IBM Software Group

Concurrency Architectures in the UML

Bruce Powel Douglass, Ph.D.
Chief Evangelist
IBM Rational
Bruce.Douglass@us.ibm.com
Twitter: @BruceDouglass
Yahoo: tech.groups.yahoo.com/group/RT-UML/
IBM: www-01.ibm.com/software/rational/leadership/thought/brucedouglass.html

Rational. software

Go to IBM

Innovation for a smarter planet

Basic Definitions

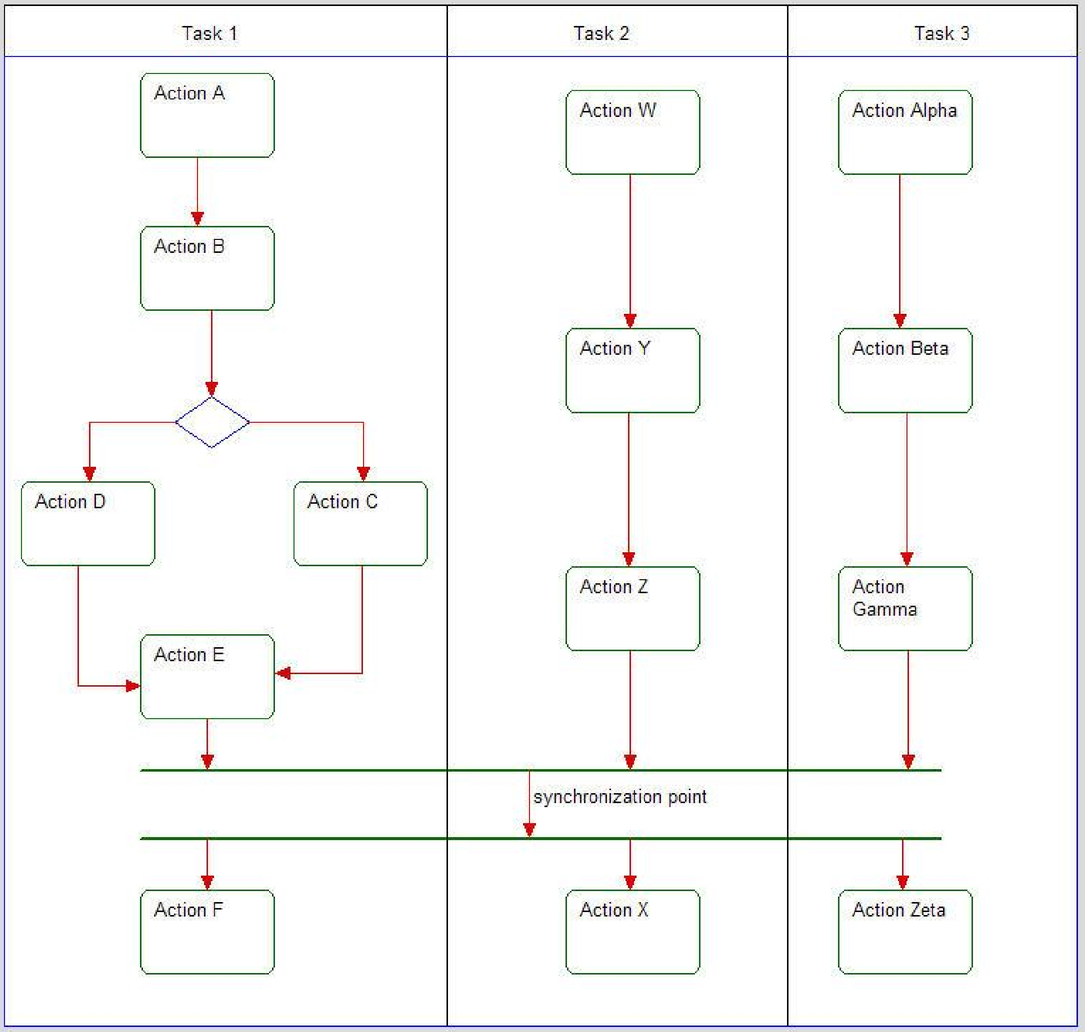
- Concurrency

> **Concurrency** refers to the simultaneous execution of action sequences

- Concurrency unit

> **A Concurrency Unit** (task or thread) has a sequence of actions in which the order of execution is known however the order of execution of actions in different concurrency units is “don’t know – don’t care” (except at explicit synchronization points)

Concurrency defines execution order dependencies



- What's the order of execution?
  - A then W then Alpha?
  - Alpha then Beta then Gamma then W then Y then A?
  - A then B then W then Y then Z then Alpha?
- ALL ARE CORRECT

If you care about the order between the sequences, then concurrency was the wrong choice!

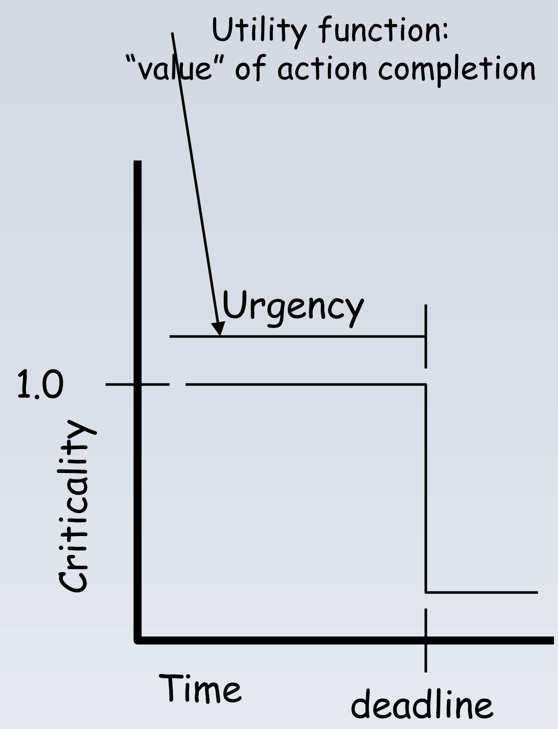
Basic Definitions

- Urgency

> **Urgency** refers to the nearness of a deadline

- Criticality

> **Criticality** refers to the importance of the task's correct and timely completion



Basic Definitions

- Deadline

> *A **deadline** is a point in time at which the completion of an action becomes incorrect or irrelevant*

- Priority

> ***Priority** is a numeric value used to determine which task, of the current ready-to-run task set will execute preferentially*

Basic Definitions

- Arrival Pattern

*The **arrival pattern** for a task or triggering event is either time-based (periodic) or event-based (aperiodic)*

- Synchronization Pattern

***Synchronization pattern** refers to the how the tasks execute during a rendezvous, e.g. synchronous, balking, waiting, or timed*

Basic Definitions

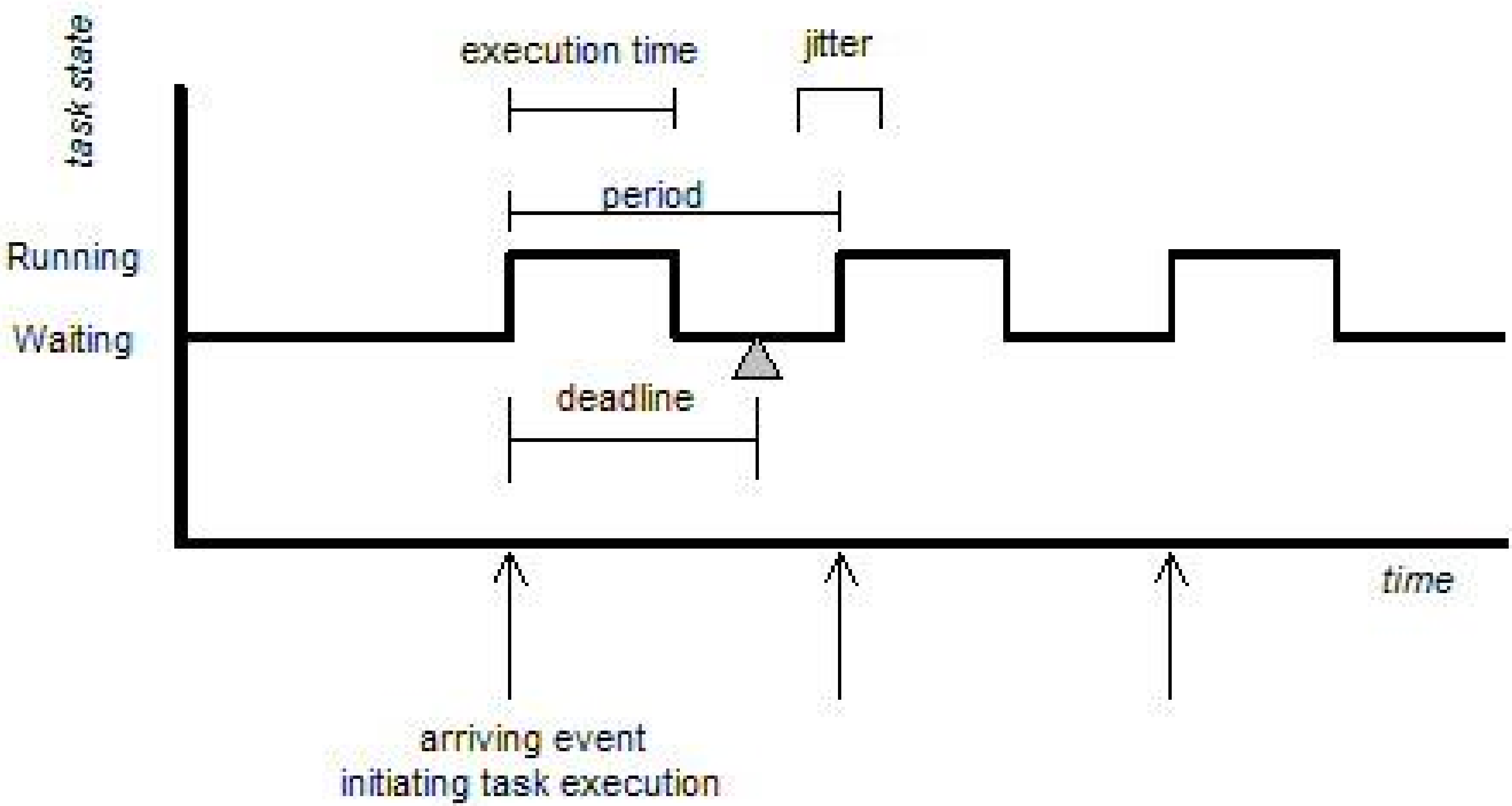
- Blocking Time

> *The **blocking time** for a task or action is the length of time it may be kept from executing because a lower priority task owns a required resource*

- Execution Time

> *The **execution time** for a task or action is the length of time it requires to complete execution*

Basic Definitions



Blocking

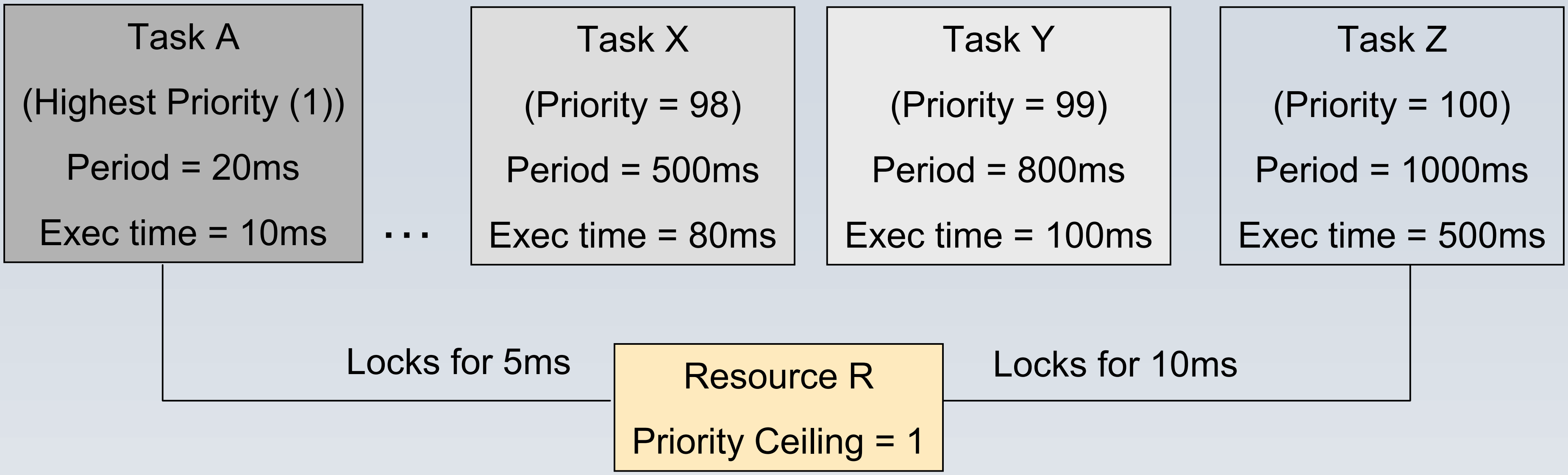
Task A	. . .	Task X	Task Y	Task Z
(Highest Priority (1))		(Priority = 98)	(Priority = 99)	(Priority = 100)
Period = 50ms		Period = 500ms	Period = 800ms	Period = 1000ms
Exec time = 10ms		Exec time = 80ms	Exec time = 100ms	Exec time = 500ms

Locks for 5ms — Resource R — Locks for 10ms

- What is the blocking time for Task Z?
- What is the blocking time for Task A?
- Will Task A always meet its deadlines?

This illustrates *unbounded priority inversion – this is ALWAYS a bad thing!*

Priority Inheritance



- The *Priority Ceiling* for a resource is the priority of the highest priority task that can ever access the resource (in this case "1")
  - While a lower priority task accesses the resource, it's priority is temporarily escalated to its resource ceiling and deescalated once it releases the resource
  - What is the blocking time for Task Z?
  - What is the blocking time for Task Y?
  - What is the blocking time for Task X?
  - What is the blocking time for Task A?
  - Will Task A always meet its deadlines?

Basic Definitions

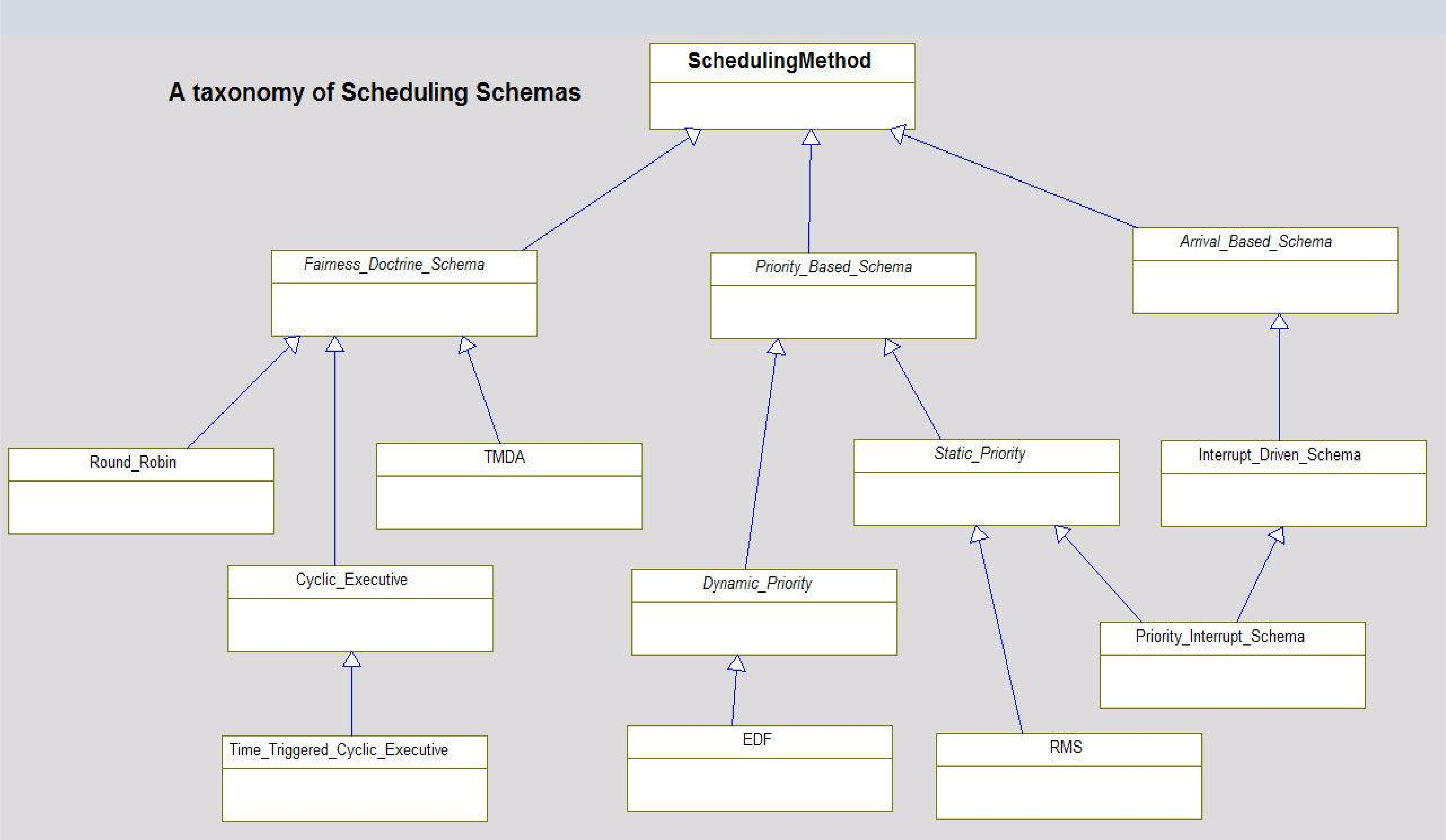
- Timeliness

__Timeliness__ refers to the ability of a task to predictably complete its execution prior to the elapse of its deadline

- Schedulability

A task set is __schedulable__ if it can be guaranteed that in all cases, all deadlines will be met

Task Scheduling Schemas



Task Scheduling Patterns

- Priority-based preemptive
  - Highest priority task not blocked runs preferentially
  - Good response time to high priority events
  - May be static (priority assigned at design) or dynamic (priority assigned at run-time)
- Non-preemptive
  - Round robin executes tasks in turn
  - May require “cooperative multitasking”
  - Single misbehaving task can hang the system
- Time Driven Multiplexed Architecture (TDMA)
  - Each task is given a specific time-slice in a round-robin fashion
  - Poor response time to events
- Cyclic executive
  - Run a set sequence in a particular order
  - Each task runs to completion
  - Poor response time to events
  - Highly predictable
- Interrupt
  - No scheduling per se, just a set of interrupt handlers
  - Requires that handlers are short (relative to arrival frequency) and atomic
  - Great response time to events of interest

Task Identification Strategies

Task Identification Strategy	**Description**
Single event groups	For simple systems, you may define a thread for each event type
Event source	Group all events from a single source together for a thread
Related information	For example, all numeric heart data
Independent Processing	When the actions can be clustered into sequences of actions in which the order *within* the sequences is defined but *between* these sequences is unimportant
Interface device	For example, a bus interface
Event properties	**Events with the same period, or aperiodic events***
Target object	For example, waveform queue or trend database
Safety Level	For example, BIT, redundant thread processing, watchdog tasks

Representing Concurrency in the UML

- Concurrency Units
  - Active classes
    - This is the primary means for representing task or thread concurrency in the system
    - Parallel operator in sequence diagrams (lifelines are instance roles typed by classes)
  - Other means represent “logical concurrency” in the “independence of execution sequence” sense and *almost never* used to represent actual threads
    - Forks/joins in activity diagrams
    - Orthogonal regions (and-states) in state machines

Representing Concurrency in the UML

- Concurrency metadata representation as
  - Constraints – user-defined “well-formedness rules”
  - Tags – value-name pairs added to model elements
- Typical concurrency metadata include
  - Priority
  - Period
  - Execution time
  - Worst case execution time
  - Worst case blocking time
  - Deadline
  - Locking time
  - Priority ceiling
  - Access control method

Active Classes are the Basis of UML Concurrency

- In UML 1.x the unit of concurrency was called the «active» class, which is normally a structured class (i.e. a class with parts)
- In UML 1.x the notation was to use a heavy border

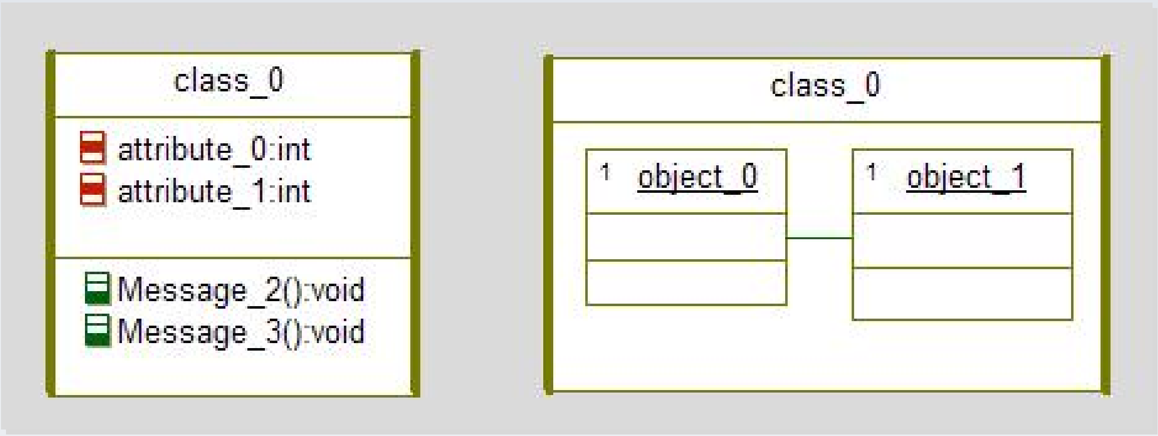
UML 1.x Active Class

- In UML 2.0 the notation has changed to double vertical lines

UML 2.0 Active Class

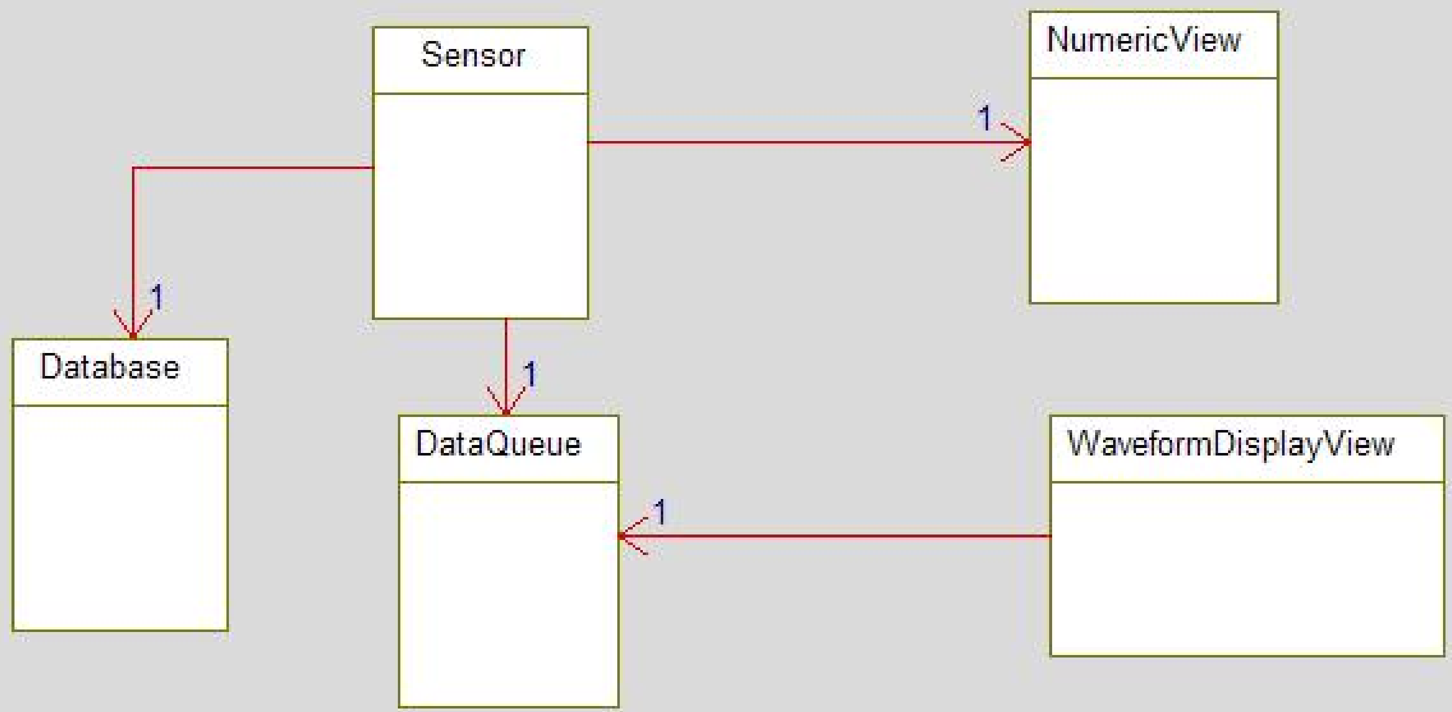
Active Classes

- «active» classes specify the metadata, structure and behavior of «active» objects
- «active» classes
  - Contain internal parts (object roles typed by classes) that execute in the thread context of the «active» class
  - Own an OS thread in which it (and its parts) executes
  - Own an event queue for their state machines and all state machines within them
  - May contain parts that are themselves «active»



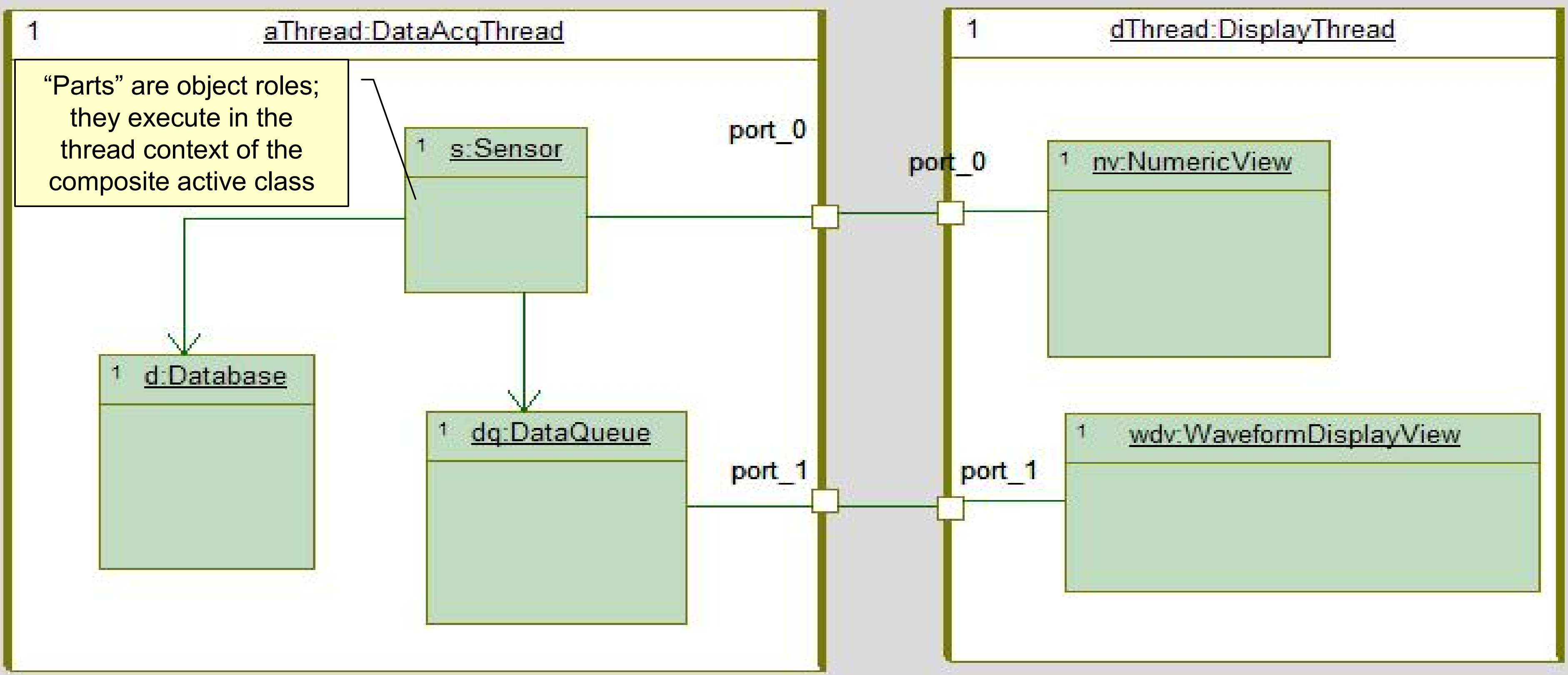
Concurrency Model

- Active class is a stereotype of a class which owns the root of a thread
- Active classes normally aggregate passive classes via composition relations
- Standard icon is a class box with heavy line



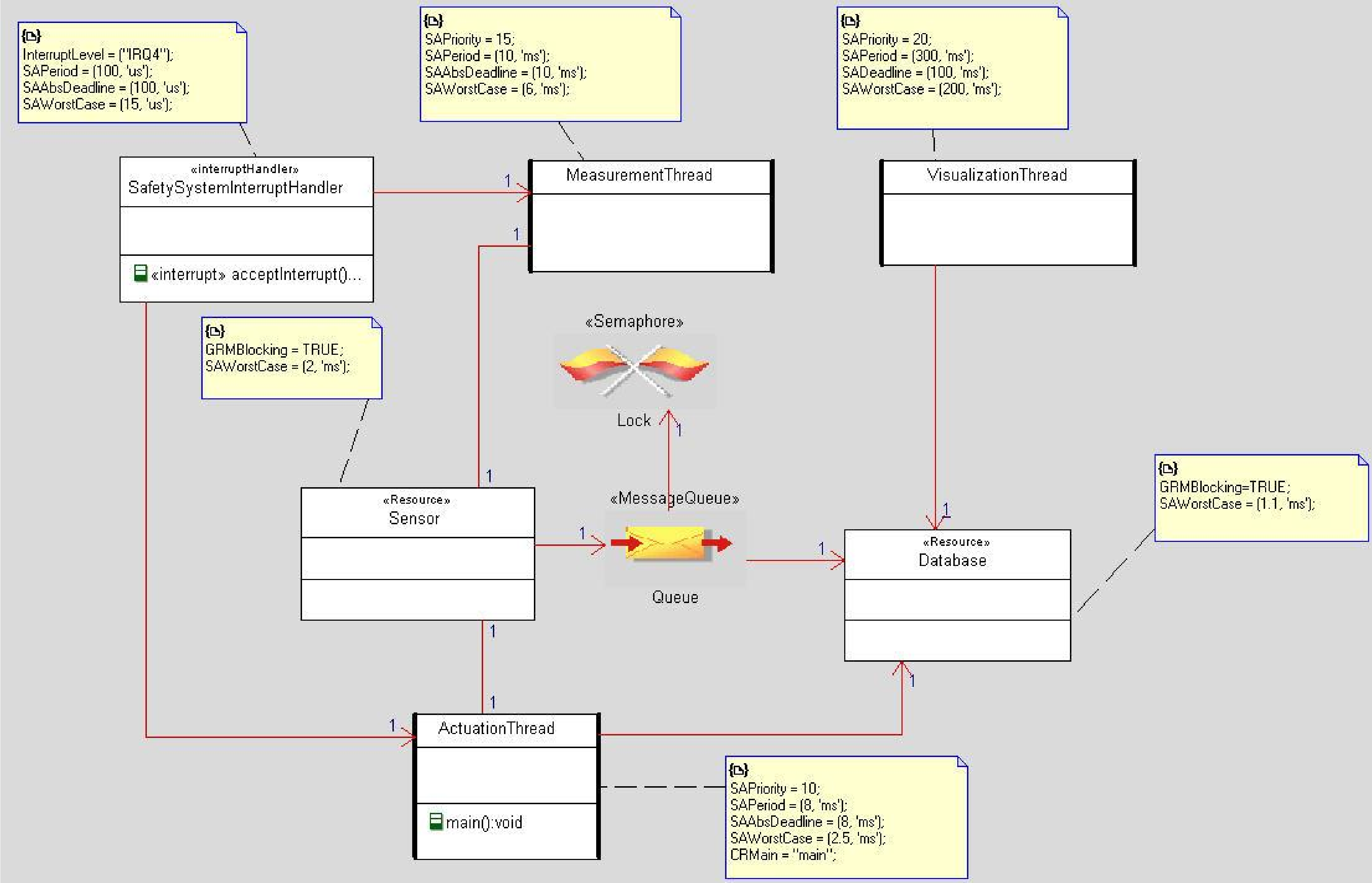
Concurrency Model

- Active class is a stereotype of a class which owns the root of a thread
- Active class normally aggregate passive classes via composition relation

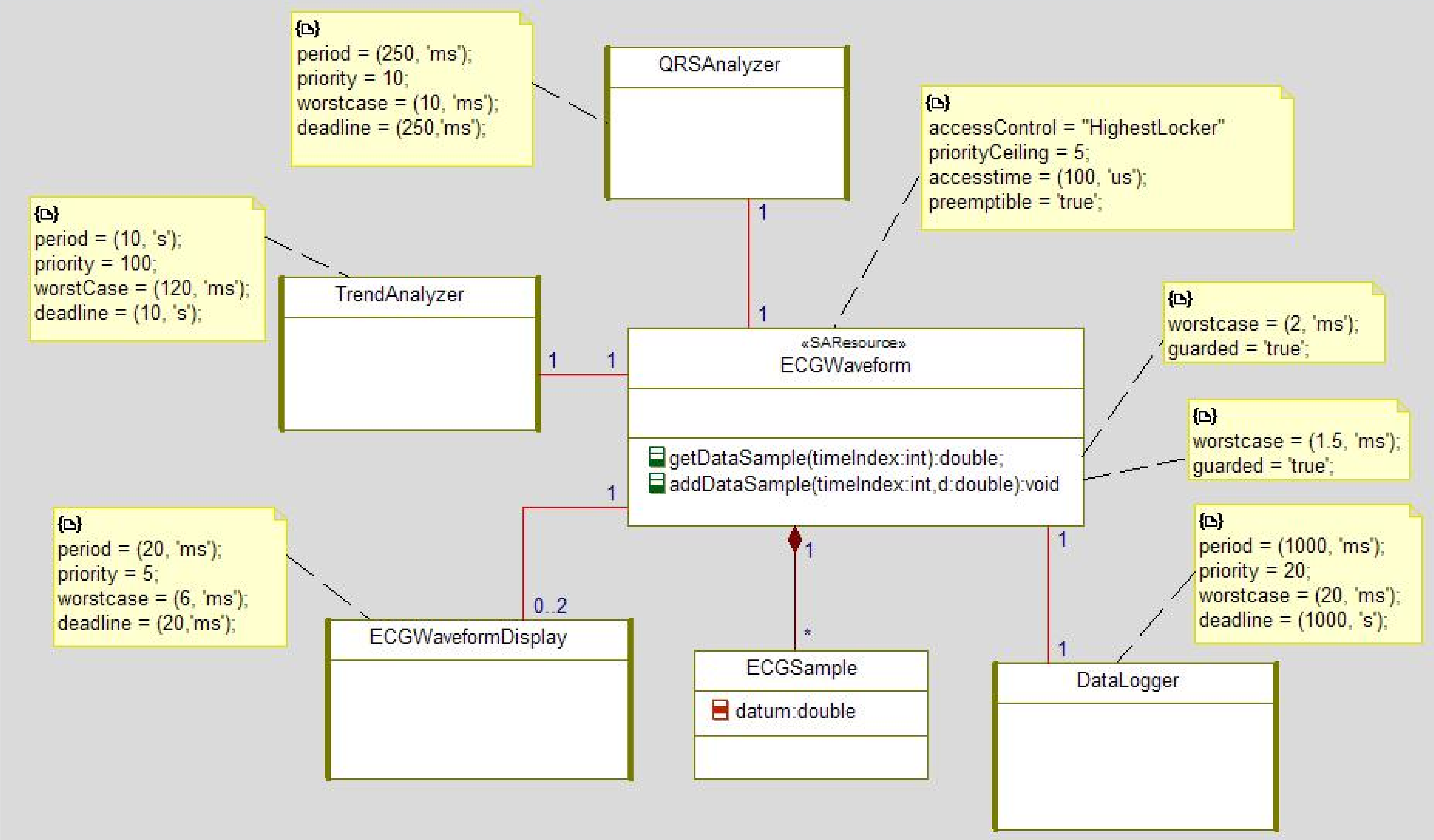


Task Diagram

- A task diagram is a class diagram that shows only model elements related to the concurrency model
  - Active objects
  - Semaphore objects
  - Message and data queues
  - Concurrency metadata in constraints and tagged values
- May use opaque or transparent interfaces


InterruptLevel = ("IRQ4");
SAPeriod = (100, 'us');
SAAbsDeadline = (100, 'us');
SAWorstCase = (15, 'us');
SAPriority = 15;
SAPeriod = (10, 'ms');
SAAbsDeadline = (10, 'ms');
SAWorstCase = (6, 'ms');
SAPriority = 20;
SAPeriod = (300, 'ms');
SADeadline = (100, 'ms');
SAWorstCase = (200, 'ms');
«interruptHandler»
SafetySystemInterruptHandler
«interrupt» acceptInterrupt()...
MeasurementThread
VisualizationThread
GRMBlocking = TRUE;
SAWorstCase = (2, 'ms');
«Semaphore»
Lock
«MessageQueue»
Queue
«Resource»
Sensor
«Resource»
Database
GRMBlocking=TRUE;
SAWorstCase = (1.1, 'ms');
ActuationThread
main():void
SAPriority = 10;
SAPeriod = (8, 'ms');
SAAbsDeadline = (8, 'ms');
SAWorstCase = (2.5, 'ms');
CRMain = "main";

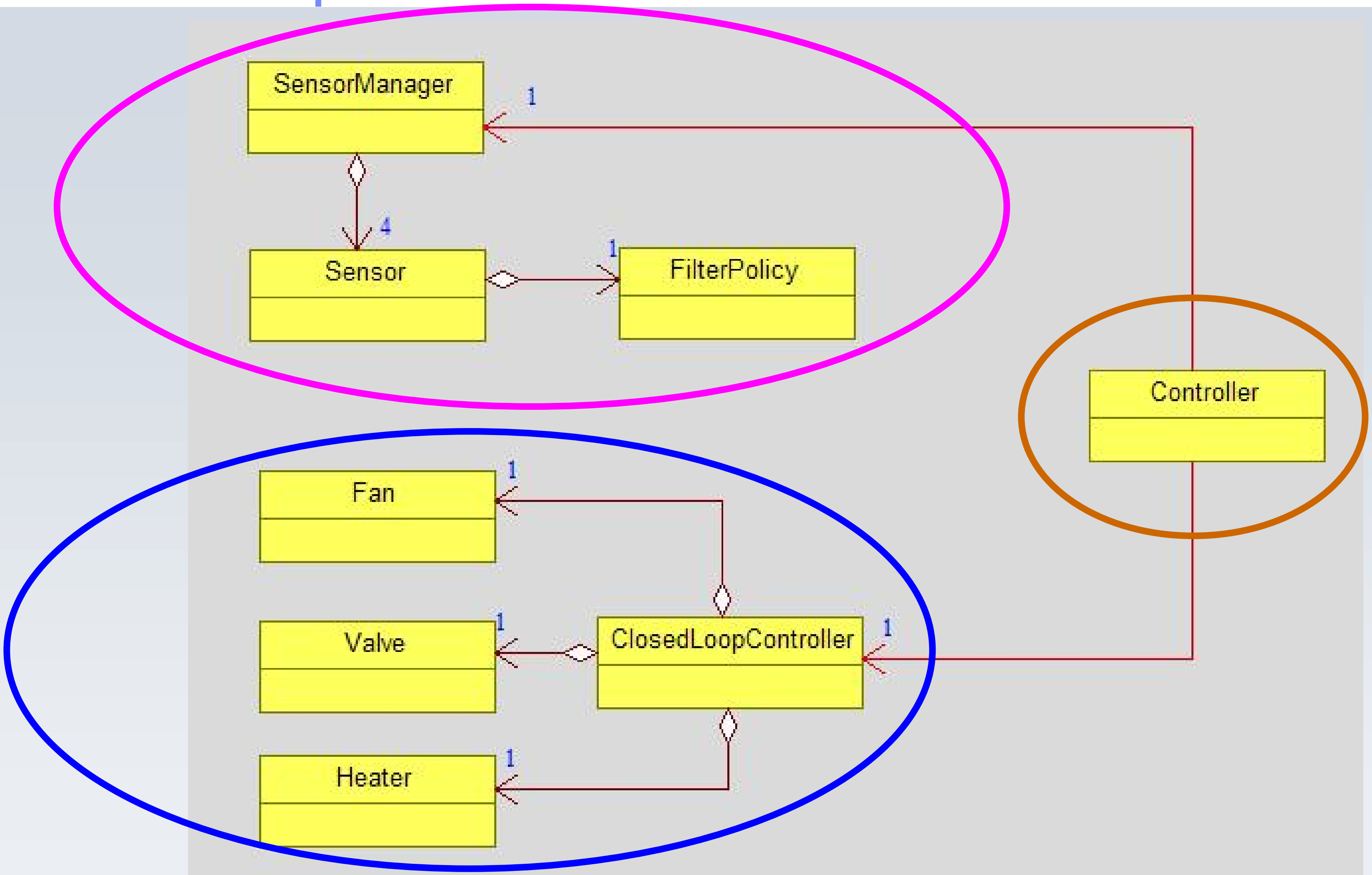
Another Task Diagram



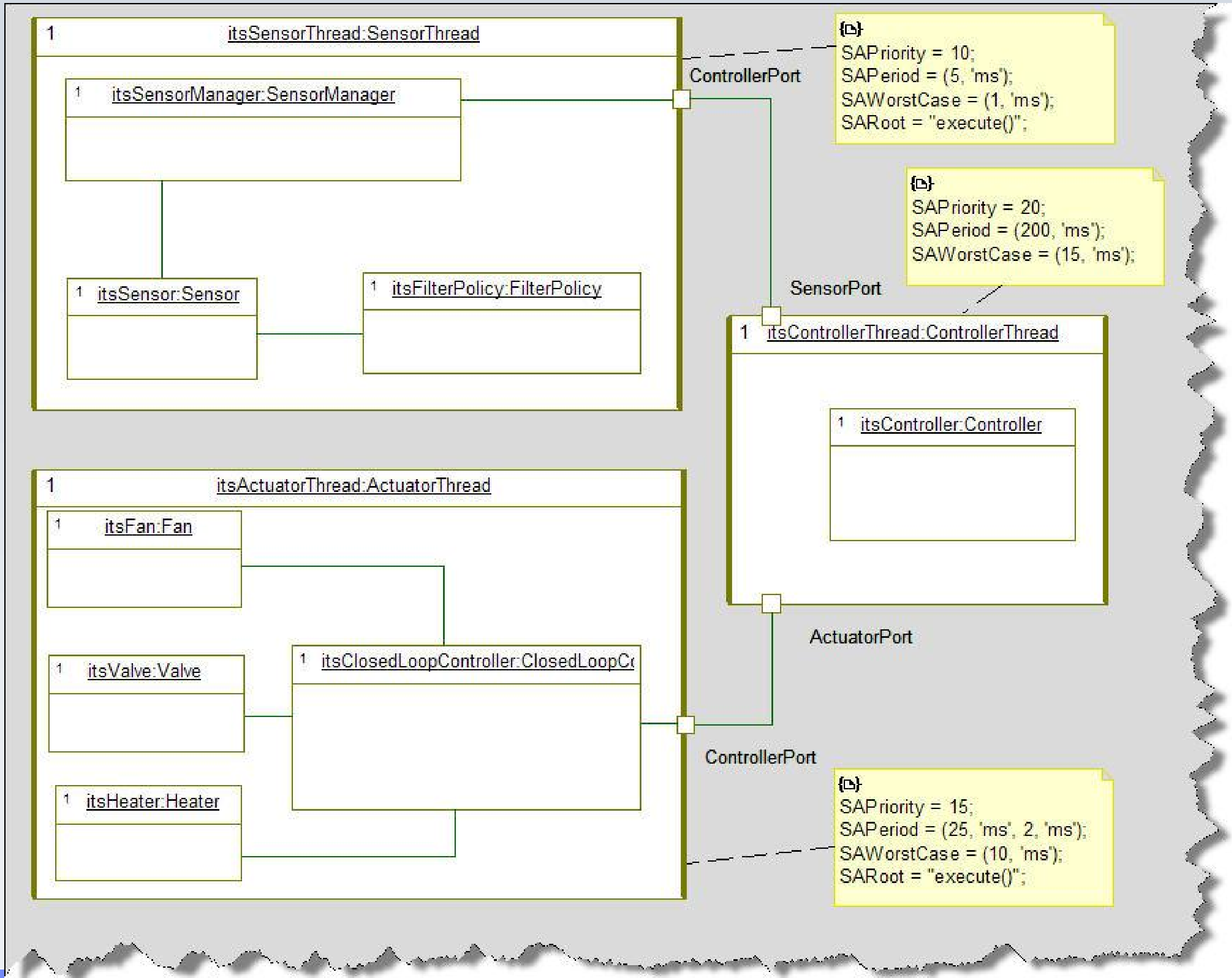
Assigning Objects to Tasks

- Recommendation: rather than make an existing class active, add a new class to own the thread
  - Put the relevant parts (typed by the classes) inside as parts
- Active classes are normally composites that delegate responsibilities to their internal parts
  - The relation between the classes is *composition*
  - The relation between the structure class and its parts is whole-part
- Semantic classes provide
  - Decomposition of complex actions required for the thread’s action and the information to be used
- Rendezvous classes provide
  - Management of the interaction between threads
  - E.g. queues, semaphores, barriers, etc.
  - Normally execute in the thread of the caller

Task Example

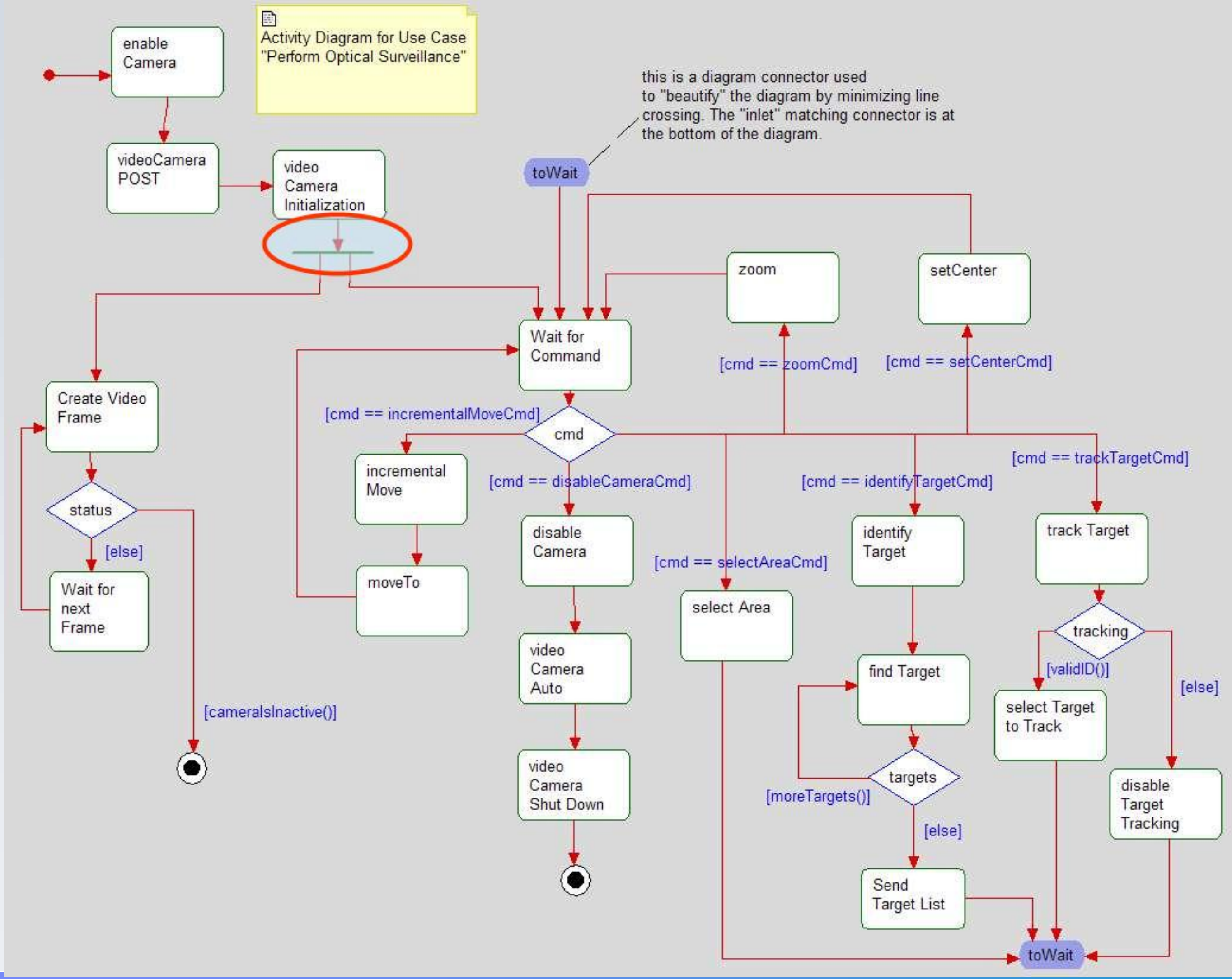


Task Example with Concurrency

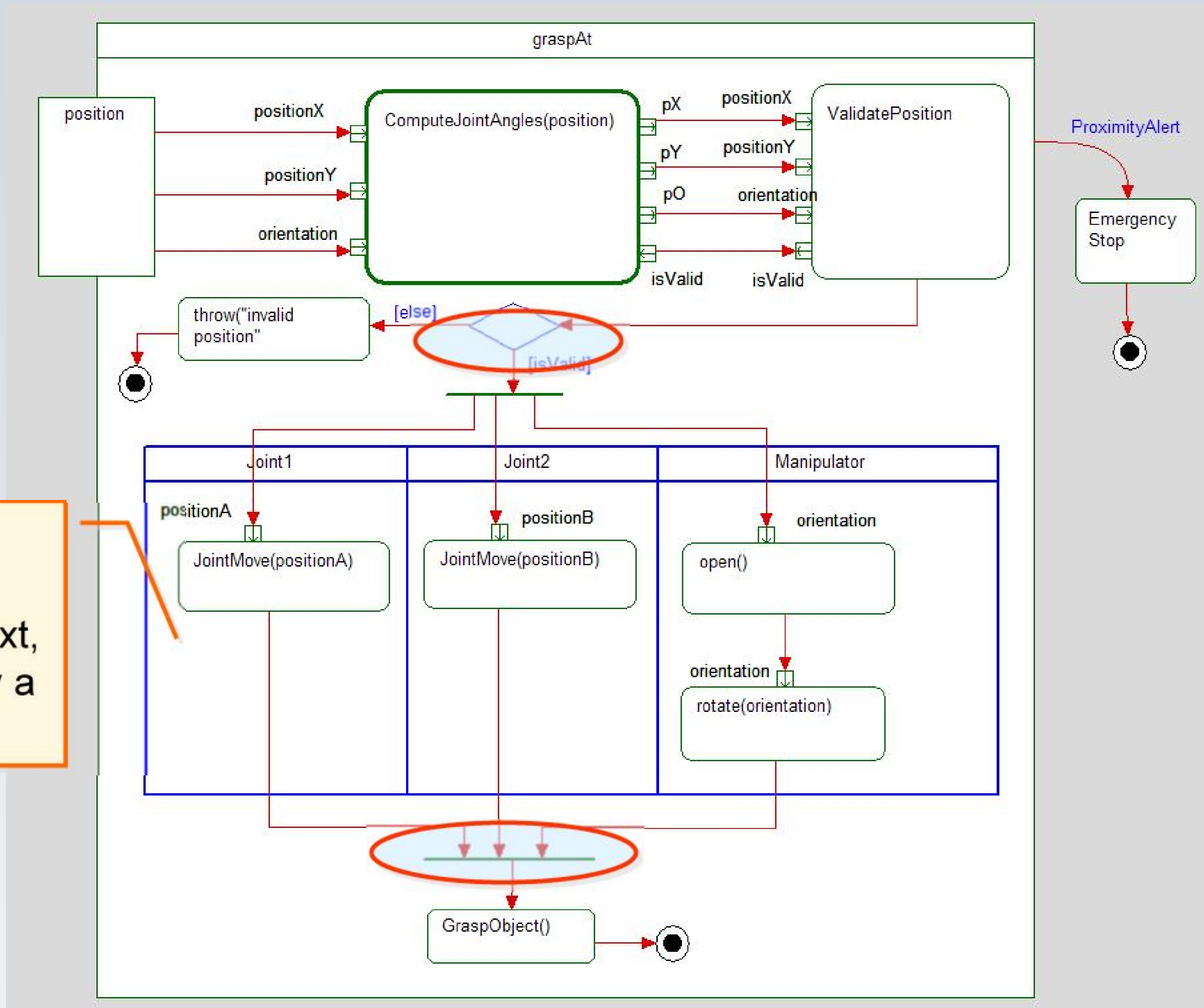


Concurrency in Activity Diagrams

- Forks and joins indicate concurrency boundaries



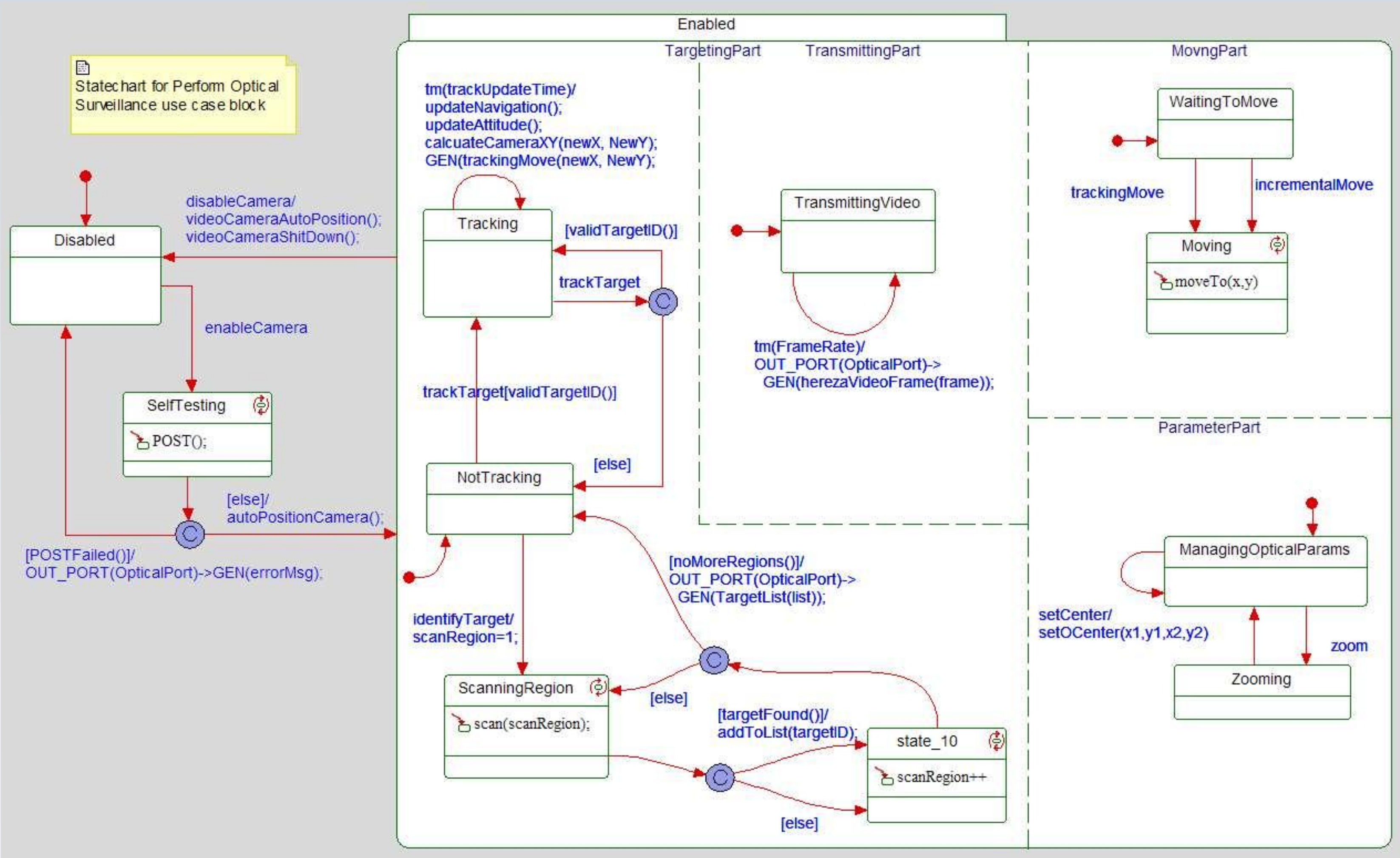
Concurrency in Activity Diagrams



Swimlanes indicate an execution context, most commonly a class

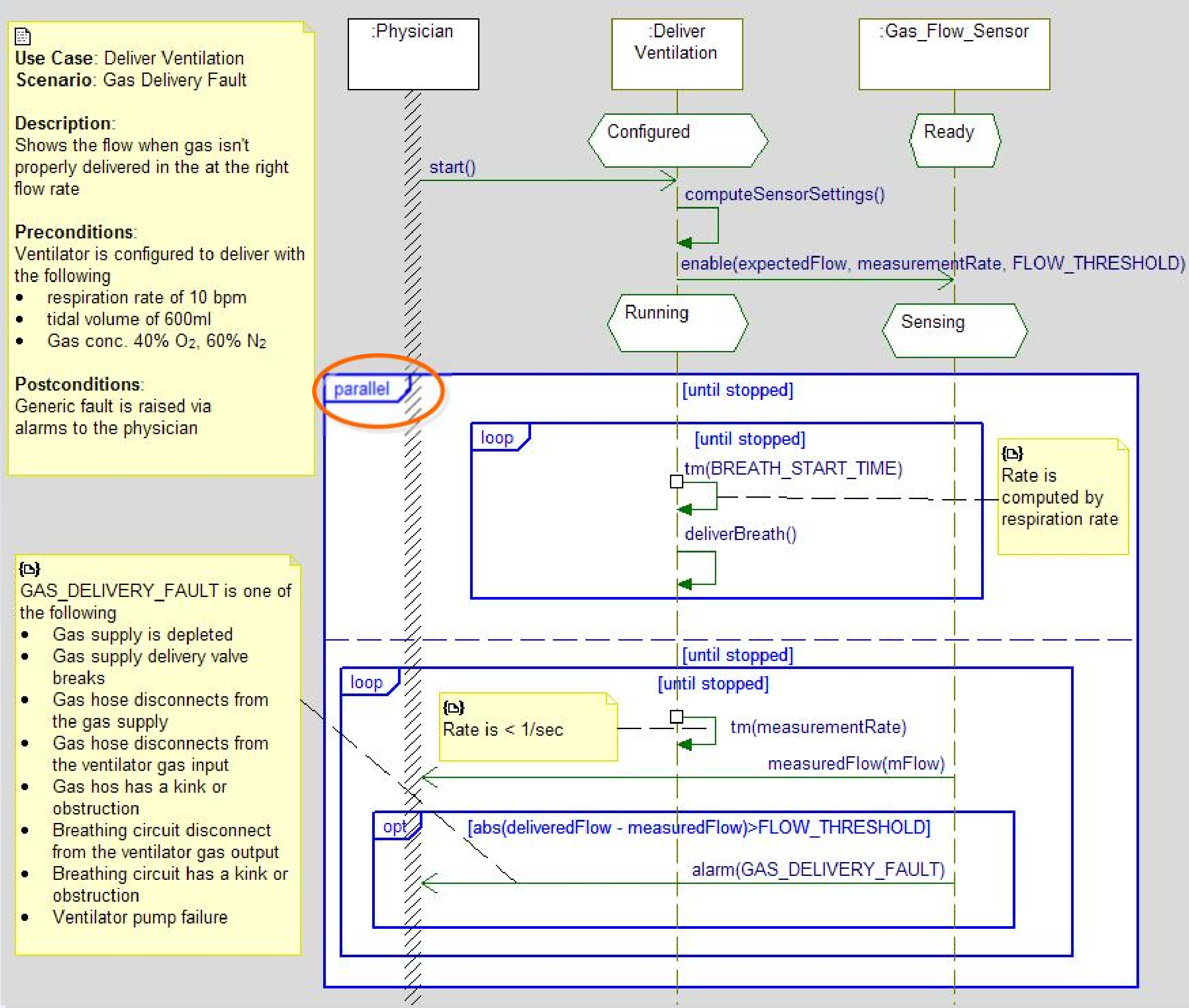
Concurrency in State Machines

- “And-states” indicate regions between which order of execution is not specified



Concurrency on Sequence Diagrams

- The *parallel* (or *para*) operator indicates parallel regions.
- The order within a region is specified by "partial ordering"
- The order of messages between parallel regions is unspecified



SPT and MARTE

- The UML Profile for Schedulability, Performance, and Time (SPT) is a UML 1.x profile for specifying timeliness metadata for models
  - The SPT was released as a finalized standard in 2003
- The Model Analysis for Real-Time and Embedded systems (MARTE) is a UML 2.x profile for specifying timeliness metadata for models
  - MARTE is still in the process of being finalized
- Both standards are *profiles:* minor extensions of the UML metamodel, with stereotypes, tags, and constraints
  - Note: Profiles must be compliant with the UML metamodel

The UML Profile for Schedulability, Performance, and Time

- Submitted in response to an OMG RFP
  - RFP for a UML Profile for Schedulability, Performance, and Time (OMG document ad/99-03-13).
  - Standardized in 2003
  - New standard being readied for UML 2
- Submitters (in alphabetical order):
  - Artisan Software Tools, Inc.
  - Telelogic Inc.
  - Rational Software Corporation, Inc.
  - Telelogic AB
  - Timesys Corporation
  - TriPacific Software

Goal of the SPT Profile

Note: The UML is considered to be fully adequate to model real-time and embedded systems. The profile is NOT necessary to make UML *applicable* to real-time systems.

- RFP calls for "proposals for a UML profile that defines standard paradigms of use for modeling of *time-, schedulability-, and performance-related aspects* of real-time systems"
  - Define some *standard* means to capture real-time modeling concerns
  - Permit exchange of model information between tools, e.g.
    - Between design automation tools
    - Between design automation and schedulability tools
  - Facilitate communication of design intent among engineering staff and other stakeholders

Guiding Principles

- Do not change the UML unless absolutely required
- Do not limit the way UML is used.
- Provide the ability to annotate a UML model to allow for [quantitative] analysis in a standard way.
- Do not require a deep understanding of applicable analysis techniques, e.g.
  - Rate monotonic analysis
  - Queuing theory

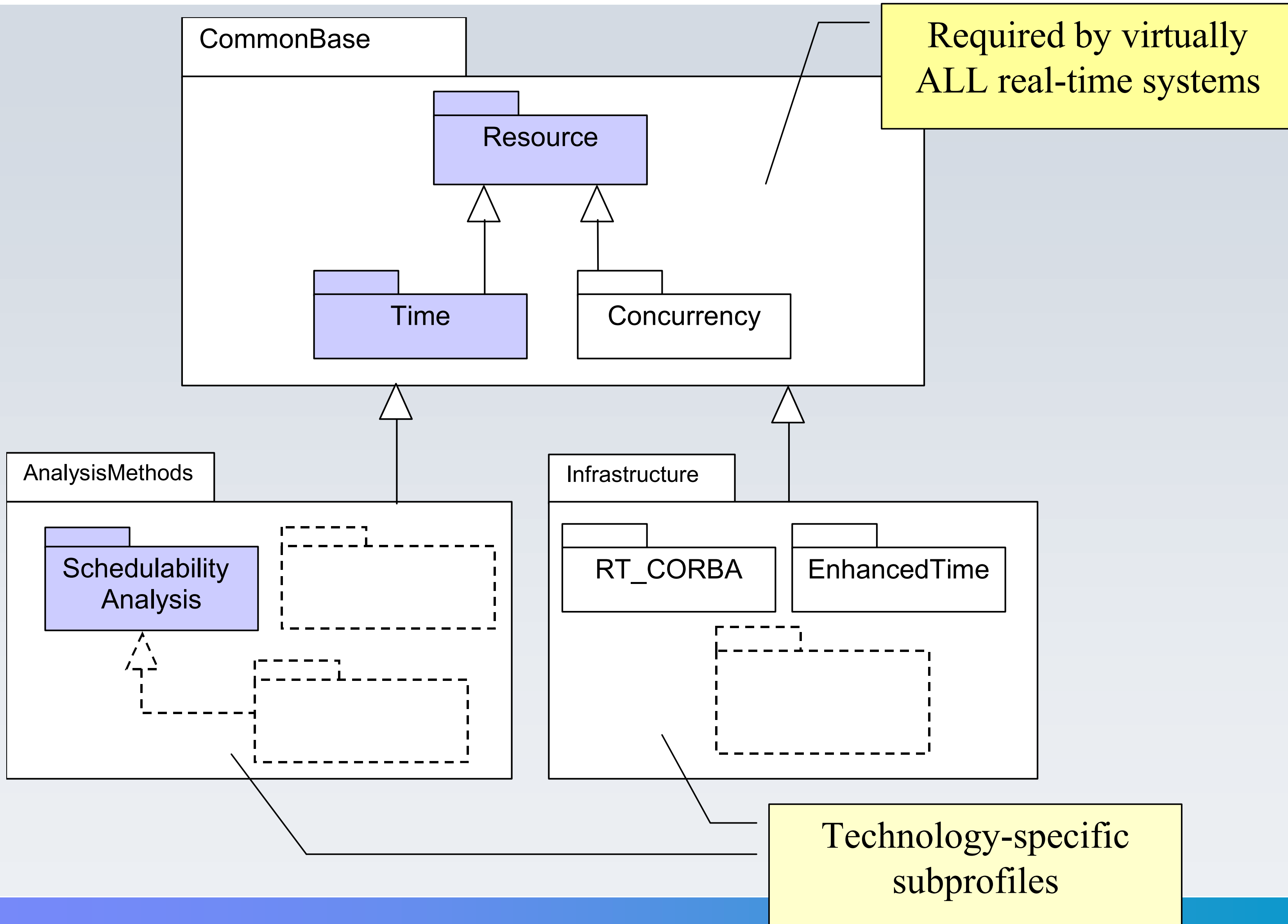
(More) Guiding Principles

- Simple analysis should be simple to do. More complex analysis may require more work.
- Support, but do not restrict modeling to existing techniques.
  - E.g. RMA, DMA
- Automated tools should be able to influence the UML model.
  - E.g. update priorities of task threads so that they become schedulable
- Support both model analysis and synthesis

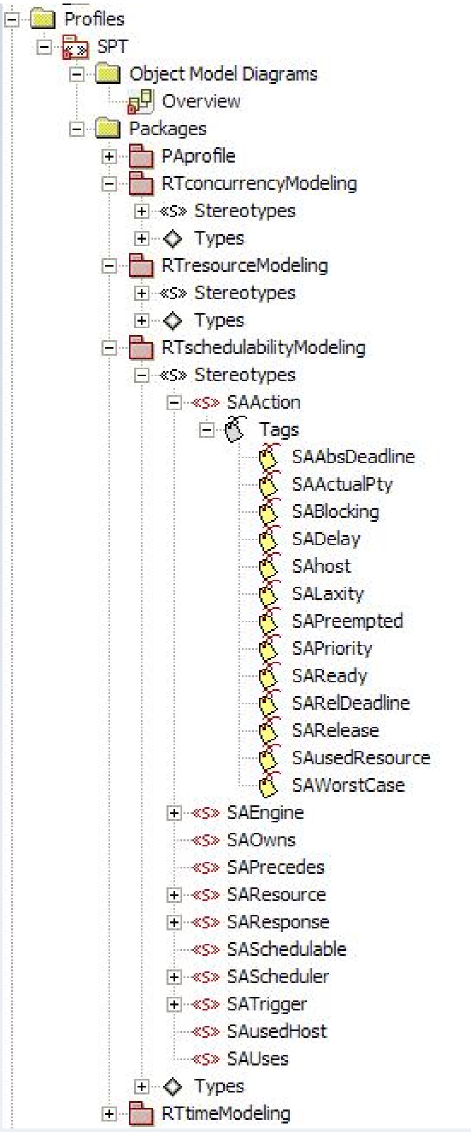
General Approach

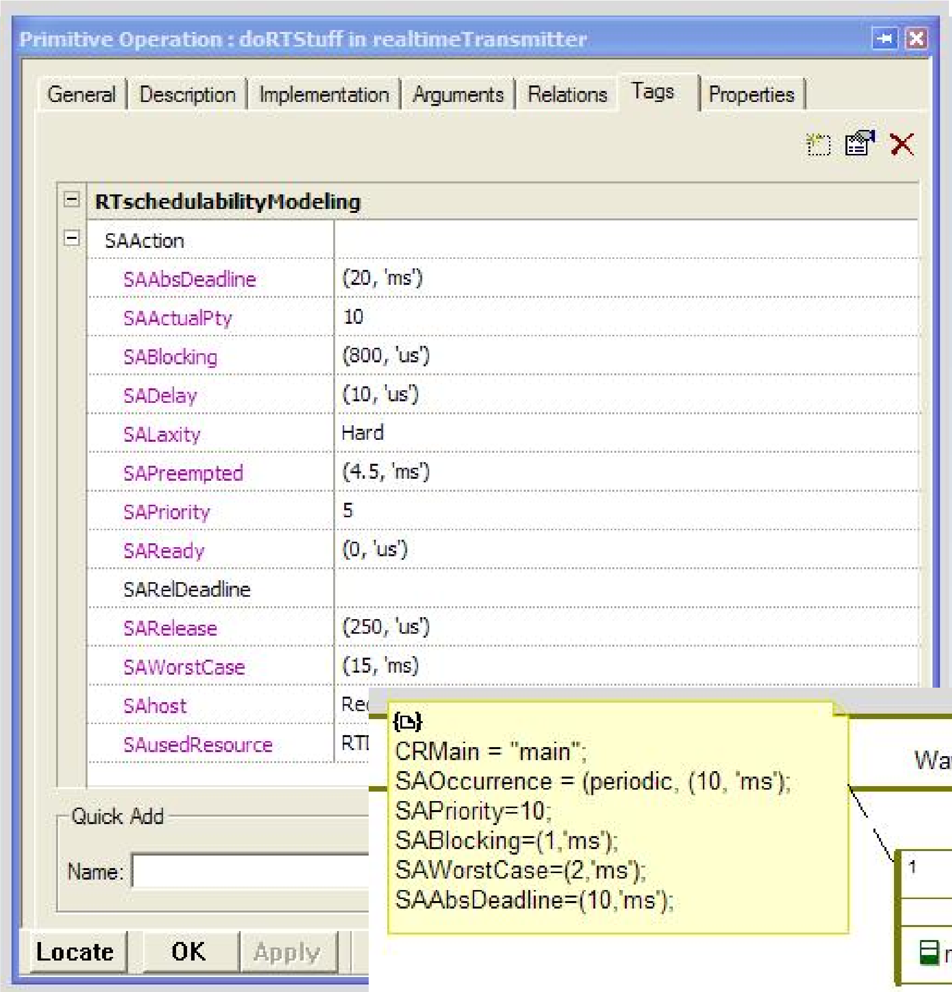
- Use light-weight extensions to add standard modeling approaches and elements
  - Stereotypes, e.g. resources
  - Tagged values, e.g. QoS properties
- Divide submission into sub-profiles to allow easier comprehension and usage of relevant parts

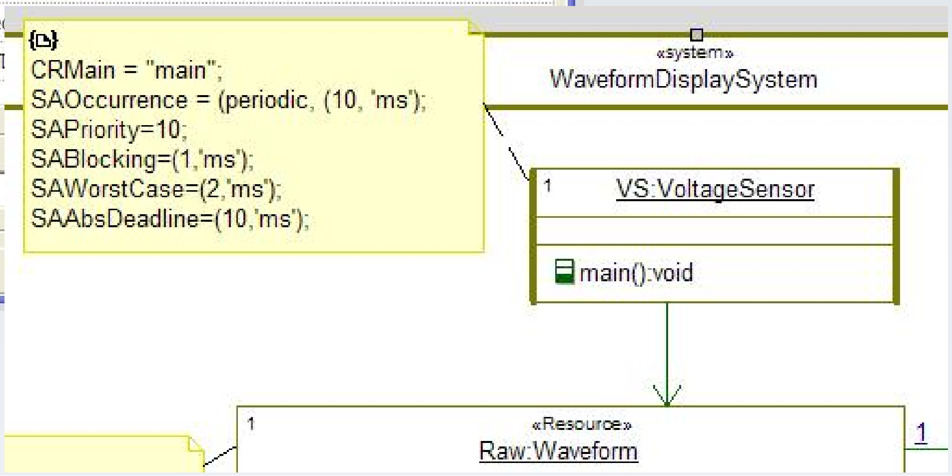
SPT Profile Structure

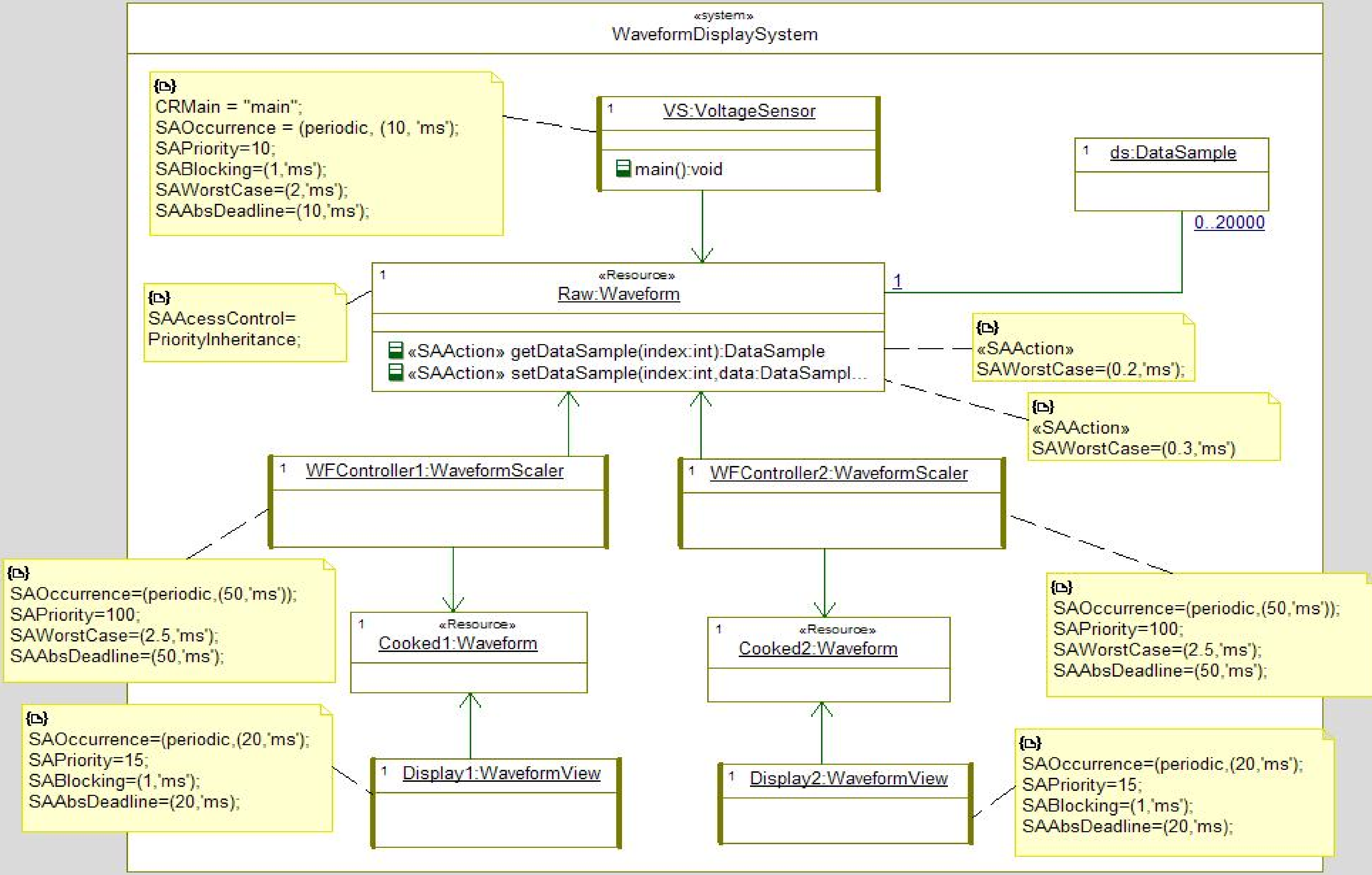


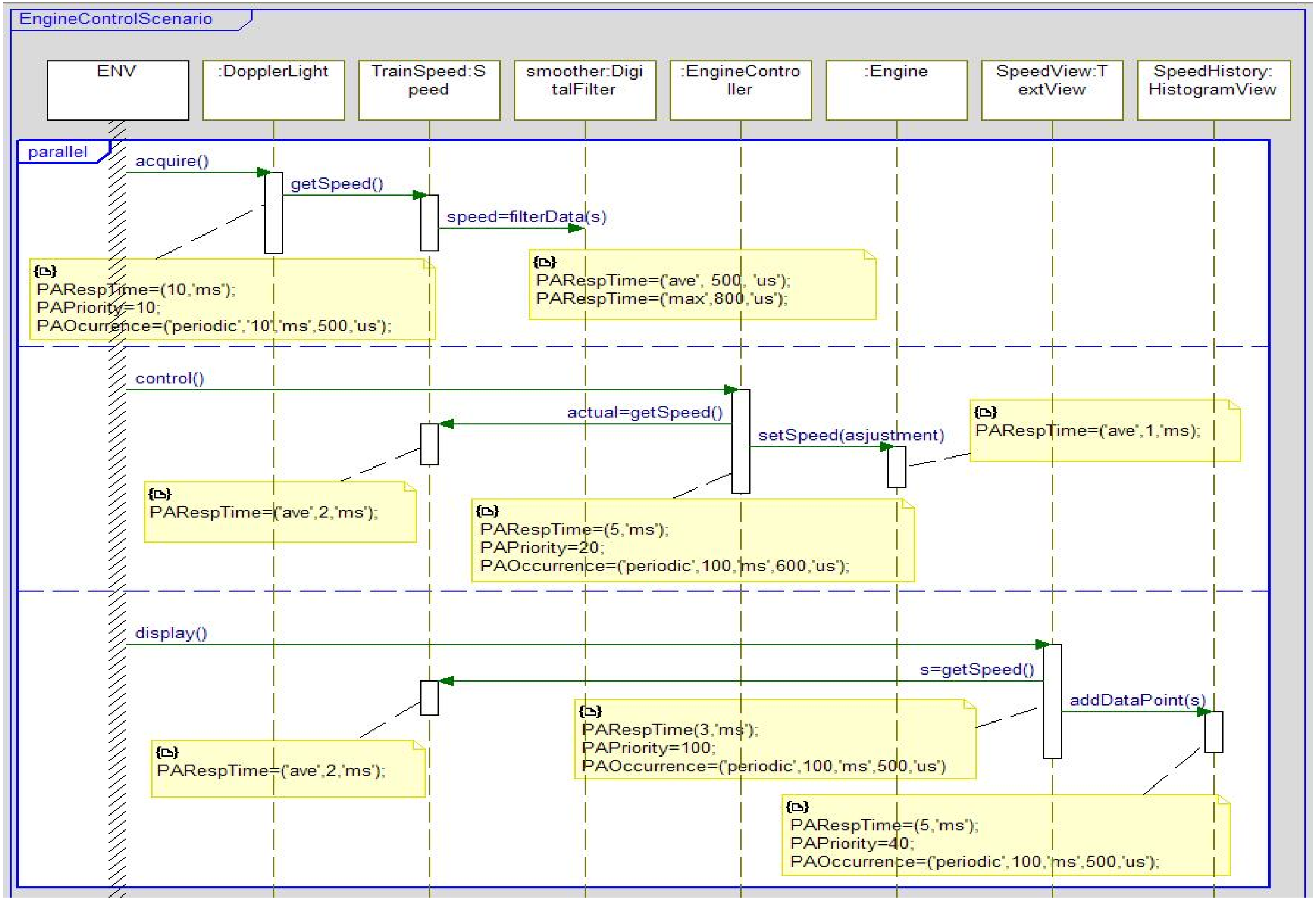
SPT Profile


Profiles
SPT
Object Model Diagrams
Overview
Packages
PAprofile
RTconcurrencyModeling
«S» Stereotypes
Types
RTresourceModeling
«S» Stereotypes
Types
RTschedulabilityModeling
«S» Stereotypes
«S» SAAction
Tags
SAAbsDeadline
SAActualPty
SABlocking
SADelay
SAhost
SALaxity
SAPreempted
SAPriority
SAReady
SARelDeadline
SARelease
SAusedResource
SAWorstCase
«S» SAEngine
«S» SAOwns
«S» SAPrecedes
«S» SAResource
«S» SAResponse
«S» SASchedulable
«S» SAScheduler
«S» SATrigger
«S» SAusedHost
«S» SAUses
Types
RTtimeModeling

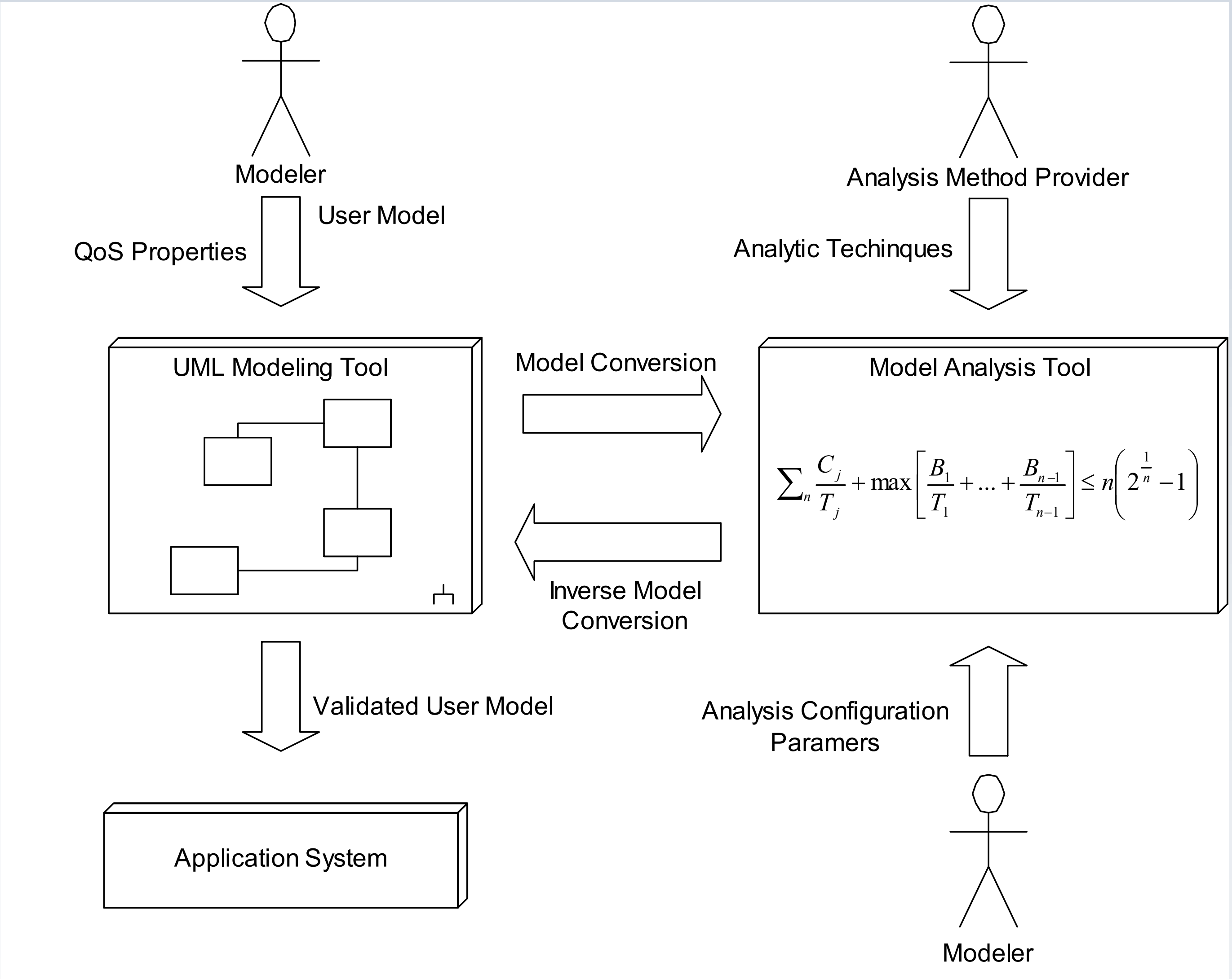

Primitive Operation : doRTStuff in realtimeTransmitter
General
Description
Implementation
Arguments
Relations
Tags
Properties
RTschedulabilityModeling
SAAction
SAAbsDeadline (20, 'ms')
SAActualPty 10
SABlocking (800, 'us')
SADelay (10, 'us')
SALaxity Hard
SAPreempted (4.5, 'ms')
SAPriority 5
SAReady (0, 'us')
SARelDeadline
SARelease (250, 'us')
SAWorstCase (15, 'ms)
SAhost
SAusedResource
Quick Add
Name:
Locate
OK
Apply


CRMain = "main";
SAOccurrence = (periodic, (10, 'ms');
SAPriority=10;
SABlocking=(1,'ms');
SAWorstCase=(2,'ms');
SAAbsDeadline=(10,'ms');
«system»
WaveformDisplaySystem
1
VS:VoltageSensor
main():void
1
«Resource»
Raw:Waveform
1


«system»
WaveformDisplaySystem
CRMain = "main";
SAOccurrence = (periodic, (10, 'ms');
SAPriority=10;
SABlocking=(1,'ms');
SAWorstCase=(2,'ms');
SAAbsDeadline=(10,'ms');
1 VS:VoltageSensor
main():void
1 ds:DataSample
0..20000
1
«Resource»
Raw:Waveform
SAAcessControl=
PriorityInheritance;
«SAAction» getDataSample(index:int):DataSample
«SAAction» setDataSample(index:int,data:DataSampl...
«SAAction»
SAWorstCase=(0.2,'ms');
«SAAction»
SAWorstCase=(0.3,'ms')
1 WFController1:WaveformScaler
1 WFController2:WaveformScaler
SAOccurrence=(periodic,(50,'ms'));
SAPriority=100;
SAWorstCase=(2.5,'ms');
SAAbsDeadline=(50,'ms');
1 «Resource» Cooked1:Waveform
1 «Resource» Cooked2:Waveform
SAOccurrence=(periodic,(50,'ms'));
SAPriority=100;
SAWorstCase=(2.5,'ms');
SAAbsDeadline=(50,'ms');
SAOccurrence=(periodic,(20,'ms');
SAPriority=15;
SABlocking=(1,'ms');
SAAbsDeadline=(20,'ms);
1 Display1:WaveformView
1 Display2:WaveformView
SAOccurrence=(periodic,(20,'ms');
SAPriority=15;
SABlocking=(1,'ms');
SAAbsDeadline=(20,'ms);


EngineControlScenario
ENV
:DopplerLight
TrainSpeed:Speed
smoother:DigitalFilter
:EngineController
:Engine
SpeedView:TextView
SpeedHistory:HistogramView
parallel
acquire()
getSpeed()
speed=filterData(s)
PARespTime=(10,'ms');
PAPriority=10;
PAOcurrence=('periodic','10','ms',500,'us');
PARespTime=('ave', 500, 'us');
PARespTime=('max',800,'us');
control()
actual=getSpeed()
setSpeed(asjustment)
PARespTime=('ave',1,'ms');
PARespTime=('ave',2,'ms');
PARespTime=(5,'ms');
PAPriority=20;
PAOccurrence=('periodic',100,'ms',600,'us');
display()
s=getSpeed()
addDataPoint(s)
PARespTime(3,'ms');
PAPriority=100;
PAOccurrence=('periodic',100,'ms',500,'us')
PARespTime=('ave',2,'ms');
PARespTime=(5,'ms');
PAPriority=40;
PAOccurrence=('periodic',100,'ms',500,'us');

Model Processing

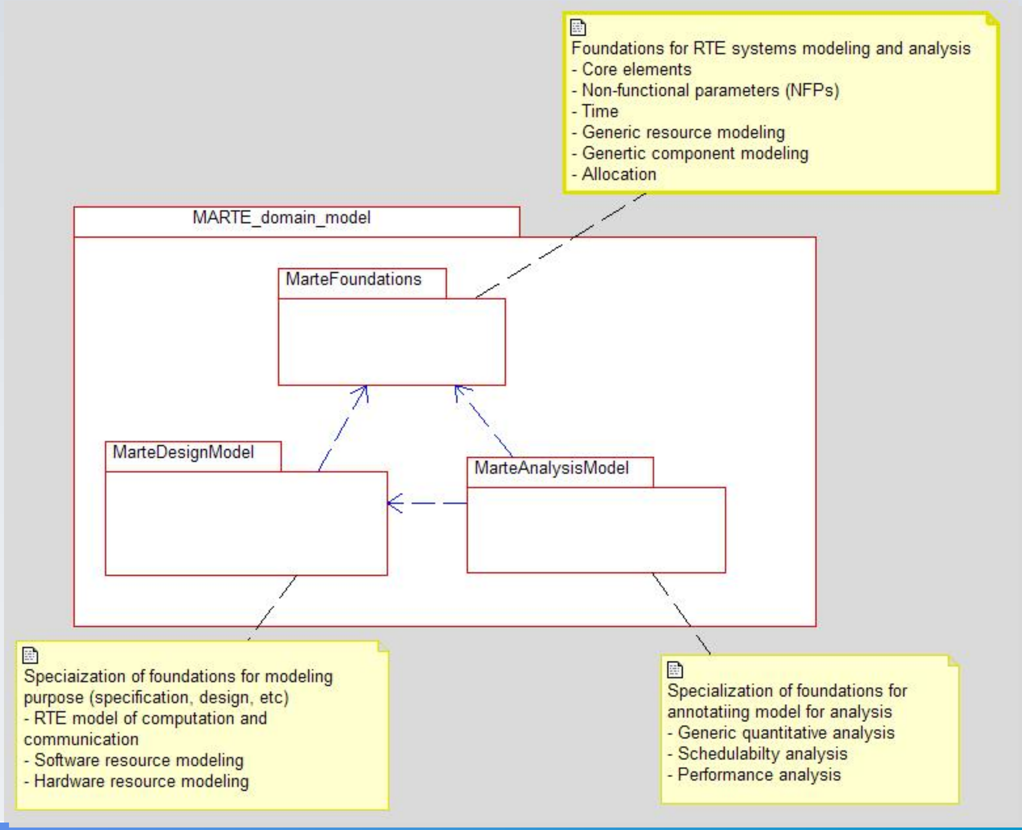


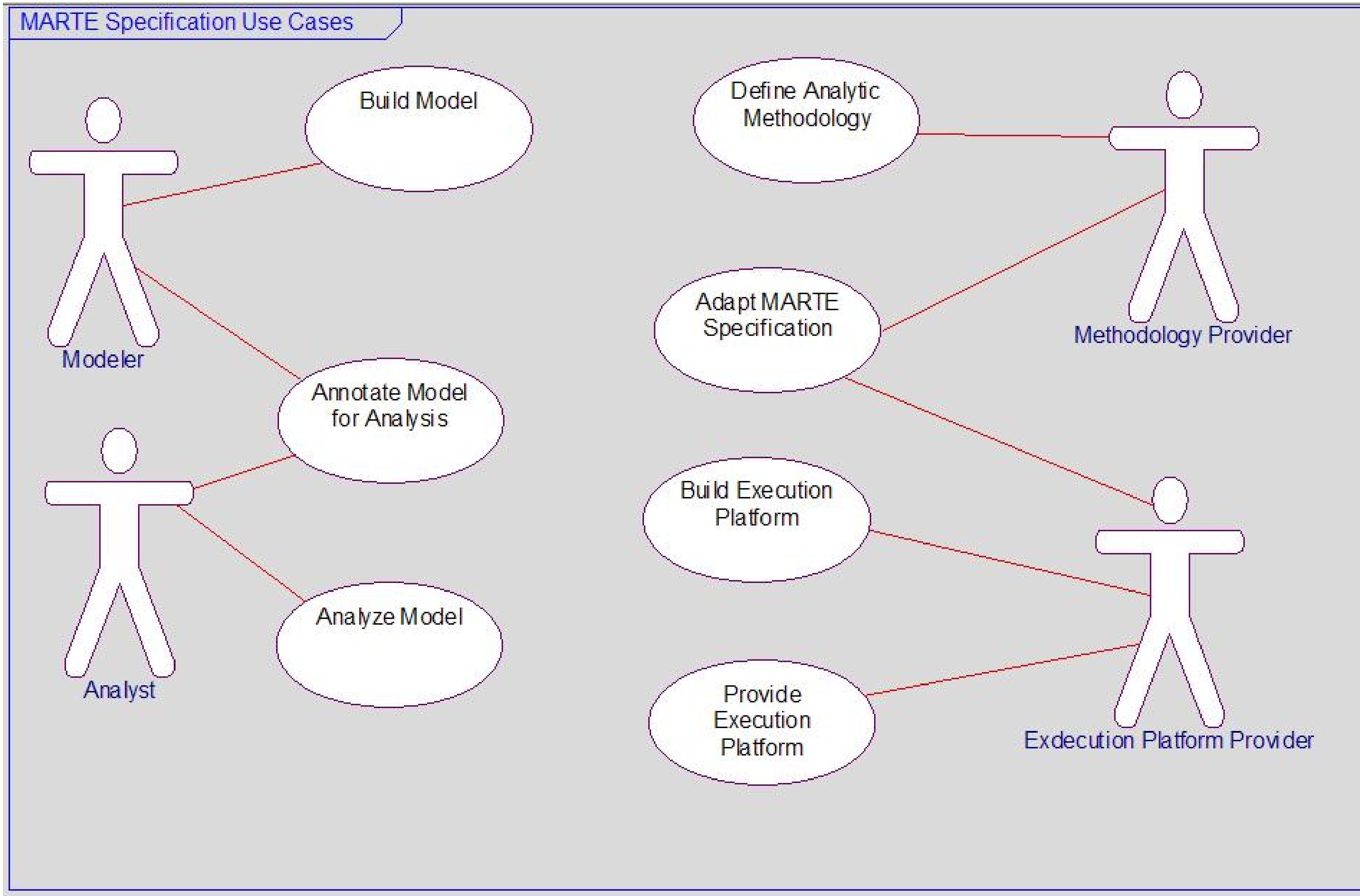
GIGO

- Select the appropriate stereotypes and tags of the schedulability model to match the kind of analysis desired
  - Global RMA
    - Elements: active objects, resources
    - Tags: execution time, deadline, period, priority, blocking time, priority ceiling
  - Detailed RMA
    - Elements: active objects, resources, actions, events, scenarios, scenario steps, messages
    - Tags: execution time, deadline, period, priority, blocking time, priority ceiling
  - Simulation
    - Depends on particular approach
  - etc

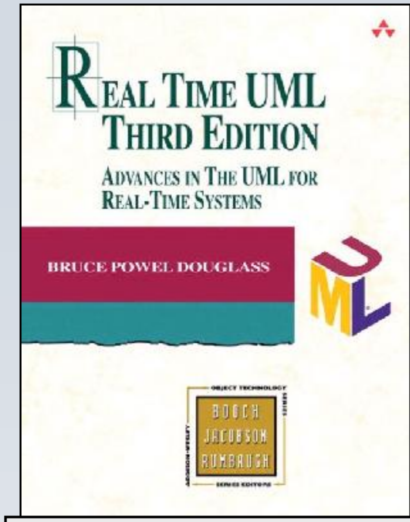
MARTE

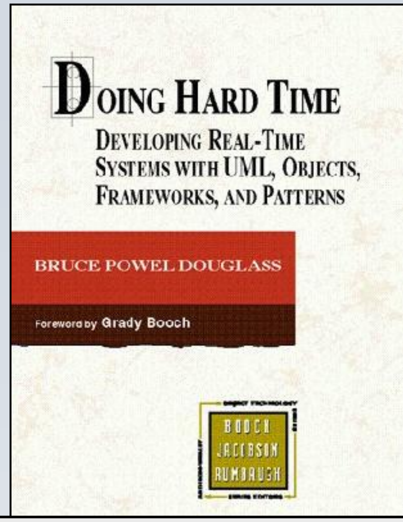
- UML Profile for Modeling and Analysis of Real-Time and Embedded Systems
  - Current status: Approved but not released
  - Latest version: 2009-11-02.pdf spec (OMG Document formal/2009-11-02)
  - Replaces SPT Profile for UML 2
  - Information available at www.OMGmarte.org

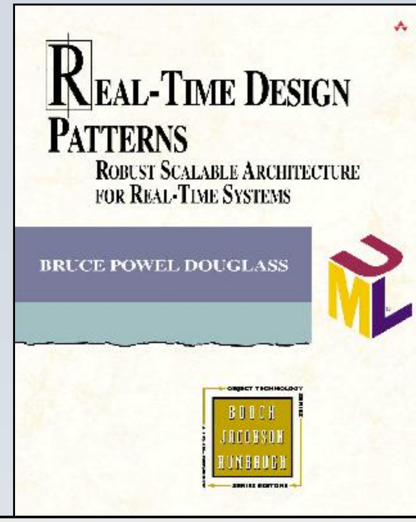


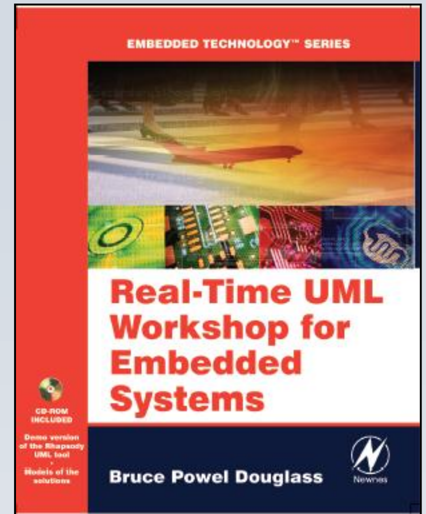

MARTE Specification Use Cases
Build Model
Modeler
Annotate Model for Analysis
Analyst
Analyze Model
Define Analytic Methodology
Adapt MARTE Specification
Methodology Provider
Build Execution Platform
Provide Execution Platform
Exdecution Platform Provider

References









See also http://www-01.ibm.com/software/rational/leadership/thought/brucedouglass.html