SWE 760

Lecture 10 –
Concurrent Real-Time Software Task Design

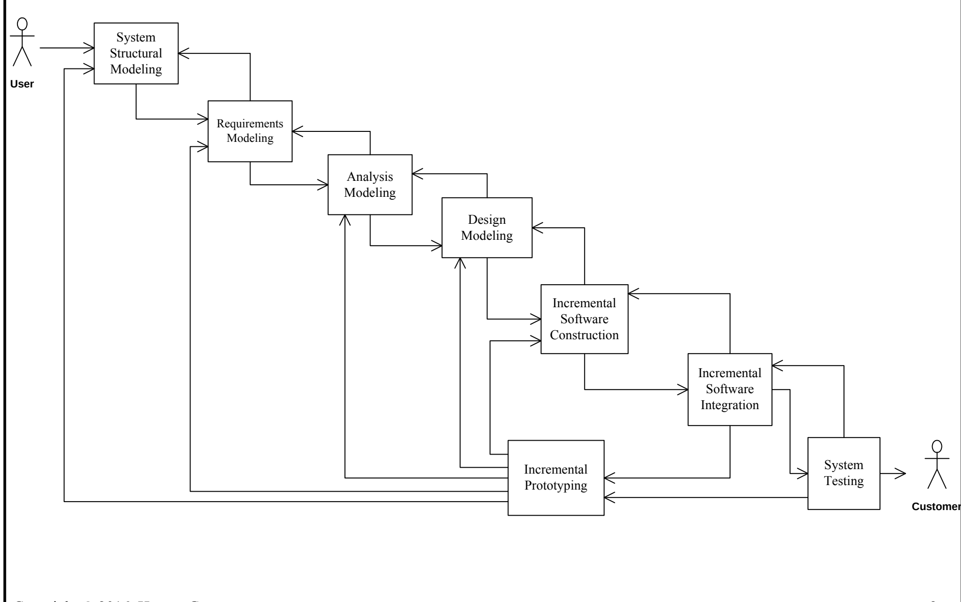
Hassan Gomaa

Dept of Computer Science
George Mason University
Fairfax, VA

Reference:

H. Gomaa, Chapter 13 - *Real-Time Software Design for Embedded Systems*, Cambridge University Press, 2016

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

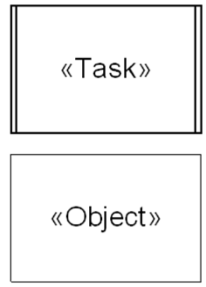
3 Develop RT Software Design Model
  - Design of Software Architecture for RT Embedded Systems
  - Apply RT Software Architectural Design Patterns
  - Design of Component-Based RT Software Architecture
  - **Design Concurrent RT Tasks**
  - Develop Detailed RT Software Design
  - Analyze Performance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

Structure System into Tasks

- Concurrent Design with UML
- Concurrent task structuring criteria
  - Structure analysis model into concurrent tasks
    - Task is an active object
    - Task has thread of control
  - Consider concurrent nature of system activities
  - Determine concurrent tasks
- Define task interfaces
- Support for concurrent tasks
  - Operating system services: multi-tasking kernel

Active and Passive Objects

- Objects may be **active** or **passive**
- **Active object**
  - **Concurrent Task**
  - Has thread of control
- **Passive object**
  - a.ka. **Information Hiding Object**
  - Has no thread of control
  - Operations of passive object are executed by task
  - Operations execute in task's thread of control
    - Directly or indirectly
- Software Design terminology
  - **Task** refers to active object
  - **Object** refers to passive object



UML notation for messages

Simple message
No decision yet made about message type

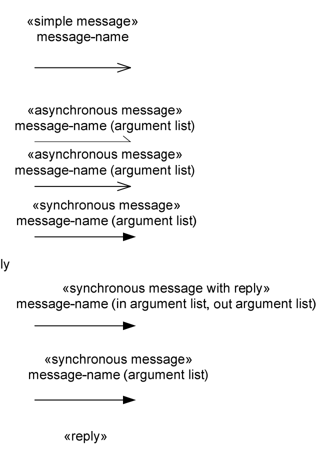
a) Asynchronous (loosely coupled) message communication
UML 1.3
UML 1.4 and in UML 2.0

b) Synchronous (tightly coupled) message communication

c) Synchronous (tightly coupled) message communication with reply

c1) Option 1:

c2) Option 2:

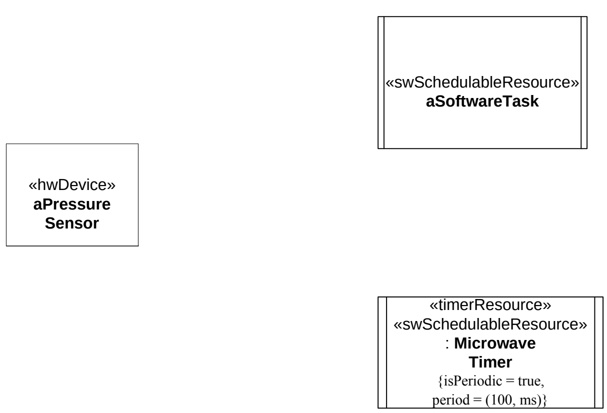


MARTE

- Modeling and Analysis of Real-Time Embedded systems
- UML profile
  - "coherent set of extensions applicable to a given domain or purpose" (Rumbaugh et al. 2005)
- Some MARTE stereotypes
  - Concurrent task: «swSchedulableResource»
  - Hardware device: «hwDevice»
  - Timer (hardware or software): «timerResource»
- MARTE stereotypes can have attributes
  - E.g., a periodic task with time intervals of 100 msec
    - {isperiodic = true, period = (100, ms)}

Examples of MARTE notation

Figure 2.19: Examples of MARTE notation for real-time embedded systems



Task Structuring Criteria

- Each task is structured using three orthogonal criteria
  - Represented using stereotypes
    - MARTE stereotype for concurrent task
      - «swSchedulableResource»
    - Object role criterion
      - Carried over from analysis model
      - E.g., «input »
    - Concurrency criterion
      - From task structuring
      - E.g., «event driven»

Concurrency Structuring Criteria

- Define how task is activated
  - Event driven task
    - Stereotypes:
      - «event driven» «swSchedulableResource»
    - Activated by external event (e.g., interrupt)
  - Periodic task
    - Stereotypes:
      - «timerResource» «swSchedulableResource»
    - Activated by timer
  - Demand driven task
    - Stereotypes:
      - «demand» «swSchedulableResource»
    - Activated by arrival of internal message

Task Structuring - Task Structuring Categories

- I/O task structuring criteria
  - How device interface objects are mapped to I/O tasks
- Internal task structuring criteria
  - How internal objects are mapped to internal tasks
- Task priority criteria
  - Importance of executing task relative to others
- Task clustering criteria
  - Whether and how objects should be combined into concurrent tasks

I/O Task Structuring Criteria

- Event driven I/O task
  - Task for each event (interrupt) driven I/O device
  - Event driven device generates interrupt
- Periodic I/O task
  - Task for each polled I/O device
  - I/O device (usually input) sampled at regular intervals
- Demand driven I/O task
  - Task for each passive I/O device (usually output)
  - Computation overlapped with output

Characteristics of I/O Devices

- Event-driven (interrupt-driven) I/O device
  - Input device
    - Generates interrupt when it has produced input
    - «interruptResource» «input» «hwDevice»
  - Output device
    - Generates interrupt when it has finished output
    - «interruptResource» «output» «hwDevice»
- Passive I/O device
- Smart device

Event Driven I/O Task

One task for each event driven I/O device

Activated by device I/O interrupt

Reads input

Converts to internal format

Disposes of input

- Sends message containing data
- Signals event (message with no data)
- Writes to data store

Figure 13.1 Example of an event driven input task

Figure 13.1a Analysis model – communication diagram

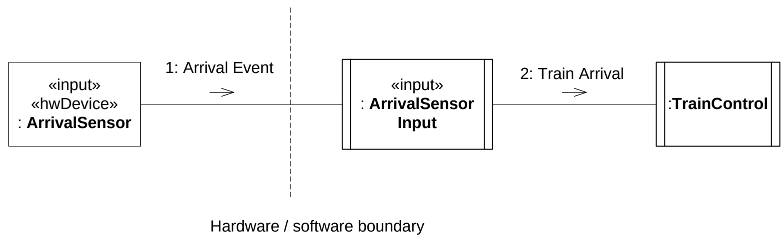
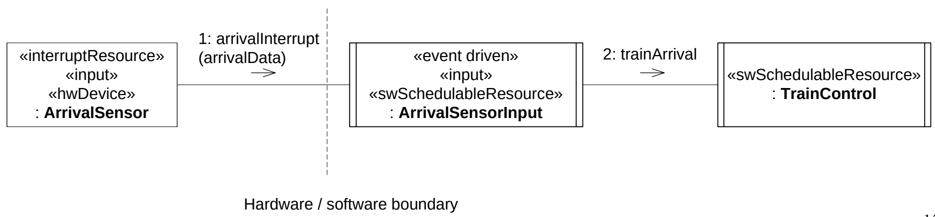


Figure 13.1b Design model – concurrent communication diagram



Event Driven Input Task - Pseudocode

Initialize external device;
loop
-- Wait for external interrupt from input device
wait (externalEvent);
Read input from device;
if input is recognized
then
  Convert input to internal format;
  -- send message to consumer task or write to passive entity object;
  send (message, consumer);
else – input was not recognized;
    -- handle error, e.g., output or send error message;
end if;
end loop;

Characteristics of I/O Devices

–Passive I/O device
- – Device does not generate interrupt
- – Input from passive device
  - – « passive » «input» «hwDevice»
  - – Polled on periodic basis
- – Output to passive device
  - – « passive » «output» «hwDevice»
  - – On demand

Periodic I/O Task

Task for each polled I/O device

    Activation of task is periodic

    Samples I/O device

Periodic I/O task

    Activated by timer event

    Performs I/O operation

    Waits for next timer event

Figure 13.2 Example of a periodic input task

Figure 13.2a Analysis model – communication diagram

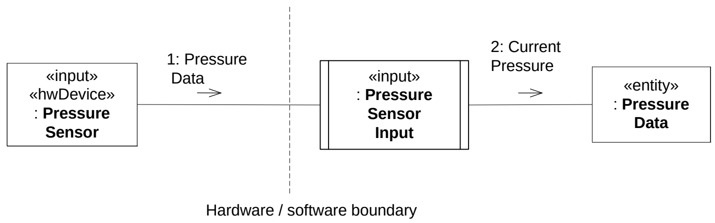
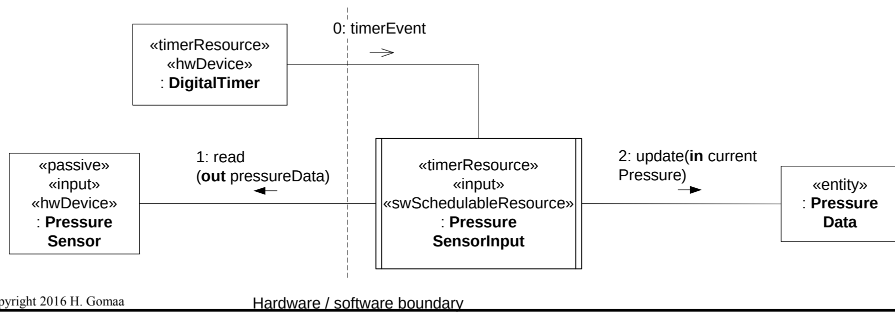


Figure 13.2b Design model – concurrent communication diagram



Periodic Input task - Pseudocode

Initialize external device;

loop

-- Wait for timer event from external clock

-- External clock could be RT clock or operating system timer;

wait (timerEvent);

Read current value of input from device;

if input is recognized

then

Convert input to internal format;

--optionally if device is digital device.

if value of input NOT EQUAL previous value

-- send message to consumer task or write to passive entity object;

then send (message, consumer);

else – input was not recognized;

-- handle error, e.g., output or send error message;

end if;

end loop;

Demand Driven I/O Task

- Task for each passive I/O device (usually output)
  - Passive I/O device does not need to be polled
  - Computation overlapped with output
    - Task output to device overlapped with
    - Computational task that produces data
- Usually for passive output device
  - Demand driven I/O task
- Passive input devices more likely to be polled
  - Periodic input task

Figure 13.3 Example of a demand output task

Figure 13.3a Analysis model – communication diagram

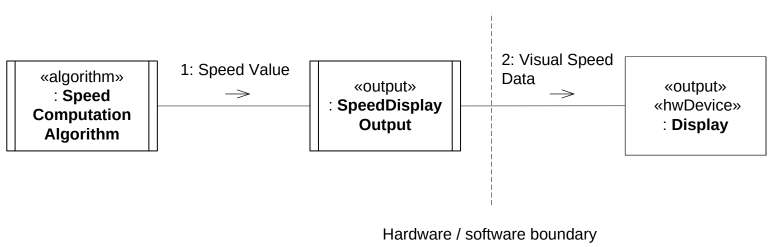
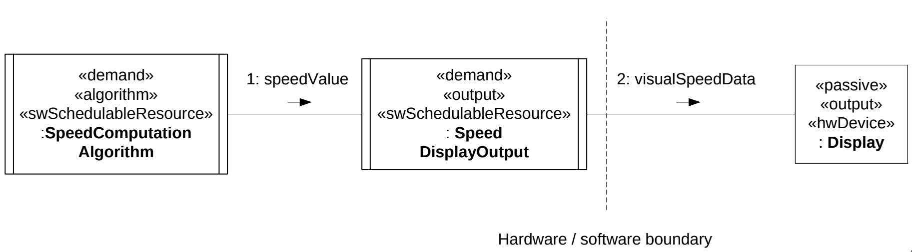


Figure 13.3b Design model – concurrent communication diagram



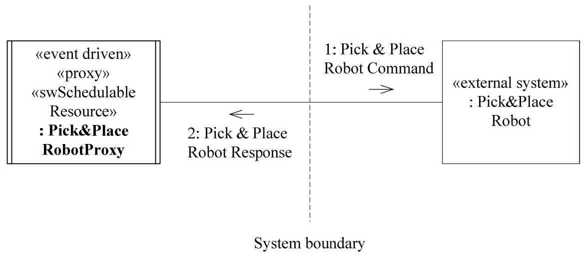
Demand Driven Output Task - Pseudocode

```
begin
Initialize output device, if needed;
loop
-- Wait for message from producer task arriving via connector;
aConnector.receive (message);
Extract message name and any message parameters from message;
-- Process message;
Convert data to output format if needed,
Output data to output device;
if output device error;
   Handle error case;
end if;
end loop;
end;
```


Event Driven Proxy Task

- Interfaces to computer-based system
  - External system
  - Smart device
    - Microprocessor driven I/O device
    - Connected to embedded system using communication link
    - Communicates with embedded system using messages
      - Uses communication protocol, e.g., TCP/IP

Figure 13.5 Example of event driven proxy task



Internal Task Structuring Criteria

- Periodic task
  - Task for each periodic activity
- Demand task
  - Task for each demand driven internal activity
- Control task
  - Task executes state machine
- User interaction task
  - Task for each sequential user activity

Periodic Task

Task for each periodic activity

Task activated periodically

- Activated by timer event
- Performs activity
- Waits for next timer event

Figure 13.6 Example of periodic task

Figure 13.6a Analysis model – communication diagram

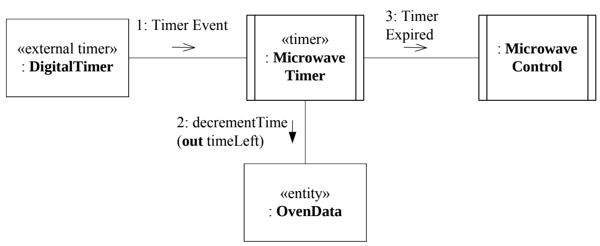
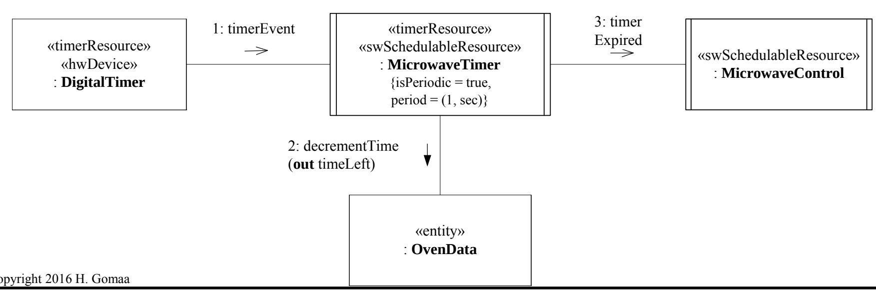


Figure 13.6b Design model – concurrent communication diagram



Periodic Algorithm Task - Pseudocode

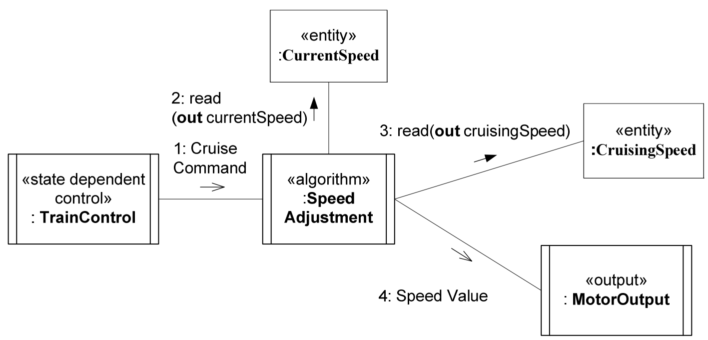
begin
loop
-- Wait for timer event;
wait (timerEvent);
execute periodic algorithm;
prepare output message containing
message name and parameters
-- send output message;
aConnector.send (message);
end if;
end loop;
end;

Demand Driven Task

- Activity executed on demand
  - Activated by internal event or message
  - Map to Demand Task
- Demand task
  - Activated on demand by event or message sent by different task
  - Performs demanded action
  - Waits for next event or message

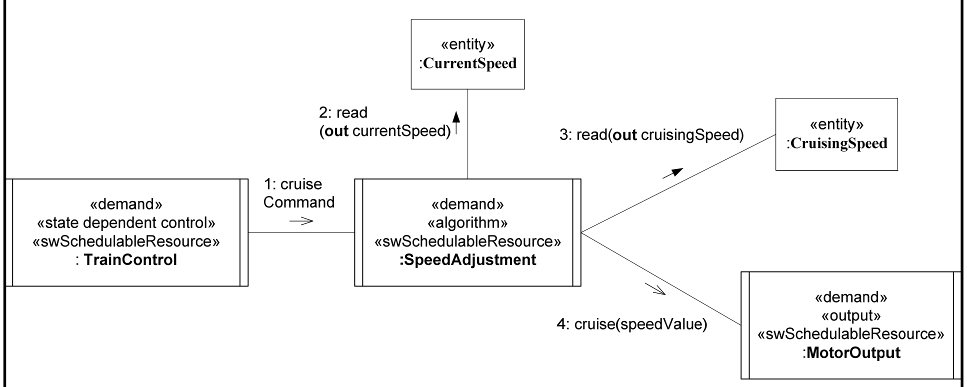
Example of Demand Driven Task -Analysis Model

Figure 13.7a Analysis model – communication diagram



Example of demand driven task

Figure 13.7b Design model – concurrent communication diagram



Demand Driven Task - Pseudocode

```
begin
loop
-- Wait for message or event from producer task arriving via message connector;
aConnector.receive (message);
Extract message name and any message parameters from message;
Perform requested action on demand
    -   Read data from passive entity object(s) if needed
    -   Execute action
    -   Update data in passive entity object(s) if needed
Prepare output message or response containing message name and parameters
-- send output message or event;
aConnector.send (message);
end loop;
end;
```


State Dependent Control Task

- Task executes statechart
  - State dependent control object executes statechart
  - Execution of statechart is sequential
- One task for each control object
- Can have multiple tasks of same type

Figure 13.8 Example of state dependent control task

Figure 13.8a Analysis model – communication diagram

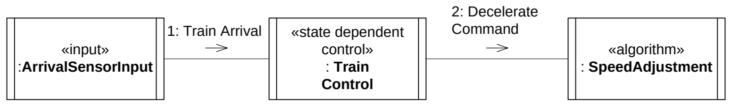
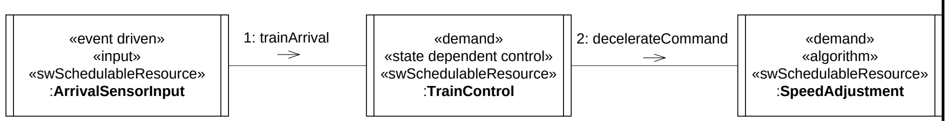


Figure 13.8b Design model – concurrent communication diagram



State Dependent Control Task - Pseudocode

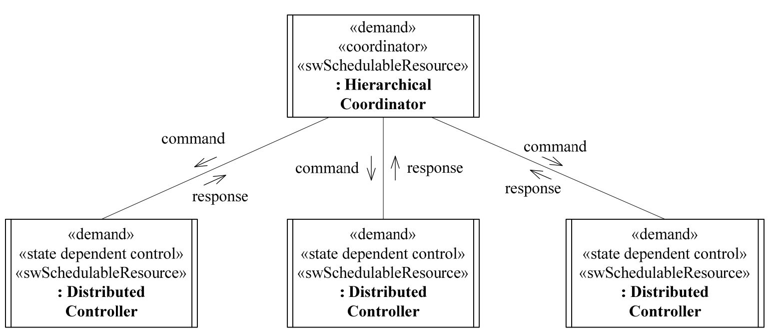
```
loop
 -- Messages from all senders are received on Message Queue
 Receive (messageQ, message);
 -- Extract the event name and any message parameters
 newEvent = message.event
 -- Assume state machine is encapsulated in object aSTM;
 -- Given the incoming event, lookup state transition table;
 -- change state if required; return action to be performed;
  aSTM.processEvent (in newEvent, out action);
  -- Execute state dependent action(s) as given on state machine;
  case state_dependent_action of
   action_1:
     execute state_dependent_action 1;
     exit;
   action_2
     execute state_dependent_action 1;
     exit;
   ...
   action_n
     execute state_dependent_action n;
     exit;
  end case;
end loop;
```

Coordinator Task

- Decision making task
  - Not state dependent
  - Action depends on input received
  - Decides when, and in what order, other actions execute
  - Encapsulates decision making logic
  - To execute action
    - Sends message to another task to execute action
    - Calls operation of passive object to execute action

Example of Coordinator Task

Figure 13.10 Example of a demand driven coordinator task



Coordinator Task - Pseudocode

```
loop
-- Messages from all senders are received on Message Queue
Receive (messageQ, message);
Extract message name and message parameters from message;
-- For coordinator task, action is not state dependent;
  case message of
   action_1:
     execute action 1;
     exit;
   action_2
     execute action 2;
     exit;
   ...
   action_n
     execute action n;
     exit;
  end case;
end loop;
```


User Interaction Task

- One task for each sequential user activity
- Multi-user system
  - One task per user
  - User may also spawn background tasks
- Windowing system
  - User engaged in multiple activities
  - Each window executes sequential activity
  - One task for each window

Figure 18.12 Example of user interaction task

Figure 13.11 Example of event driven user interaction task and demand driven service task

Figure 13.11a Analysis model – communication diagram

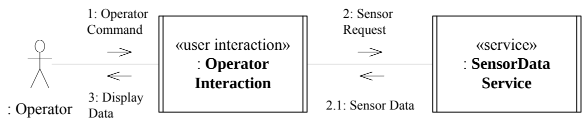


Figure 13.11b Design model – concurrent communication diagram

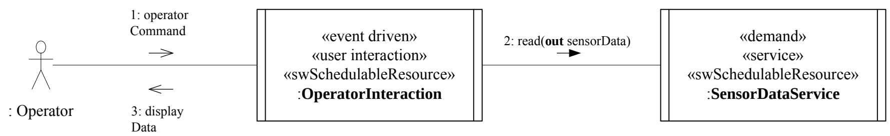
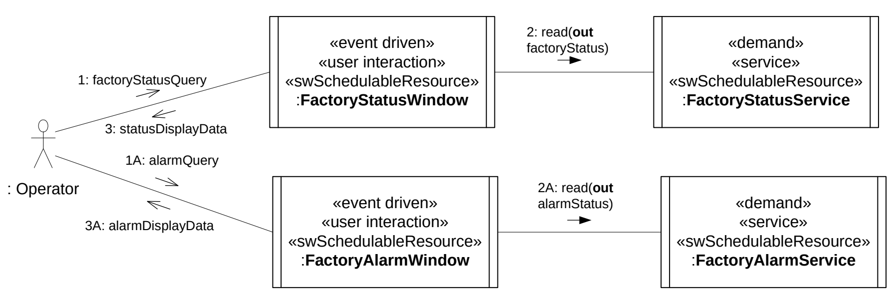


Figure 18.12 Example of user interaction task

Figure 13.11 Example of event driven user interaction tasks and demand driven service task

Figure 13.11c Design model – concurrent communication diagram



User Interaction Task - Pseudocode

```
begin
loop
Output menu or prompt to user
Wait (user response)
Read user input
Process user input and have further interactions with user if necessary.
-- Send message with user request to consumer task
aConnector.send (user request);
-- Wait for response from consumer task arriving via message connector;
aConnector.receive (consumer response);
Extract and process consumer response;
Prepare textual and/or graphical output for user;
Output response to user
end loop;
end;
```


Task Priority Criteria

- Important consideration
  - Performance Analysis
  - Real-Time Scheduling
    - Rate Monotonic Analysis
- Time critical
  - Activity that has hard deadline
  - Map to time critical task
  - E.g., high priority event driven input task
    - Arrival Sensor Input task (Fig. 13.1)
- Non-time-critical computationally intensive
  - Low priority activity
  - E.g., low priority computationally intensive task
    - Speed Computation Algorithm task (Fig. 13.3)

Task Clustering Criteria

- Temporal clustering
  – Activities activated by same event
- Sequential clustering
  – Activities must be executed sequentially
- Control clustering
  – Control object grouped with objects it activates
- Task Inversion
  – Map all objects of same type to one task

Temporal Clustering

- Candidate tasks
  – Tasks determined during first stage of Task Structuring
- Two or more candidate tasks
  – Each candidate task executes an activity
  – Candidate tasks activated by same event, e.g. timer
  – No sequential dependency between tasks
- Candidate tasks grouped into one task
- Best form of temporal clustering
  – Candidate tasks are functionally related
  – Have same sampling rates
- Weaker form of temporal clustering
  – Sampling rates are multiples of one another

Figure 13.12a Example of temporal clustering
- periodic I/O tasks before temporal clustering

Design model – concurrent communication diagram

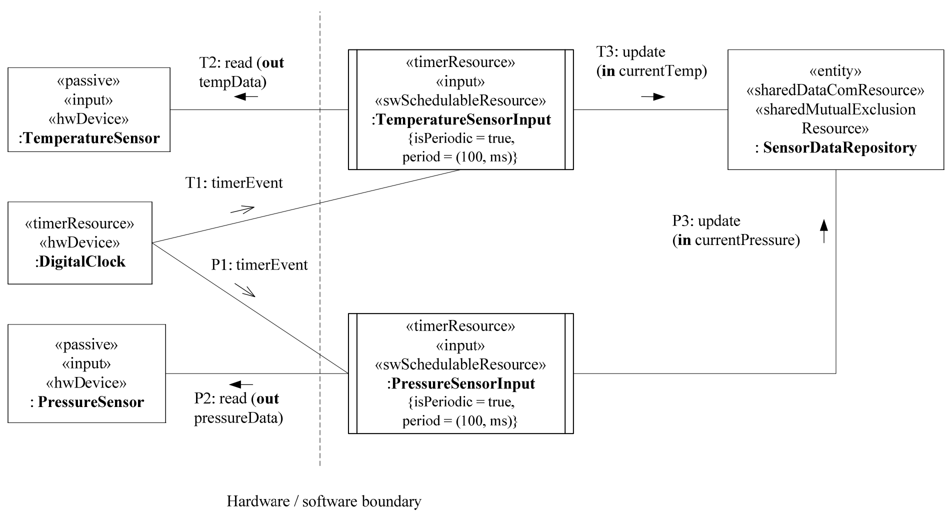
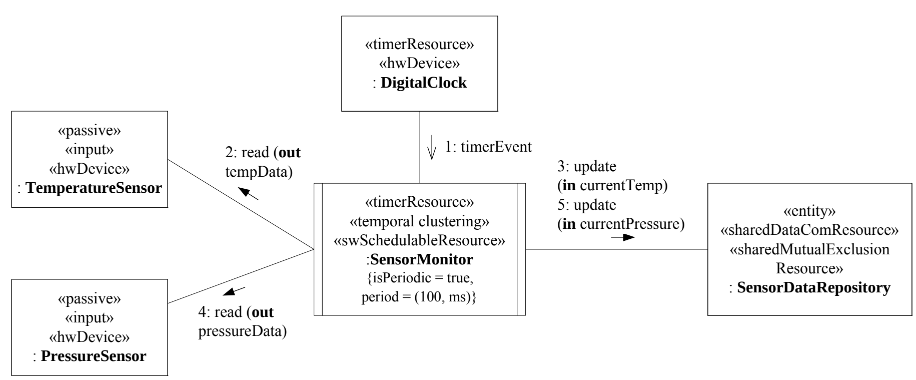


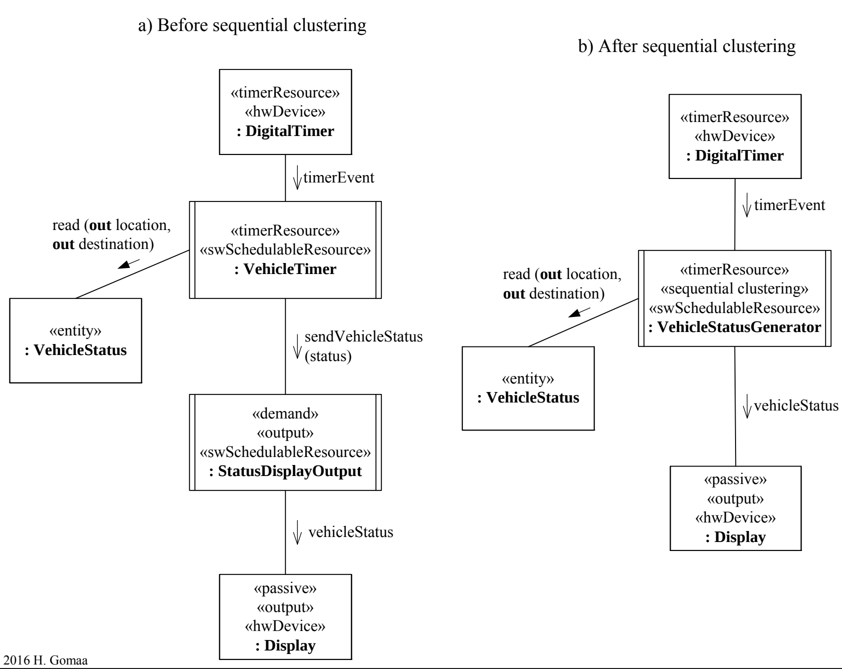
Figure 13.12b Example of temporal clustering
- After temporal clustering



Sequential Clustering

- Candidate tasks
  - Tasks determined during first stage of Task Structuring
- Two or more candidate tasks
  - Each candidate task executes an activity
  - Candidate tasks must execute in predefined sequence
  - First candidate task in sequence activated by event
  - Subsequent candidate tasks should execute without delay
- Candidate tasks grouped into one task

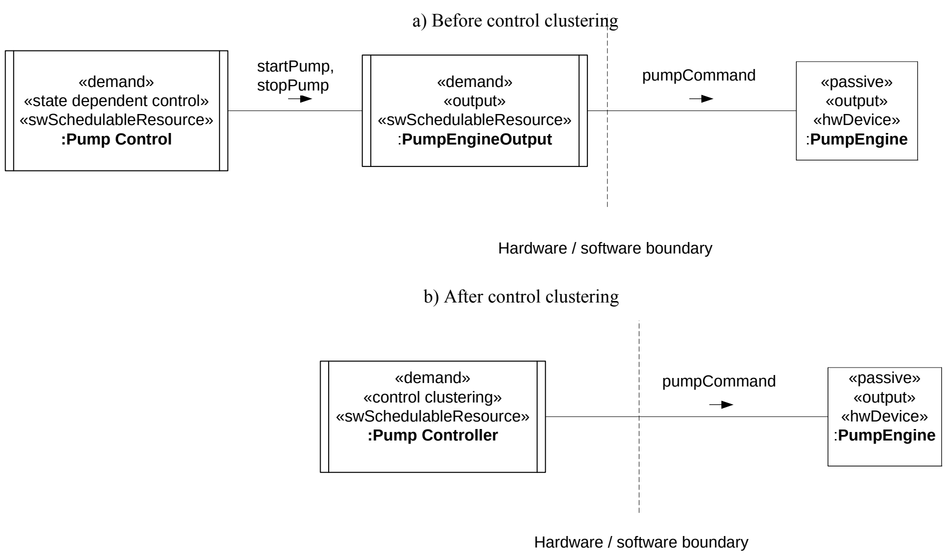
Figure 13.13 Example of Sequential Clustering



Control Clustering

- State dependent control object grouped with actions or activities
- State dependent
  - Action triggered by control object
    - Executes at state transition - Combine
  - Activity enabled/disabled by control object
    - Executes for duration of state - Do not combine
  - Activity triggered by control object
    - Executes for duration of state - Possibly combine
- Non-state dependent actions
  - Send events to control object- Possibly combine

Figure 13.14 Example of Control Clustering

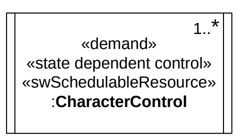


Multiple Instance Task Inversion

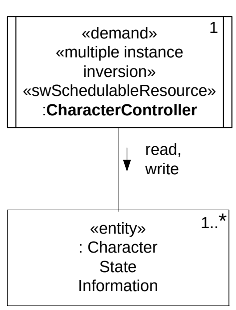
- Multiple tasks of same type
  - Used to model multiple objects of same type
- Potential Problem
  - High overhead of modeling each object using separate task
- Task inversion
  - Model all objects of same type using one task
  - Use separate data record for each object to store state information

Figure 13.15 Example of multiple instance task inversion

a) Design model – one task for each character



b) Design model – task inversion
one task for all characters



Design Task Interfaces

- Based on Analysis Model simple message interfaces
- Need to determine type of message communication
  - Also referred to as message communication patterns
- Asynchronous message communication
- Synchronous message communication
  - – With reply
  - – Without reply
- Event synchronization
  - External event (interrupt)
  - Timer event
  - Internal event
- Passive objects
  - – Task interfaces to information hiding object

Figure 13.16 Examples of asynchronous message communication

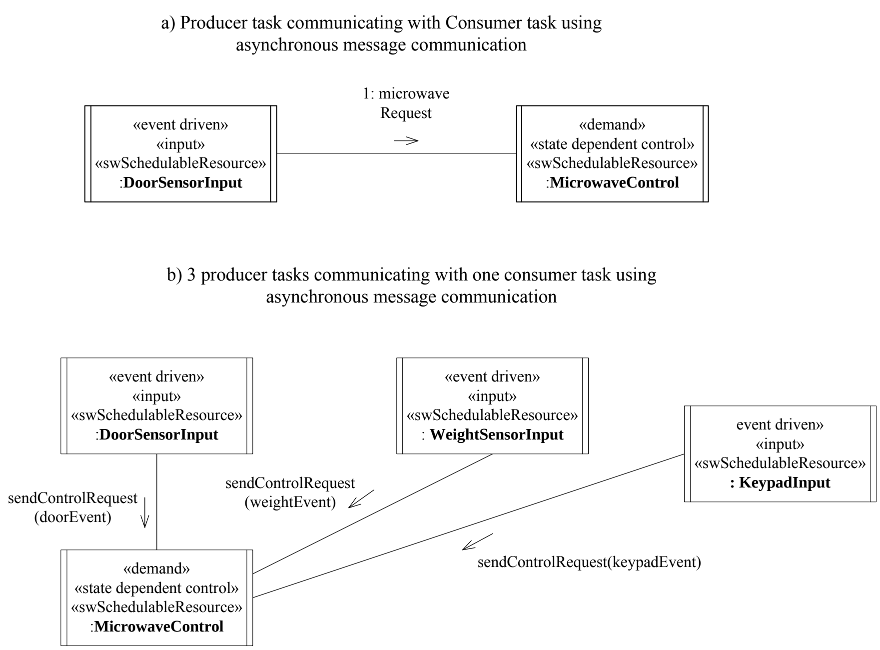


Figure 13.17 Example of synchronous message communication with reply

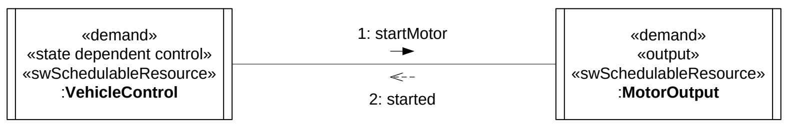
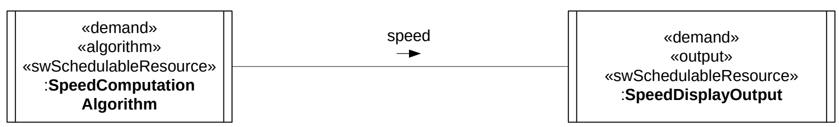


Figure 13.18 Example of synchronous message communication without reply



Event Synchronization

- Types of events
  - External event (interrupt)
  - Timer event
  - Internal event
- Two tasks may need to synchronize their operations
  - If message contains no data, can use internal event
- Source task signals event
  - signal (event)
- Destination task waits for event
  - wait (event)
  - Suspended until event signaled

Figure 13.19 Example of external event

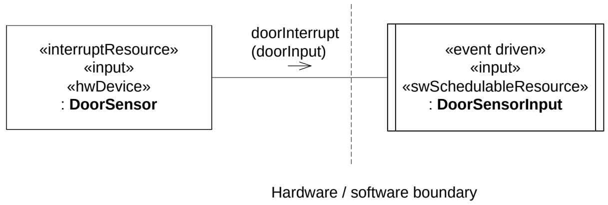


Figure 13.20 Example of timer event

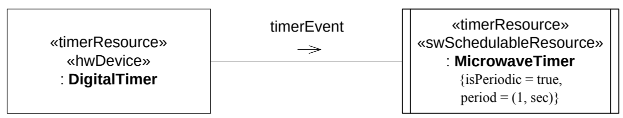
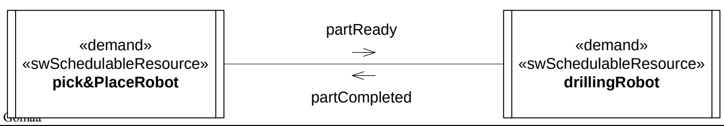


Figure 13.21 Example of internal event synchronization between two tasks



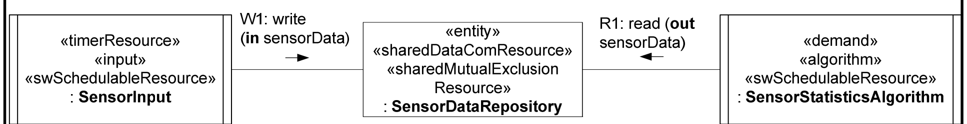
MARTE Notation

- Modeling and Analysis of Real-Time Embedded systems
- MARTE stereotypes
  - Concurrent task:
    - «swSchedulableResource»
  - Passive entity object
    - Shared resource
      - «sharedDataComResource»
    - Mutually exclusive resource
      - «sharedMutualExclusionResource»

Information Hiding Object

- Passive entity object
  - Encapsulates data
  - Hides contents of data structure
  - Data accessed indirectly via operations
- Passive object accessed by two or more tasks
  - Operations must synchronize access to data
  - Mutually exclusive access to data

Figure 13.22 Example of tasks invoking operations of passive object

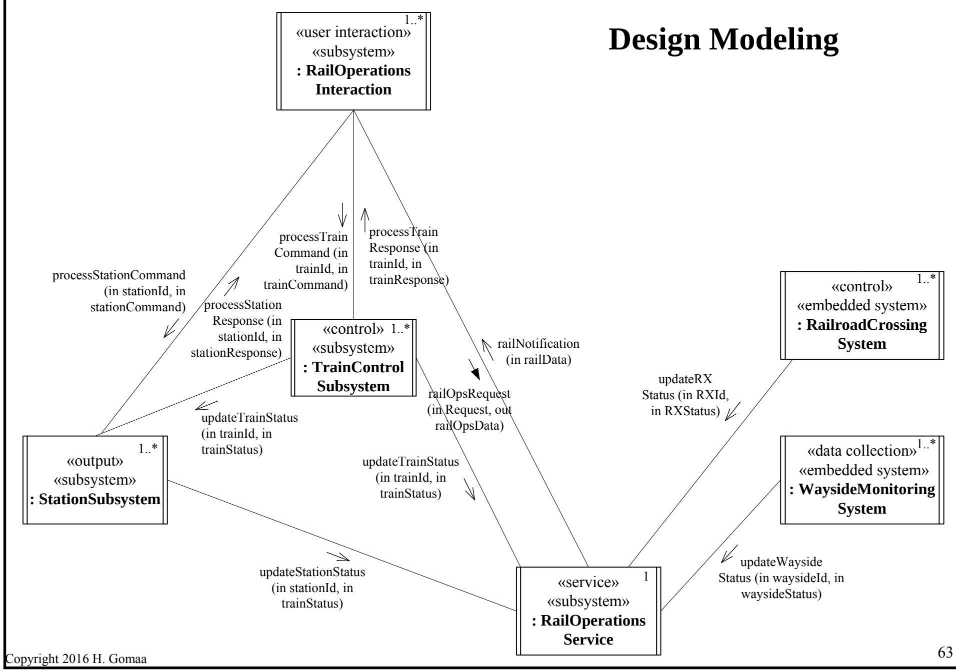


Case Study:
Light Rail Control System

- 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



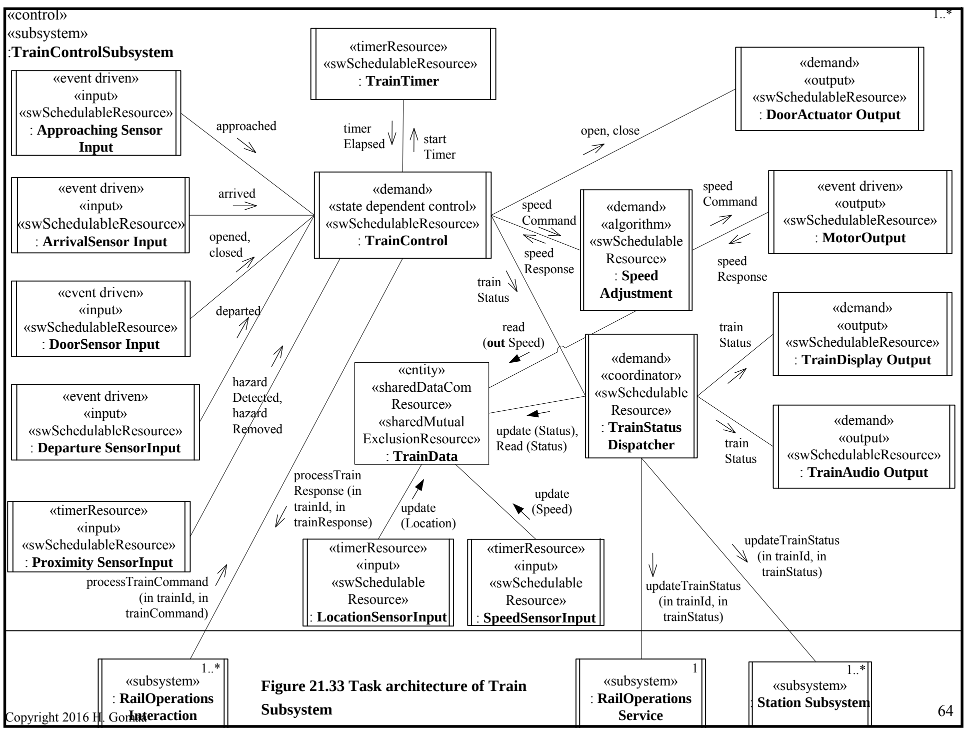


Figure 21.33 Task architecture of Train Subsystem

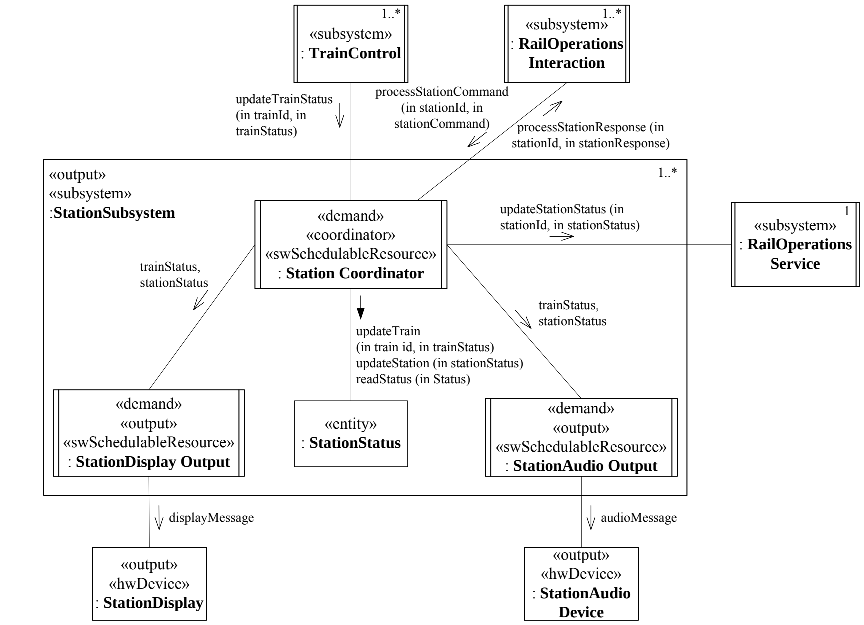


Figure 21.34 Task architecture of Station Subsystem

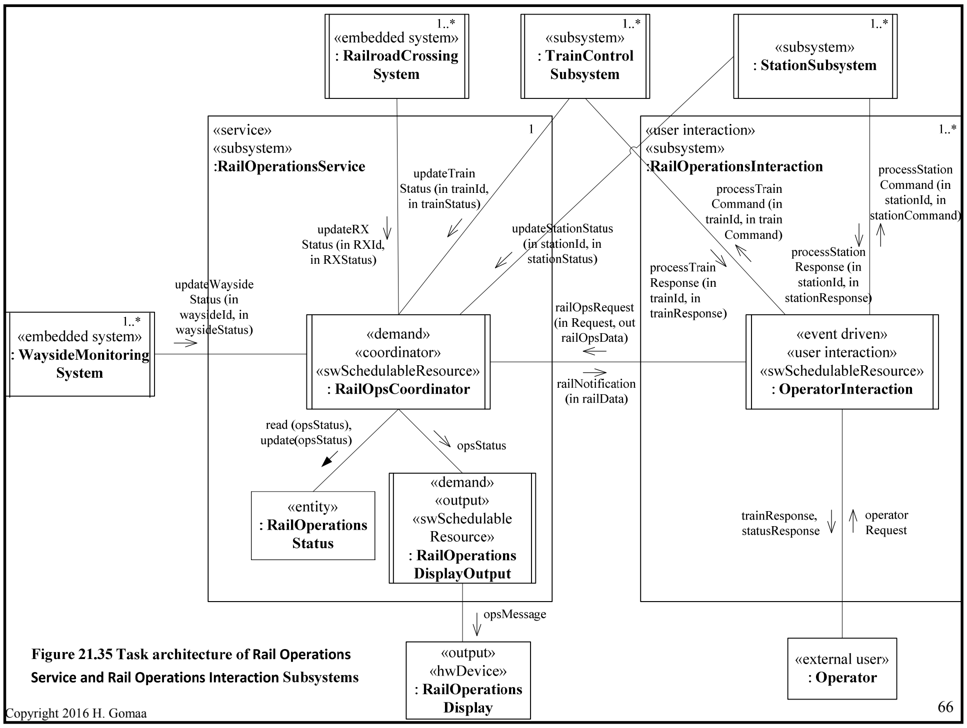


Figure 21.35 Task architecture of Rail Operations Service and Rail Operations Interaction Subsystems

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