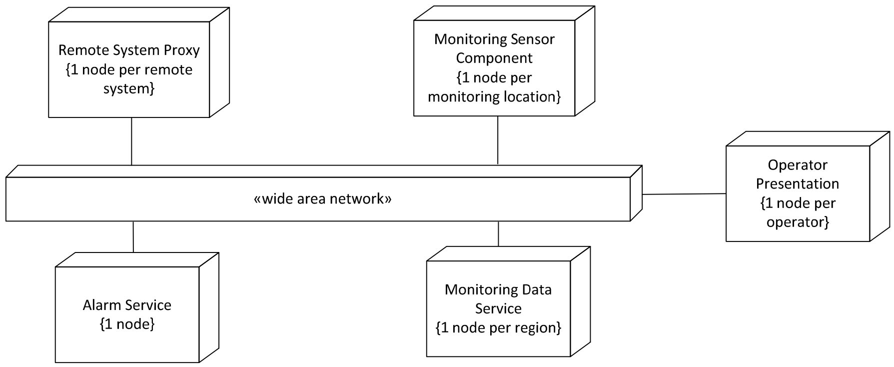


Figure 12.13 Example of software deployment: emergency monitoring system

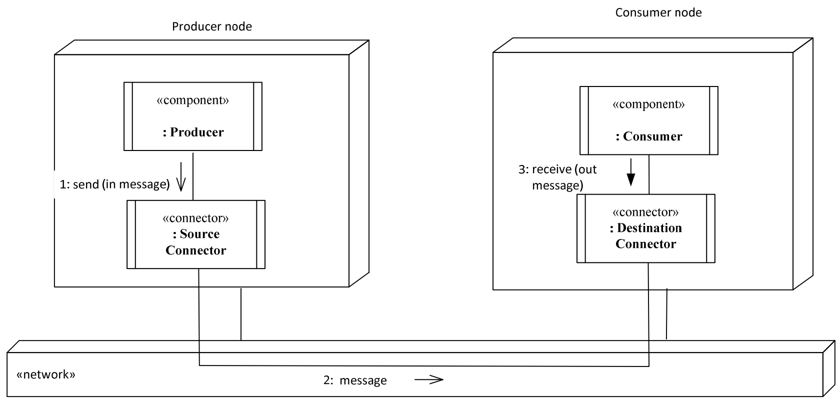


Design of Software Connectors

- Encapsulates details of communication between components
- Producer component on one node
  - Communicates with Consumer component on other node
- Connector Design is distributed
  - Source Connector on Producer node
  - Destination Connector on Consumer node
- Connector Design is based on message communication pattern
  - Asynchronous message communication
  - Synchronous message communication without Reply
  - Synchronous message communication with Reply

Example of Software Connector

Figure 12.14: Example of software connector design



Design of Distributed Message Connectors

- Asynchronous message communication (Fig. 12.14)
  - Distributed message queue connector
  - Source Connector
    - Encapsulates outgoing message queue
    - Provides *send (in message)* operation
  - Destination Connector
    - Encapsulates incoming message queue
    - Provides *receive (out message)* operation

Design of Distributed Message Connectors

- Synchronous message communication without Reply
  - Distributed message buffer connector (Fig. 12.14)
  - Source Connector
    - Encapsulates outgoing message buffer
    - Provides *send (in message)* operation
  - Destination Connector
    - Encapsulates incoming message buffer
    - Provides *receive (out message)* operation

Design of Distributed Message Connectors

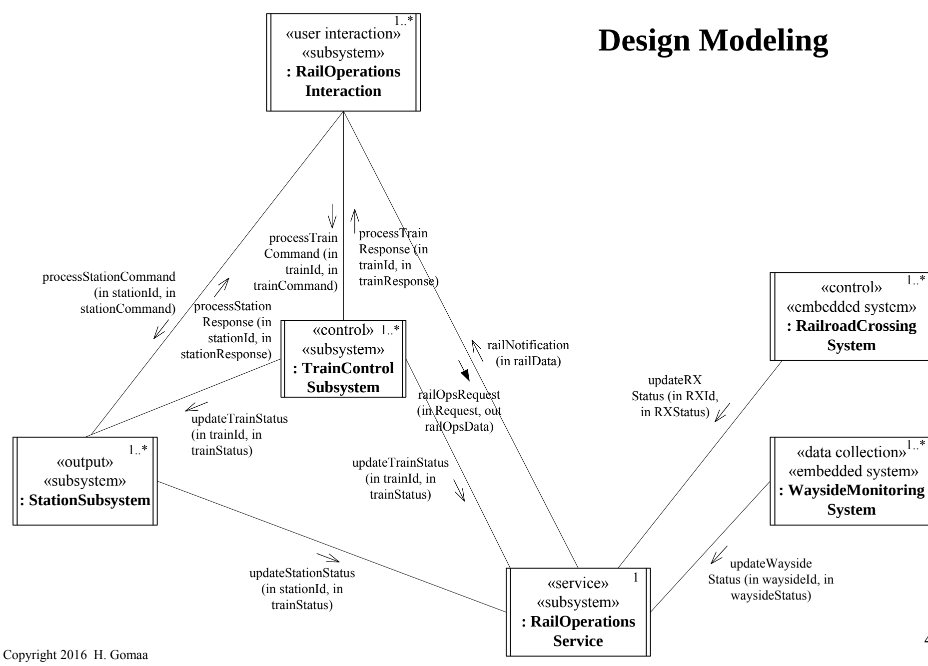
- Synchronous message communication with Reply
  - Distributed message buffer and response connector
  - Source Connector
    - Encapsulates
      - Outgoing message buffer
      - Incoming response buffer
    - Provides *send (in message, out response)* operation
  - Destination Connector
    - Encapsulates
      - Incoming message buffer
      - Outgoing response buffer
    - Provides two operations
      - *receive (out message)*
      - *reply (in response)*

Case Study:
Light Rail Control System

- Example of Component-based Software Design
  - Fig. 21.32 - Software Architecture for Distributed Light Rail Control System
    - Concurrent communication diagram
  - Fig. 21.36 - Component-based Software Architecture for Distributed Light Rail System
    - Composite structure diagram
    - Components, ports, and connectors
  - Fig. 21.37 – Component ports and interfaces
    - Provided and required interfaces for each component
  - Fig. 21.38 – Design of component interface specifications
    - Design of operations provided by each interface

Figure 21.32 Distributed software architecture for Light Rail System

Design Modeling



Component-based Software Design

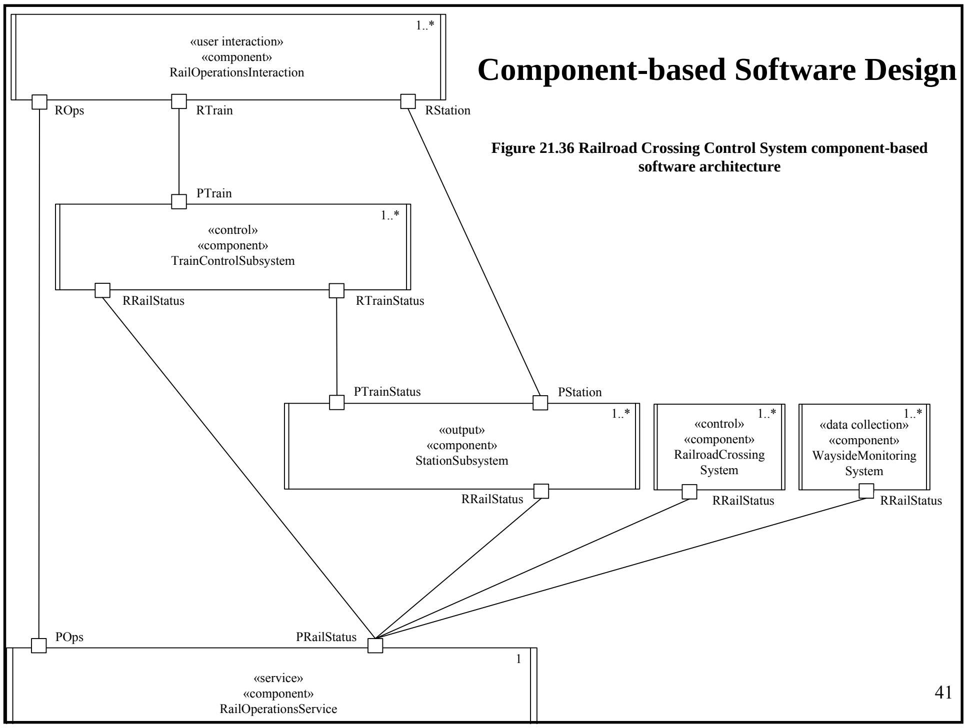
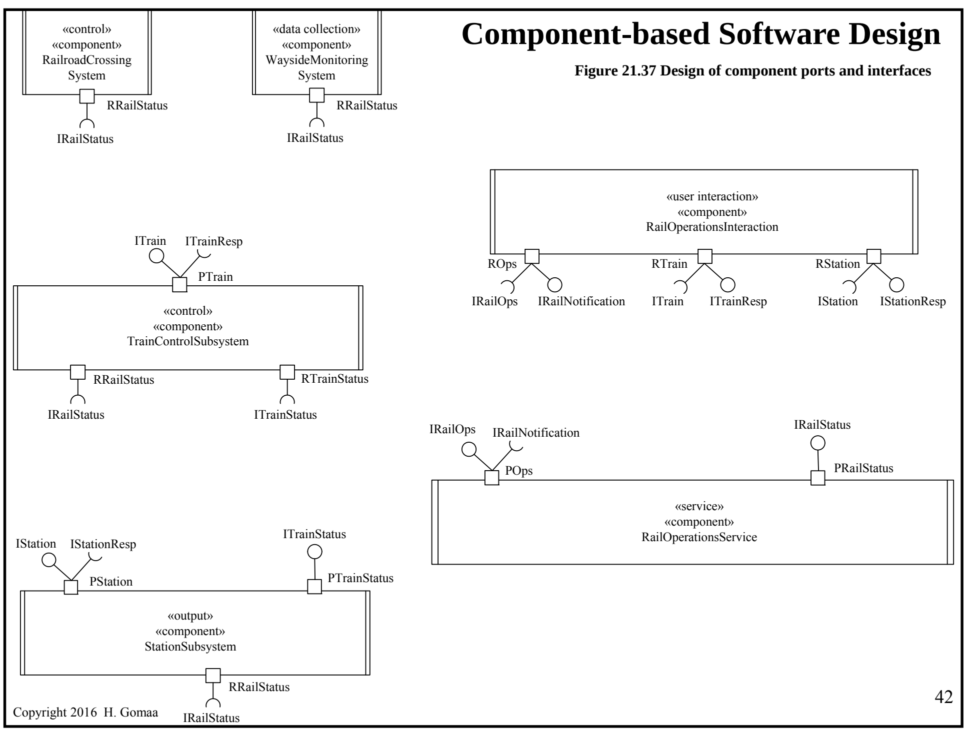


Figure 21.36 Railroad Crossing Control System component-based software architecture

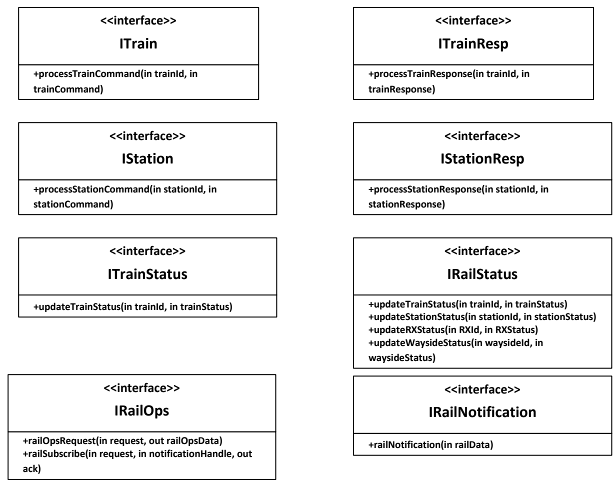
Component-based Software Design

Figure 21.37 Design of component ports and interfaces



Component-based Software Design

Figure 21.38 Design of component interface specifications



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