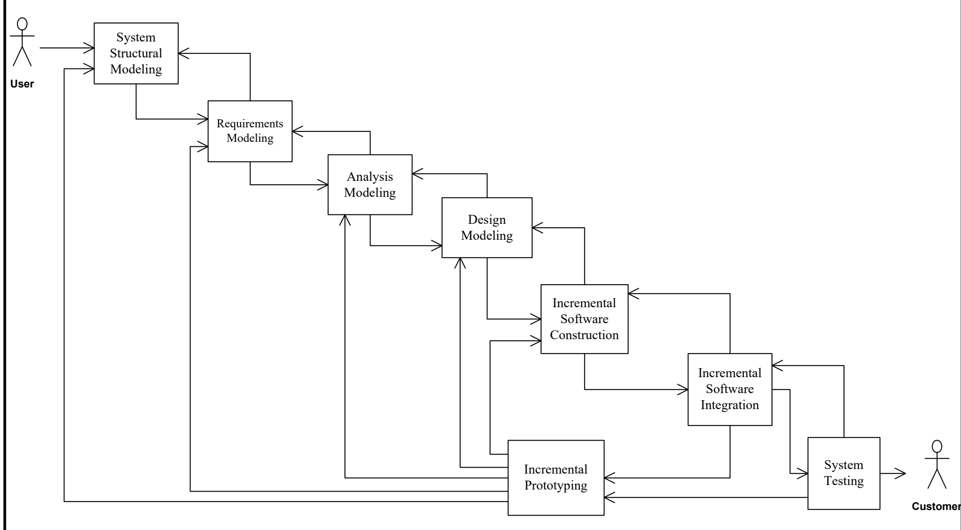
SWE 760

Lecture 12 – Performance Analysis of Real-Time Designs

Reference:

H. Gomaa, Chapters 17, 18 - *Real-Time Software Design for Embedded Systems*, Cambridge University Press, 2016

Figure 4.1 COMET/RTE life cycle model



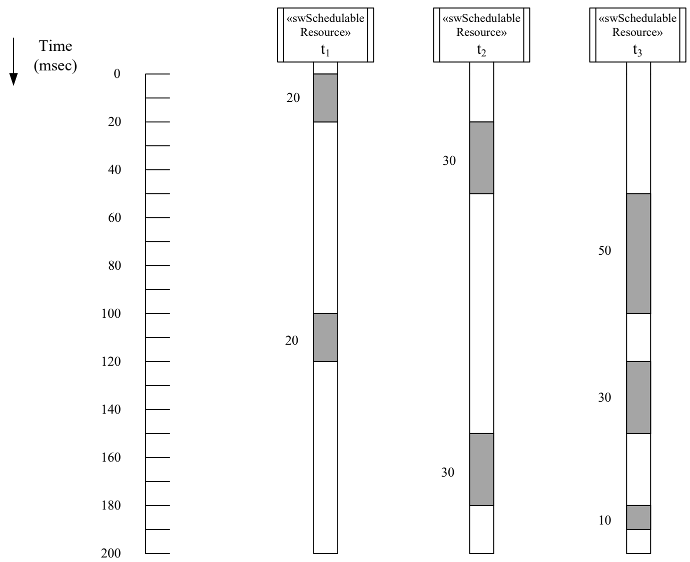
Software Modeling for RT Embedded Systems

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**

Performance Analysis of Real-Time Designs

- Quantitative analysis of RT designs
  - Allow early detection of potential performance problems
- Investigate alternative
  - Software designs
  - Hardware configurations
- Approaches
  - Real-time scheduling theory
    - Rate Monotonic Analysis (RMA) [SEI]
  - Combine RMA with Event sequence analysis
    - Sequence of components to process external event
- UML timing diagram
  - Sequence diagram annotated with time
  - Depicts time-ordered execution sequence of tasks

Figure 2.20: UML Timing Diagram (time annotated sequence diagram)



Rate Monotonic Analysis (RMA)

- Based on Real-Time scheduling theory
  - Priority pre-emption scheduling
- RMA theory
  - Initially developed for independent periodic tasks
  - Made practical by Software Engineering Institute
- Extended to address
  - Aperiodic tasks
  - Scheduling with task synchronization
  - Address priorities imposed by application
- RMA Integrated with COMET/RTE
  - Design method for real-time systems

Rate Monotonic Analysis

- Given for each task
  - CPU time Ci
  - Period Ti
  - Task Utilization Ui = Ci / Ti
- Rate Monotonic Algorithm
  - Assigns tasks fixed priorities based on their periods
    - Task with shorter period
    - Given higher priority
  - Computes whether group of n tasks will meet their deadlines
    - Assume Task deadline (elapsed time to complete execution) = Period Ti

Utilization Bound Theorem

- "A set of n independent periodic tasks scheduled by the rate monotonic algorithm will always meet its deadlines, for all task phasings, if
  C1 / T1 + C2 / T2 .... Cn / Tn =< U(n) "
- Total task Utilization = U(n)
- Upper bound U(n) converges to 0.69 as n approaches infinity

Table 17.1: Utilization Bound Theorem

Number of Tasks n	Utilization Bound U(n)
1	1.000
2	0.828
3	0.779
4	0.756
5	0.743
6	0.734
7	0.728
8	0.724
9	0.720
Infinity	0.690

Completion Time Theorem

"For a set of independent periodic tasks,

If each task meets its deadline when all tasks are started at the same time,

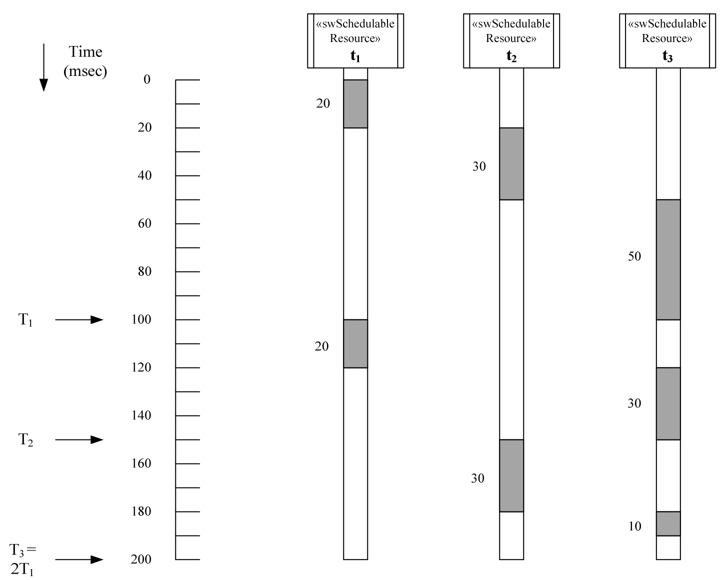
Then the deadlines will be met for any combination of start times."

- Consider execution of given Task ti during its first period
- Consider execution of all higher priority tasks
  - Have shorter periods and can pre-empt Task ti
- Illustrate using timing diagram
  - Time annotated sequence diagram

Example of Completion Time Theorem

- Consider 3 tasks
  - Task t1
    - CPU time C1 = 20 msec
    - Period T1 = 100 msec
    - Utilization U1 = 0.2
  - Task t2
    - CPU time C2 = 30 msec
    - Period T2 = 150 msec
    - Utilization U2 = 0.2
  - Task t3
    - CPU time C1 = 90 msec
    - Period T1 = 200 msec
    - Utilization U3 = 0.45
- Total utilization = 0.85 > Upper Bound for Utilization Bound Theorem (0.779 for three tasks)

Figure 17.1 Timing diagram for tasks executing on a single processor system



More Advanced Rate Monotonic Analysis

- Aperiodic task
  - Treat as equivalent periodic task
  - Use worst case scenario
  - "Period" of aperiodic task
    = minimum event inter-arrival time Ta
- Scheduling with task synchronization
  - 2 or more tasks accessing passive entity objects
  - Task in critical section can block higher priority task
    - Priority inversion
  - Priority ceiling protocol

Scheduling with Task Synchronization

- Priority inversion
  - Low priority task
    - Acquires critical resource
    - Blocks higher priority tasks
- Priority Ceiling Protocol
  - Ensures high priority task blocked by at most one lower priority task
  - Increase priority of low priority task
    - To that of highest priority task blocked by it
  - Worst case blocking time Bi = CPU time of one lower priority task

Application Priorities

- Rate monotonic priorities
  - Tasks allocated priorities based on length of period
    - Task with shorter period assigned higher priority
  - Task can only be pre-empted by higher priority task with shorter period
- Non-rate monotonic priorities
  - Referred to as "Rate monotonic priority inversion"
  - Tasks allocated priorities based on application need
    - Could be different from rate monotonic priorities
  - E.g., Give task higher priority to address interrupt handling
- Extensions to RMA to address rate monotonic priority inversion
  - Task can be pre-empted by higher priority tasks with longer periods

Generalized Real-Time Scheduling Theory

- For a given Task ti with CPU time Ci & Period = Ti, need to consider:
- Execution time of Task ti
  - Utilization = Ci / Ti
- Pre-emption by higher priority tasks with periods Tj < Ti
  - Rate monotonic priorities
  - Utilization of Task tj = Cj / Tj
- Pre-emption by higher priority tasks with periods > Ti
  - Non-rate monotonic priorities
  - Utilization of task tk = Ck / Ti
- Blocking by lower priority tasks
  - Priority Inversion
  - Worst case blocking utilization = Bi / Ti

Generalized Real-Time Scheduling Theory

- Generalized Theory needs to be applied to each task
- For each task, need to consider
  - Execution utilization by task ti
  - Pre-emption utilization by every higher priority task with period < Ti
  - Pre-emption utilization by every higher priority task with period > Ti
  - Worst case Blocking utilization

Generalized Real-Time Scheduling Theory

- Generalized Utilization Bound Theorem
  - Estimate Total utilization during Period Ti
    = Execution utilization
    + Pre-emption utilization
    + Worst case Blocking utilization
- Generalized Completion Time Theorem
  - Estimated task elapsed time
    = CPU time
    + pre-emption time
    + blocking time

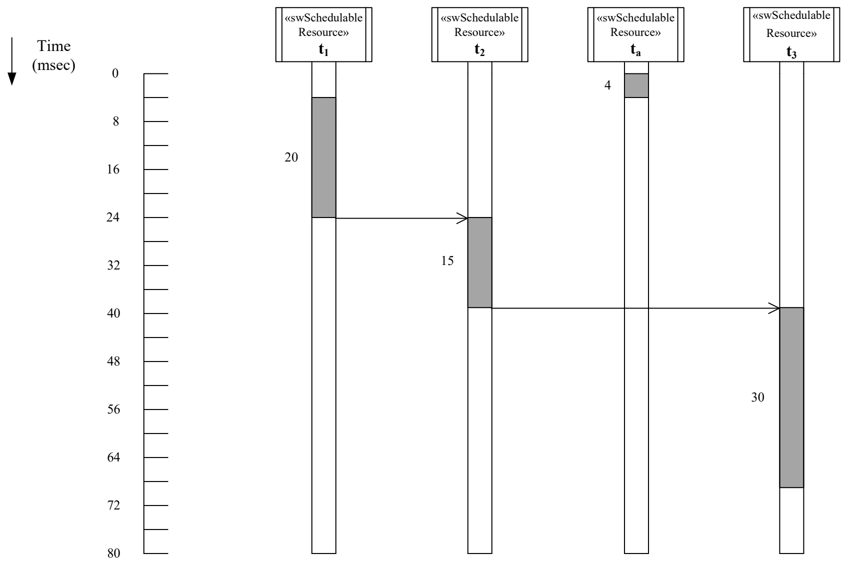
RMA in Design

- Pessimistic designer
  - Consider worst cases
  - At design time
    - CPU times are estimates
  - If in doubt, better to over-estimate CPU time
- Use pessimistic Utilization Bound Theorem
  - Worst case utilization = 0.69
- Analyze Design Alternatives
  - If design fails to meet performance requirements
    - Investigate different hardware configurations
    - Investigate different software designs
      - Restructure task design
      - COMET task clustering criteria

Event Sequence Analysis

- RMA assumes tasks are independent
- Real RT systems
  - Tasks interact with each other
  - Event sequence scenarios
- Performance requirements of real-time system
  - Response times to external events
- Given external event
  - Determine sequence of tasks activated
- Event sequence analysis
  - Use event sequence timing diagrams
  - Consider sequence of tasks required to
    - Process external event
    - Generate system output

Figure 17.3 Timing diagram for tasks executing in an event sequence on a single processor system



Performance Analysis of Multiprocessor Systems

- Use timing diagrams
- Depict tasks on different CPUs
- Global scheduling
  - A task can execute on any CPU
- Partitioned scheduling
  - Tasks are partitioned
    - i.e., pre-assigned to individual CPUs
  - A given task only executes on a single CPU
- Event sequence analysis
  - Use event sequence timing diagrams
  - Consider sequence of tasks on multiple CPUs

Figure 17.4 Timing diagram for tasks executing on a dual processor system

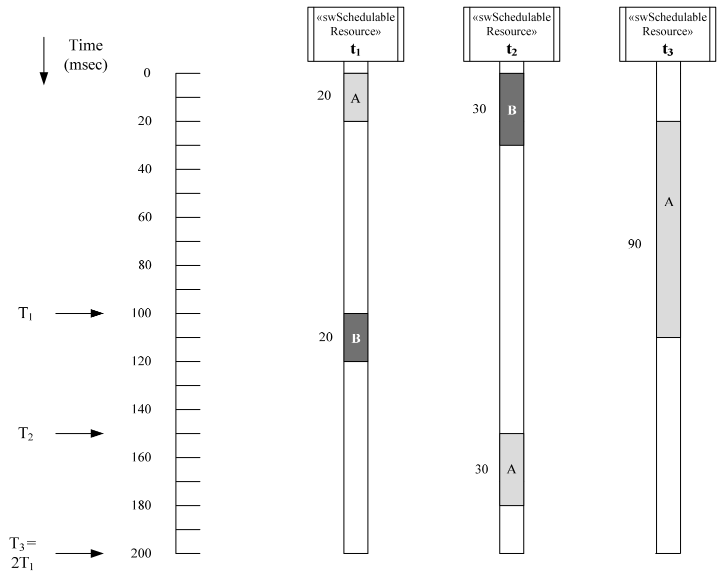
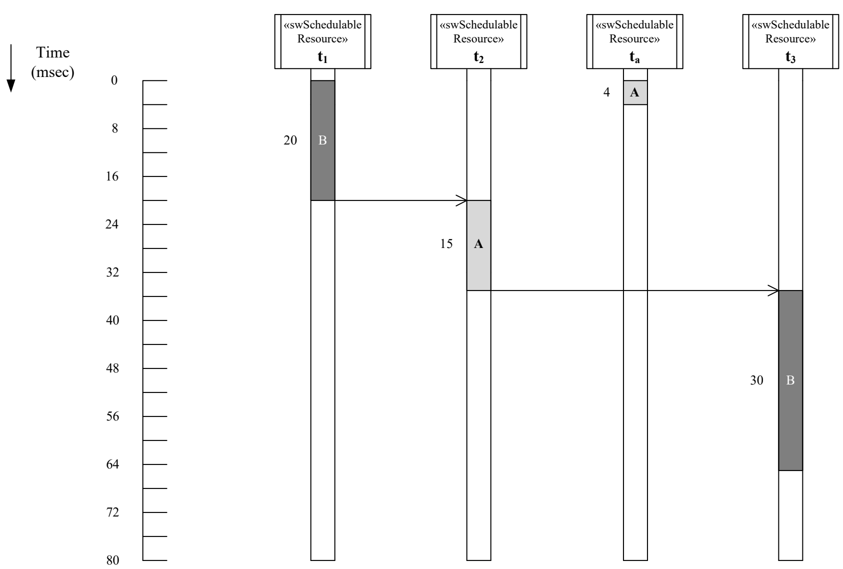


Figure 17.6 Timing diagram for tasks in an event sequence executing on a dual processor system



Integrating RMA with COMET Real-Time Design

- Structure system into concurrent tasks
  - Use COMET task structuring criteria
- Define task interfaces
- Estimate for each Task ti
  - CPU time Ci
  - Period Ti - worst case estimate for aperiodic task
- Estimate / measure system overhead
  - Add to each task's estimated CPU time
- Allocate task priorities
  - Initially use rate monotonic priority
  - Allocate non-rate monotonic priority if dictated by application

Case Study: - 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

Use Case Model

- Use cases for train arrival and departure
  - – Depart from Station
    - A train leaves the station
  - – Arrive at Station
    - A train arrives at the station
  - – Control Train at Station
    - Passengers get on and off the train at the station
- Use cases for detecting presence and removal of hazards
  - – Detect Hazard Presence
    - Proximity sensor detects an obstacle ahead
  - – Detect Hazard Removal
    - Proximity sensor detects that obstacle has been removed

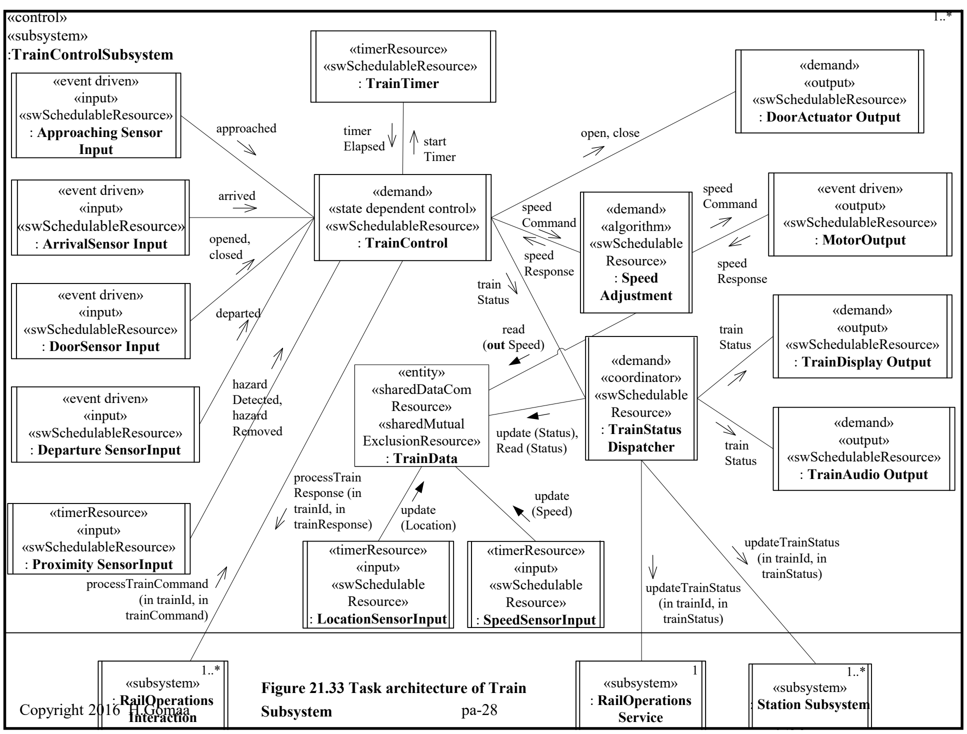


Figure 21.33 Task architecture of Train Subsystem

Light Rail Control System
- Performance Analysis

- Train Control Subsystem
  - CPU times of Periodic tasks
    - Tasks activated by timer
    - Aperiodic tasks with minimum event inter-arrival times
  - Consider task CPU time (Ci) and context switching time (Cx)
    - Total task CPU time = Ci + 2*Cx

Table 18.1: Train Subsystem CPU Times

Task	C_i (msec)	Periodic tasks $(C_i + 2*C_x)$ (msec)	Arrival Sensor event sequence tasks $(C_i + C_x + C_m)$ (msec)	Proximity Sensor event sequence tasks $(C_i + C_x + C_m)$ (msec)
Approaching Sensor Input (C_0)	4	5		
Arrival Sensor Input (C_1)	4	5	5	
Train Control (C_3)	5	6	6	6
Speed Adjustment (C_5)	9	10	10	10
Motor Output (C_7)	4	5	5	5
Message communication overhead (C_m)	0.7			
Context switching overhead (C_x)	0.3			
Proximity Sensor Input (C_8)	4	5		5
Speed Sensor Input (C_9)	2	3		
Location Speed Sensor Input (C10)	5	6		
Train Status Dispatcher (C11)	10	11		
Train Display Output (C12)	14	15		
Train Audio Output (C13)	11	12		
Total CPU time used by tasks in event sequence			26	26

Light Rail Control System - Performance Analysis

- CPU times of tasks in event sequence
  - Consider Ci , Cx, message communication overhead (Cm)
    - Total task CPU time = Ci + Cx + Cm
- Tasks in Arrival event sequence
  - Arrival Sensor Input, Train Control, Speed Adjustment, Motor Output
    - $C_e = C_1 + C_3 + C_5 + C_7 + 3*C_m + 4*C_x$
- Tasks in Proximity event sequence
  - Proximity Sensor Input, Train Control, Speed Adjustment, Motor Output
    - $C_p = C_8 + C_3 + C_5 + C_7 + 3*C_m + 4*C_x$

Light Rail Control System - Rate Monotonic Analysis

- Real-Time Scheduling Parameters
- Steady state Periodic and Aperiodic Task Parameters
  - Total utilization = 0.68

    < Upper Bound for Utilization Bound Theorem (0.69)
- Tasks will meet their deadlines

Table 18.2: Real-Time Scheduling Parameters: Steady state Periodic and Aperiodic Task Parameters

Task	CPU time C_i	Period T_i	Utilization U_i	Priority
Speed Sensor Input	3	10	0.30	1
Location Sensor Input	6	50	0.12	2
Proximity Sensor Input	5	100	0.05	3
Motor Output	5	100	0.05	4
Speed Adjustment	10	100	0.10	5
Train Status Dispatcher	11	600	0.02	6
Train Display Output	15	600	0.03	7
Train Audio Output	12	600	0.02	8
Total Utilization for all tasks			0.68	

Light Rail Control System - Rate Monotonic Analysis

- Consider impact of event sequence tasks
- Consider "equivalent" event sequence task added to periodic/aperiodic tasks
- Arrival event sequence task
  - Use period of Arrival Sensor = 200 msec
  - Total utilization = 0.81
    - > Upper Bound for Utilization Bound Theorem (0.69)
- Proximity event sequence task
  - Use period of Proximity Sensor = 100 msec
  - Total utilization = 0.89
- Arrival + Proximity event sequence tasks
  - Total utilization = 1.02

Table 18.3: Real-Time Scheduling Parameters - with event sequence tasks

Task	CPU time C_i	Period T_i	Arrival event sequence - Utilization U_a	Priority (case 1)	Proximity event sequence - Utilization U_p	Priority (case 2)	Arrival & Proximity event sequences Utilization U_q	Priority (case 3)
Speed Sensor Input	3	10	0.30	1	0.30	1	0.30	1
Location Sensor Input	6	50	0.12	2	0.12	2	0.12	2
Proximity Sensor Input	5	100	0.05	3				
Motor Output	5	100	0.05	4	0.05	4	0.05	4
Speed Adjustment	10	100	0.10	5	0.10	5	0.10	5
Train Status Dispatcher	11	600	0.02	7	0.02	6	0.02	7
Train Display Output	15	600	0.03	8	0.03	7	0.03	8
Train Audio Output	12	600	0.02	9	0.02	8	0.02	9
Arrival Event Sequence Task	26	200	0.13	6			0.13	6
Proximity Event Sequence Task	26	100			0.26	3	0.26	3
Total Utilization for all tasks			0.81		0.89		1.02	

Integrating RMA with COMET Real-Time Design

- Develop event sequence scenario for each external event
  - Consider execution sequence of tasks in scenario
  - Estimate period Ti for tasks in scenario
  - Consider all tasks that could pre-empt tasks in scenario
  - Determine if sequence of tasks in scenario complete before Ti?
- Apply generalized utilization bound theorem and/or generalized completion time theorem to tasks in each event sequence

Figure 18.2 Timing Diagram: Event sequence for a train arriving at a station

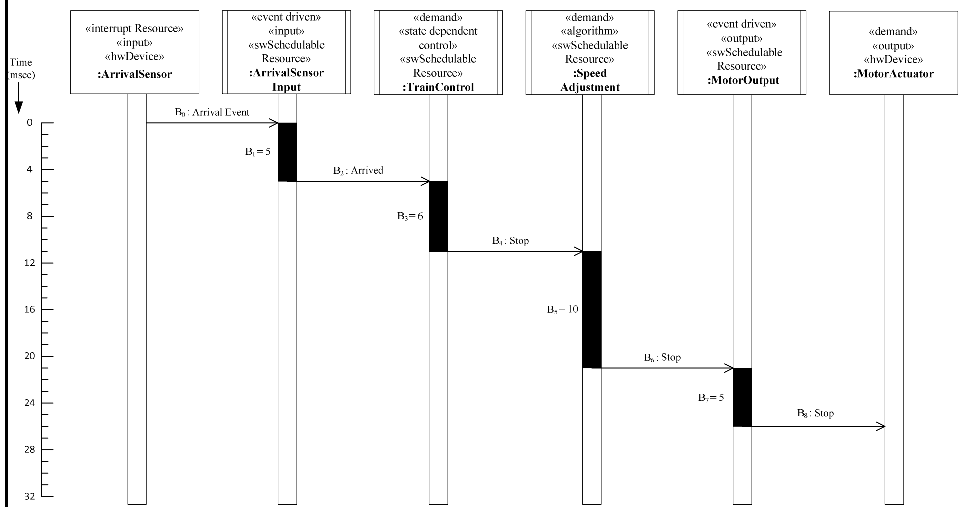
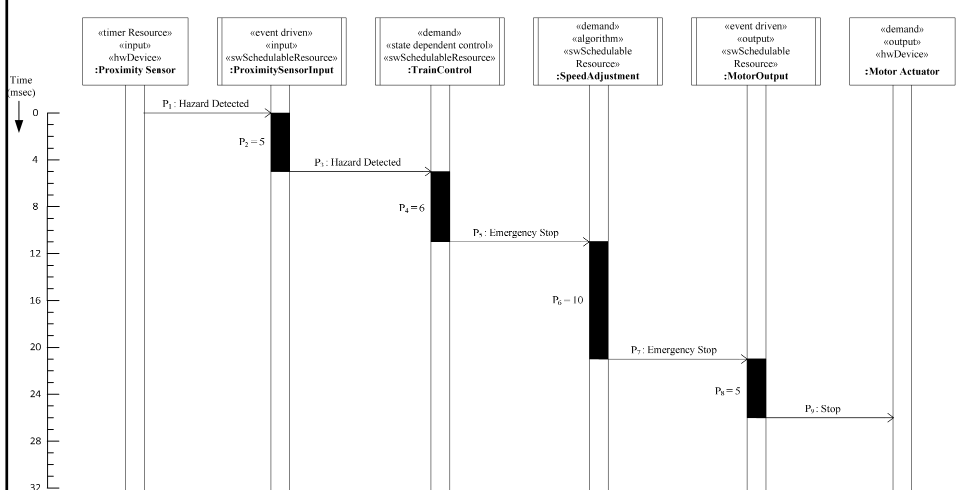


Figure 18.3 Timing Diagram: Event sequence for a hazard detected



Combining Event Sequence Analysis & RMA

- Each event sequence initiated by external event
  - Period Te = minimum inter-arrival time of external event
- Consider CPU times of all tasks in event sequence
  - Estimate CPU time & CPU overhead
- Use RMA to estimate response time to external event
- Consider pre-emption of tasks in event sequence by
  - Higher priority tasks with periods < Te
  - Higher priority tasks with periods > Te
- Consider blocking by lower priority tasks

Detailed Rate Monotonic Analysis

- Will tasks meet their deadlines?
  - Consider each task individually
- Tasks in Arrival event sequence
  - Arrival Sensor Input, Train Control, Speed Adjustment, Motor Output
  - Use period of Arrival Sensor = 200 msec
- Tasks in Proximity event sequence
  - Proximity Sensor Input, Train Control, Speed Adjustment, Motor Output
  - Use period of Proximity Sensor = 100 msec
- Tasks in both Arrival + Proximity event sequences
  - Use period of Proximity Sensor = 100 msec
- Total utilization = 0.77

  > Upper Bound for Utilization Bound Theorem (0.69)

Table 18.4: Real-Time Scheduling: Periodic and aperiodic task parameters

Task	CPU time C_i	Period T_i	Utilization U_i	Rate Monotic Priorities (case 1)	Non-rate Monotic Priorities (case 2)
Speed Sensor Input	3	10	0.30	1	1
Location Sensor Input	6	50	0.12	2	2
Proximity Sensor Input *	5	100	0.05	3	4
Motor Output *	5	100	0.05	4	5
Speed Adjustment *	10	100	0.10	5	6
Train Status Dispatcher	11	600	0.02	8	8
Train Display Output	15	600	0.03	9	9
Train Audio Output	12	600	0.02	10	10
Arrival Sensor Input *	5	200	0.03	7	3
Train Control *	6	100	0.06	6	7
Total Utilization for all tasks			0.77		

(* Tasks in event sequence)

Detailed Rate Monotonic Analysis

- Use non-rate monotonic analysis
- Give four input tasks the highest priority
  - Raise priority of Arrival Sensor Input from 7 to 3
- Apply Generalized Real-Time Scheduling Theory
- For each task, need to consider
  - Execution utilization by task ti
  - Pre-emption utilization by every higher priority task with period < Ti
  - Pre-emption utilization by every higher priority task with period > Ti
  - Worst case Blocking utilization
- Based on this detailed analysis, tasks meet their deadlines

Performance Analysis Using Timing Diagrams

- Depict task execution using timing diagrams
  - Based on task priority
  - Consider tasks in each event sequence
  - Tasks in both Arrival + Proximity event sequences
  - Figure 18.4
    - Scenario with tasks in event sequence scenario
    - Consider other tasks also executing on CPU
- Performance Analysis of Multiprocessor Systems
  - Consider dual-processor system with global scheduling
  - Figure 18.5
    - Scenario with tasks in event sequence scenario
    - Consider other tasks also executing on two CPUs

Figure 18.4 Timing Diagram for tasks in Train Control Subsystem executing on single CPU

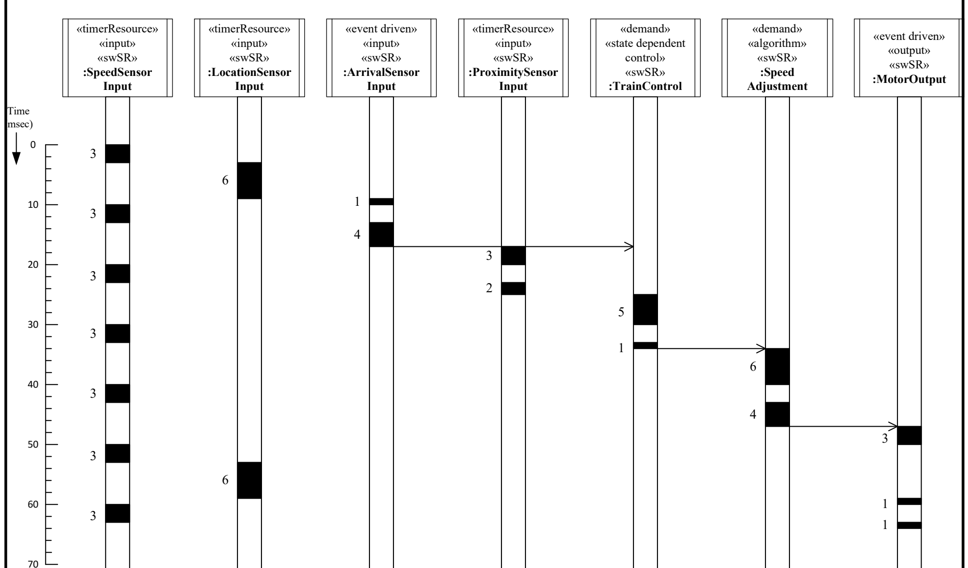
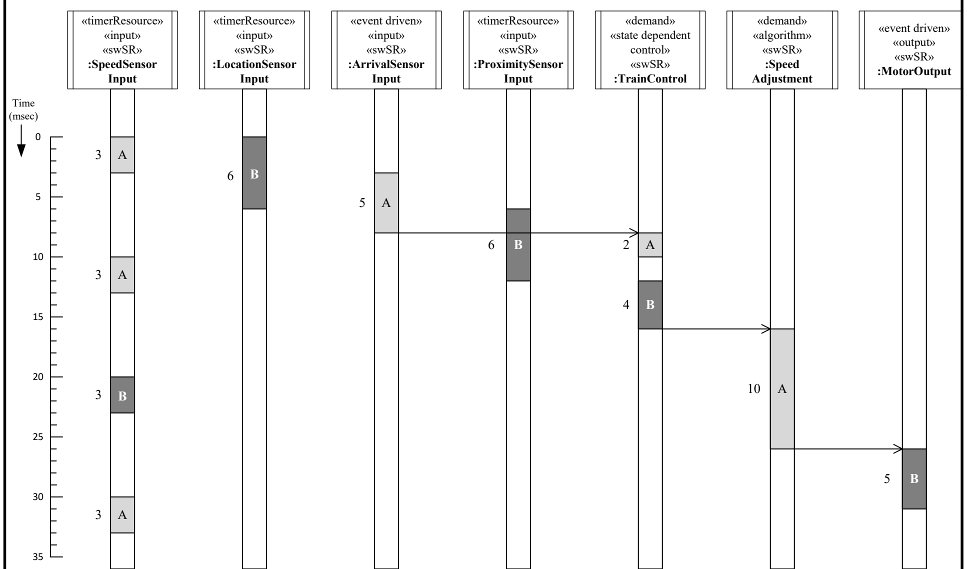


Figure 18.5 Timing Diagram for tasks in Train Control Subsystem executing on two CPUs



Review: Performance Analysis of COMET Designs

- Quantitative analysis of COMET designs
  - Allow early detection of potential performance problems
- Investigate alternatives
  - Software designs
  - Hardware configurations
- Approaches
  - Rate Monotonic Analysis (RMA)
  - Event sequence analysis
  - Combined RMA and Event sequence analysis

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**