Performance-Oriented UML

Bran Selic
Principal Engineer
Rational Software Canada
bselic@rational.com

Tutorial Objectives

- To identify the requirements for enabling performance analysis of UML models
- To introduce the purpose, structure, and key elements of the recently adopted UML Profile for Schedulability, Performance, and Time ("Real-time UML standard")
  - Focus on performance modeling capabilities
- To describe engineering development processes suitable for performance modeling based on the real-time UML standard

Tutorial Structure

1. Models, software models, and software physics
2. A review and critique of UML
3. An introduction to the standard real-time UML profile
4. Modeling resources and quality of service
5. Modeling time in UML
6. Performance analysis with UML
7. Performance (engineering) oriented development
8. Summary and conclusions

Models

- Model (Merriam-Webster):
  - a miniature representation of something
  - *a pattern of something to be made*
  - a type or design of product (as a car)
  - *a description or analogy used to help visualize something (as an atom) that cannot be directly observed*
  - *a system of postulates, data, and inferences presented as a mathematical description of an entity or state of affairs*

Why Do We Construct Models?

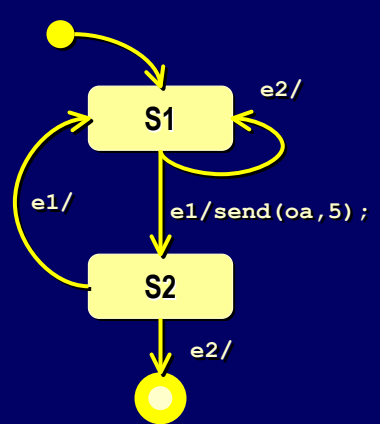
- To learn about the interesting properties of a system
  1. Without having to construct the actual system
     — because it may be too expensive
  2. Without having to experiment with the actual system
     — because it may not be available for experimentation
  3. Without having to view the entire system
     — because the whole system may be too complex to understand

Example of a Software Model

- An abstract description of a program using domain-specific concepts

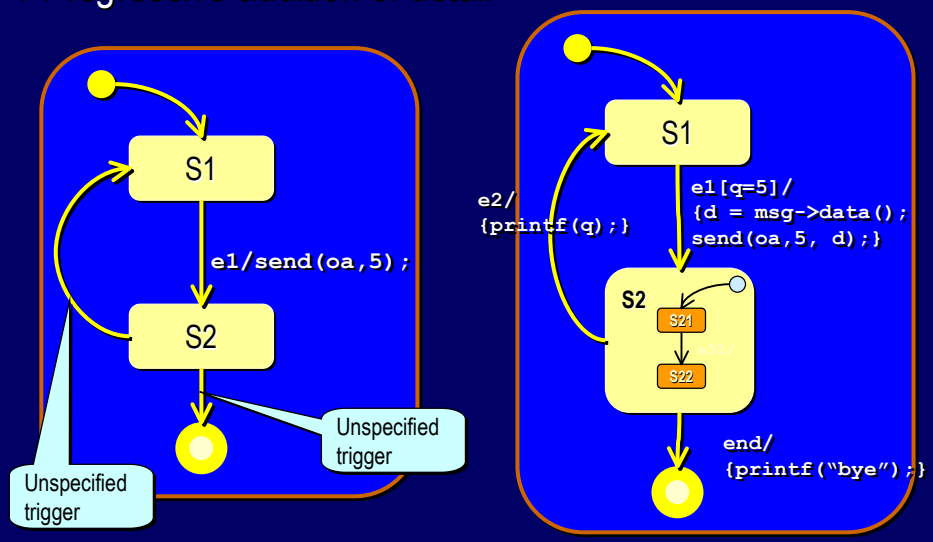
```
case mainState of
   initial: send("I am here");
            end
   S1:      case event of
              e1: send(oa,5);
                  next(S2);
                  end
              e2: next(S1);
                  end

            end
   S2:      case event of
              e1: next(S1);
                  end
              e2: terminate;
                  end
            end
   end
```



Evolution of Software Models

- Progressive addition of detail



The Remarkable Thing About Software Models

Software is the only engineering medium that allows us to directly evolve our models into implementations!

What is Software Made of?

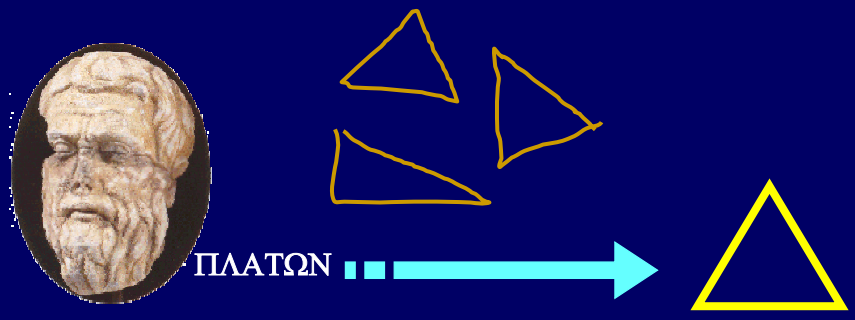
"Because [programs] are put together in the context of a set of information requirements, they observe no natural limits other than those imposed by those requirements. Unlike the world of engineering, there are no immutable laws to violate."

- Wei-Lung Wang
Comm. of the ACM (45, 5)
May 2002

"All machinery is derived from nature, and is founded on the teaching and instruction of the revolution of the firmament."

- Vitruvius
On Architecture, Book X
1st Century BC

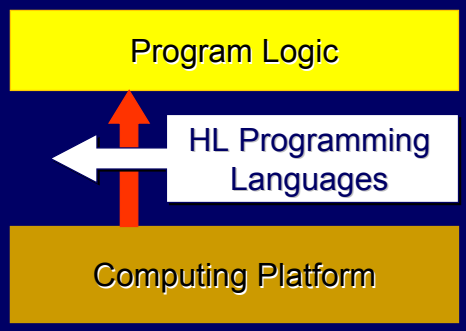
The Stuff of Software: Ideal or Real?



- Platonic view: by focussing on the imperfect world of physical reality we may miss the essence
- Software seems much closer to the realm of ideas than the physical world

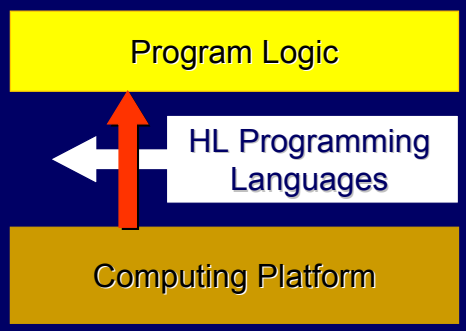
The Principle of Platform Independence

- *Design program logic to be independent of the implementation platform -- as much as possible*
  - separation of concerns
  - software portability



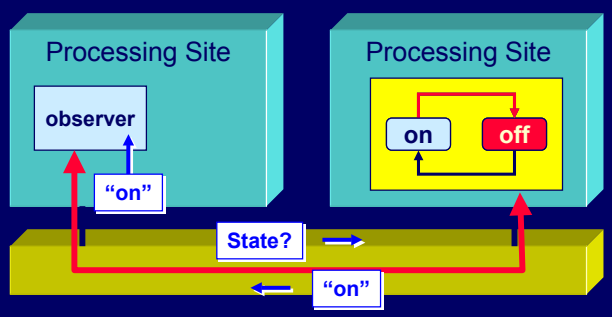
The Case of Real-Time Software

- Key question: Is the response fast enough?
- The quantitative characteristics of the computing environment encroach upon the purity of the logic
  - Real-time software design involves engineering tradeoffs



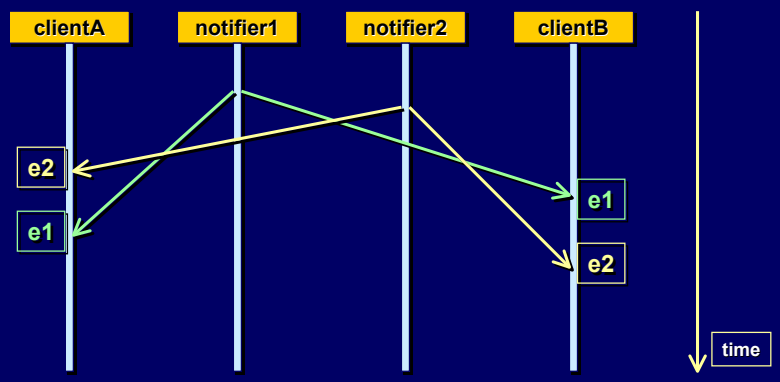
The Case of Distributed Systems

- Possibility of out of date status information due to transmission delays



More on Transmission Delays

- Inconsistent views of system state:
  - different observers see different event orderings



(Physical) quantity changes the (logical) quality

Impossibility Result

It is not possible to guarantee that agreement can be reached in finite time over an asynchronous communication medium, if the medium is lossy or one of the distributed sites can fail

- Fischer, M., N. Lynch, and M. Paterson, "Impossibility of Distributed Consensus with One Faulty Process" *Journal of the ACM*, (32, 2) April 1985.

What Happened?

- The physical world is insinuating itself into the "logic" of the application
- Performance sensitive systems = systems in which the reaction time of software is significant
  - Practically any software that interacts with the physical world
- Conclusion:
  - In performance sensitive systems, such as real-time programs, it is generally not possible to separate out the pure application logic from so-called "platform" issues
- What can we do to salvage the benefits of "platform independence" (portability, separation of concerns) for the case of real-time software?

Tutorial Structure

1. Models, software models, and software physics
2. A review and critique of UML
3. An introduction to the standard real-time UML profile
4. Modeling resources and quality of service
5. Modeling time in UML
6. Performance analysis with UML
7. Performance (engineering) oriented development
8. Summary and conclusions

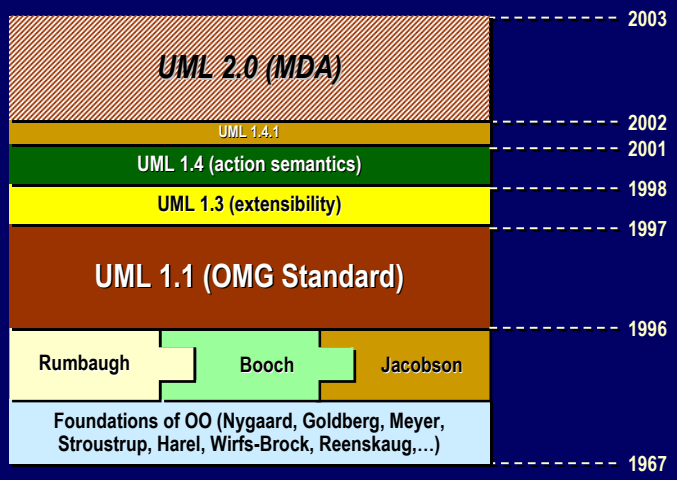
The OMG and the Unified Modeling Language

- Object Management Group (OMG):
  - An industrial consortium of software vendors and users
  - Provides common "technology recommendations" (standards) for the software industry
  - CORBA: a standard for distributed software applications
- The Unified Modeling Language (UML):
  - object-oriented modeling language for modeling software-based systems
  - Adopted extensively by practitioners and researchers
  - Basis for OMG's Model Driven Architecture (MDA) initiative

The Model Driven Architecture (MDA) Initiative

- Based on the success of UML, the OMG has formulated a vision of a method of software development based on the use of models
- Key characteristic of MDA:
  - The focus and principal products of software development are models rather than programs
  - "The design is the implementation" (i.e. UML as both a modeling and an implementation language)
- UML plays a crucial role in MDA
  - Automatic code generation from UML models
  - Executable UML models
  - Requires a more precise definition of the semantics of UML (UML 2.0)

The Evolution of UML



Components of UML

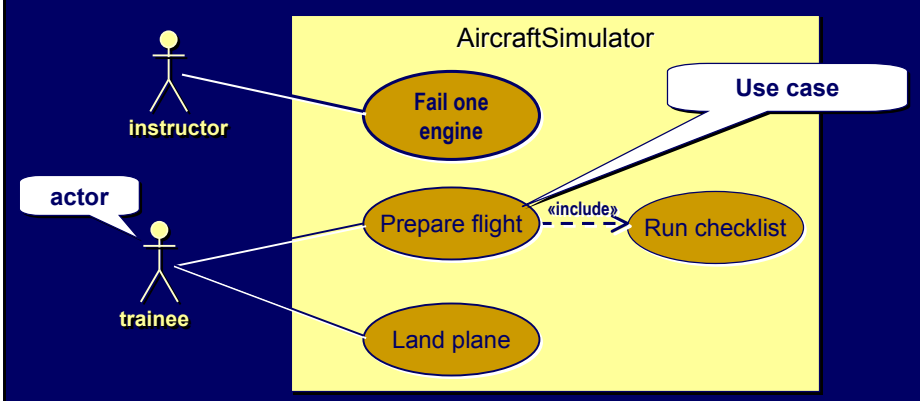
- Basic set of (extensible) modeling concepts
  - used for modeling both problems and solutions (object, class, association)
  - deep semantic roots
- Formal rules of semantically meaningful composition (well-formedness)
- Graphical notation for modeling concepts
  - 8 different diagram types (requirements, structure, behavior, deployment)

UML Model Views

- Requirements (use case diagrams)
- Static structure (class diagrams)
  - kinds of objects and their relationships
- Object behavior (state machines)
  - possible life histories of an object
- Inter-object behavior (activity, sequence, and collaboration diagrams)
  - flow of control among objects to achieve system-level behavior
- Physical implementation structures (component and deployment diagrams)
  - software modules and deployment on physical nodes

Use Case Diagrams

- Used to capture functional requirements
  - useful as principal drivers of the overall development process

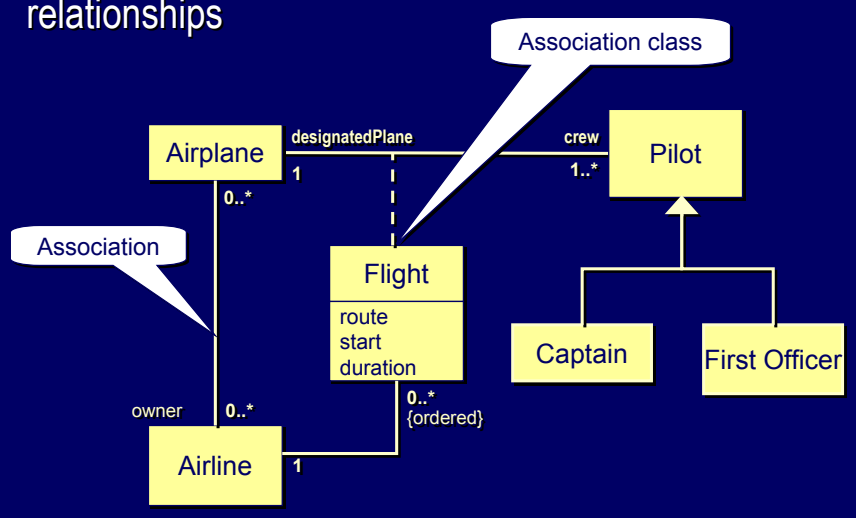


Use Cases and Performance Concerns

- As useful as in any other domain
  - fundamental drivers of system definition, development, and testing
- However….
  - Focus on function (functional requirements)
  - In RT systems, much focus on non-functional requirements
    - e.g., end-to-end delays, maximum response times,…
  - Even though UML is a standard, there is no standard way of associating such non-functional requirements with use cases

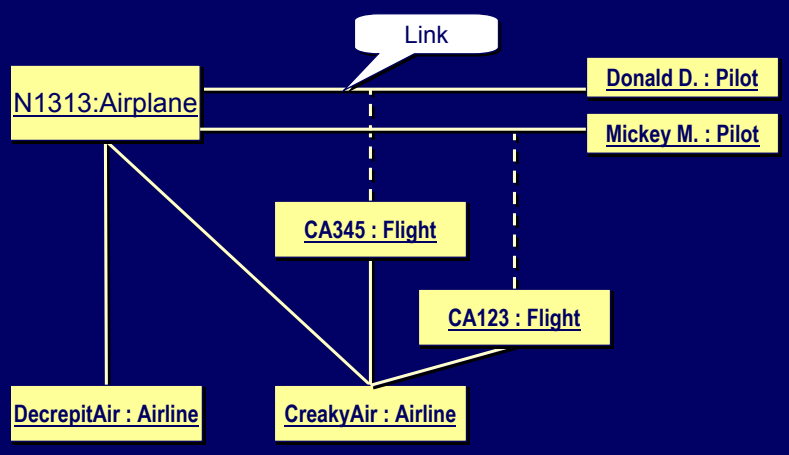
Class Diagram

- Shows the kinds of entities in a system and their general relationships



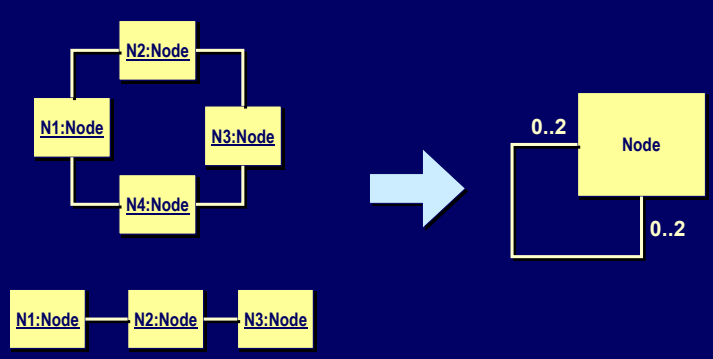
Object Instance Diagram

- Shows object instances in a particular case of a general class diagram



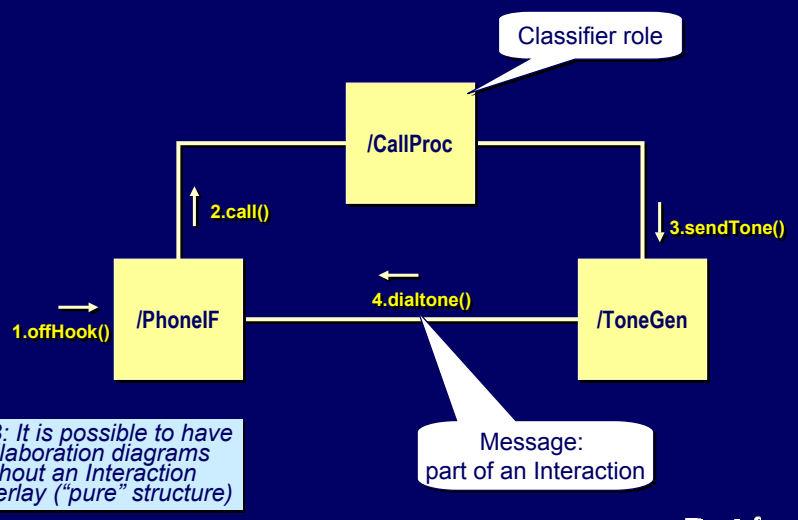
Class Diagrams and Performance Concerns

- Because they abstract out certain specifics, class diagrams are not suitable for performance analysis



Collaboration Diagram

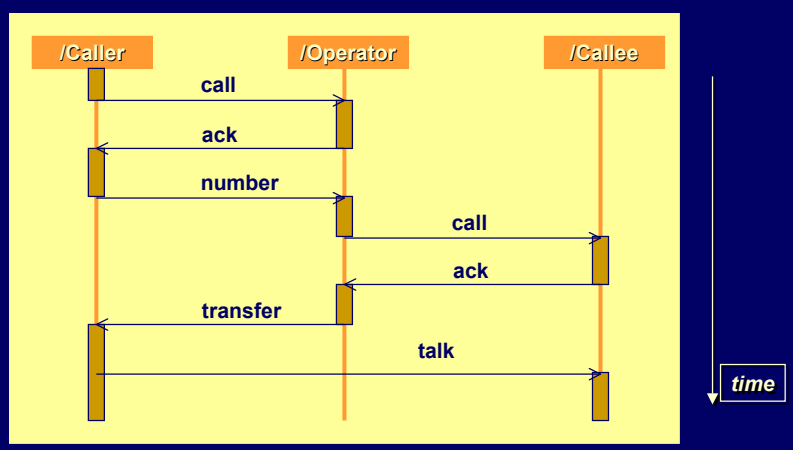
- Depict generic structural and behavioral patterns



NB: It is possible to have collaboration diagrams without an Interaction overlay ("pure" structure)

Sequence Diagrams

- Show interactions between objects with a focus on communications (a different representation of a collaboration)

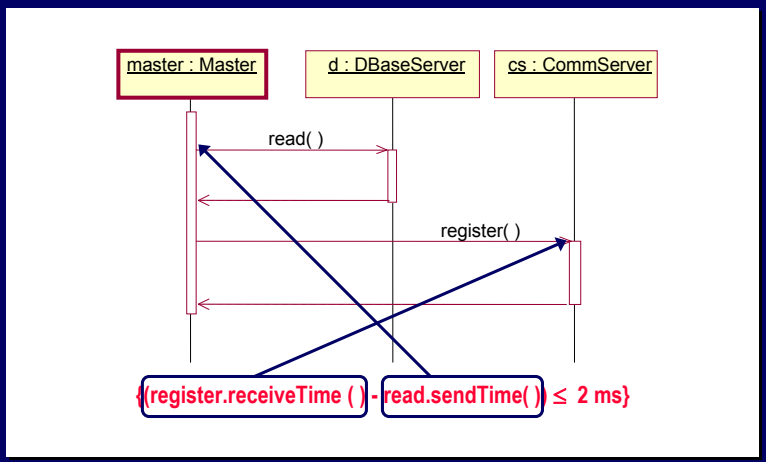


Sequence Diagrams and Performance Concerns

- Sequence diagrams are extremely useful for showing object interactions
  - very common, particularly in real-time systems
  - well suited for event-driven behavior
  - in telecom, many protocol standards are defined using sequence diagrams
- However…
  - No standard way of denoting performance requirements
  - Very limited capabilities for specifying timing constraints and characteristics
  - Do not scale up to complex systems

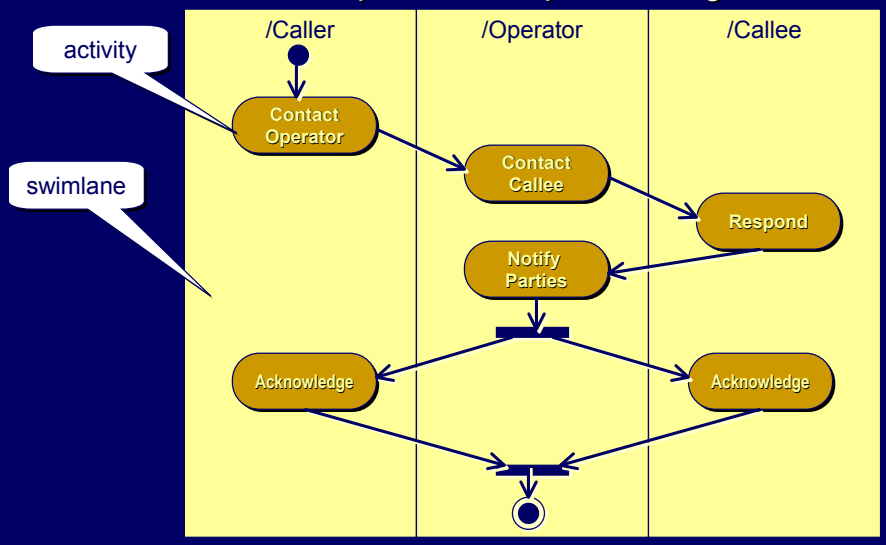
Using Timing Marks with Sequence Diagrams

- Specifying constraints



Activity Diagrams

- Different focus compared to sequence diagrams

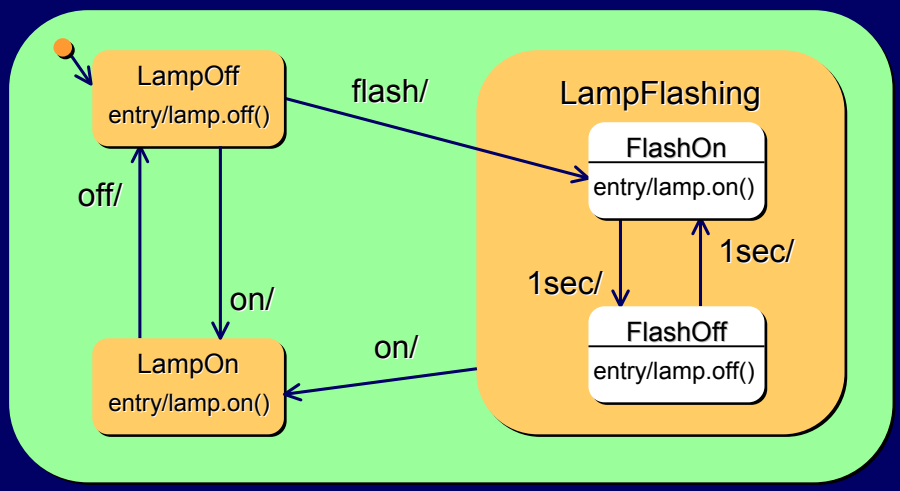


Activity Diagrams and Performance Concerns

- Better than sequence diagrams for
  - showing concurrency (forks and joins are explicit)
  - scaling up to complex systems
- However…
  - No standard way of denoting performance requirements and timing characteristics
  - Less well-suited for describing event-driven behavior

UML Statechart Diagrams

- Very suitable for specifying complex event-driven behavior

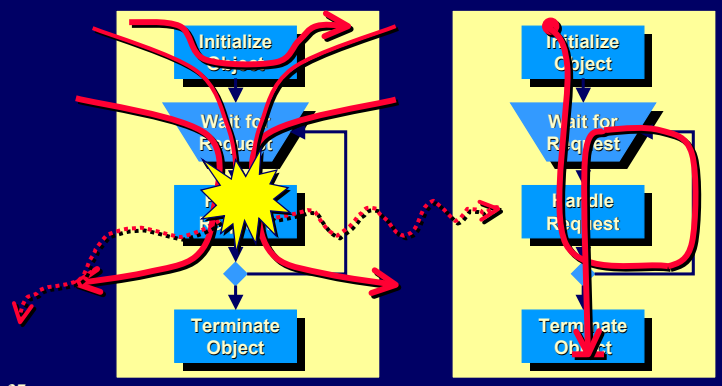


State Machines and Performance Concerns

- Many software systems are event-driven
  - very well suited to those systems
  - Hierarchical capabilities scale up very nicely
- However…
  - not directly connected to time (except for time events)

Objects and Concurrency

- *Passive objects:* have no control of their communications
  - Clients determine when to invoke an operation
- *Active objects:* can control when to respond to requests
  - Can avoid concurrency conflicts
  - Require at least one independent engineering-level thread



UML Concurrency Model

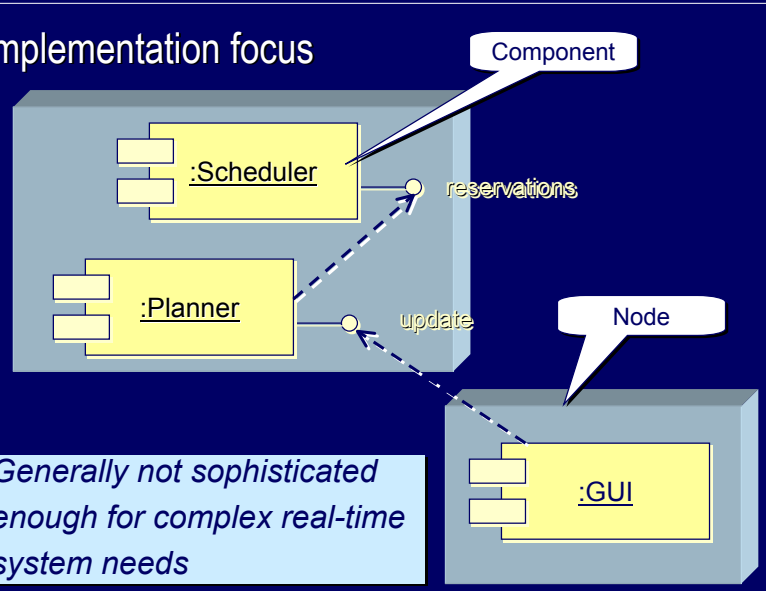
- Active objects are the major concurrency mechanism of UML
  - automatically resolve certain classes of concurrency conflicts
- However...
  - Not clear how this model maps to concurrency mechanisms that are used in the real-time domain (processes, threads, semaphores, real-time scheduling methods, etc.)?

Scheduling in UML

- The actual scheduling policy is unspecified
  - A *semantic variation point*
  - Can be customized to suit application requirements

Component and Deployment Diagrams

- Implementation focus



Generally not sophisticated enough for complex real-time system needs

Implementation Diagrams and Performance Concerns

- Probably the weakest part of UML
- Not sophisticated enough to capture the various complex aspects of deployment common to many performance sensitive systems
  - deferred mapping of software to hardware
  - mapping of software to software
- No standard way to describe the quantitative requirements/characteristics of hardware and software (e.g., scheduling discipline, CPU speed, transmission delays)

Summary: UML and Performance Sensitive Systems

- Using UML for performance sensitive systems automatically brings the benefits of the object paradigm
  - structural focus, inheritance, strong encapsulation, polymorphism,…
- However, there are many open questions
  - best ways of using UML
  - missing or non-standard concepts
  - ability to create predictive models

Tutorial Structure

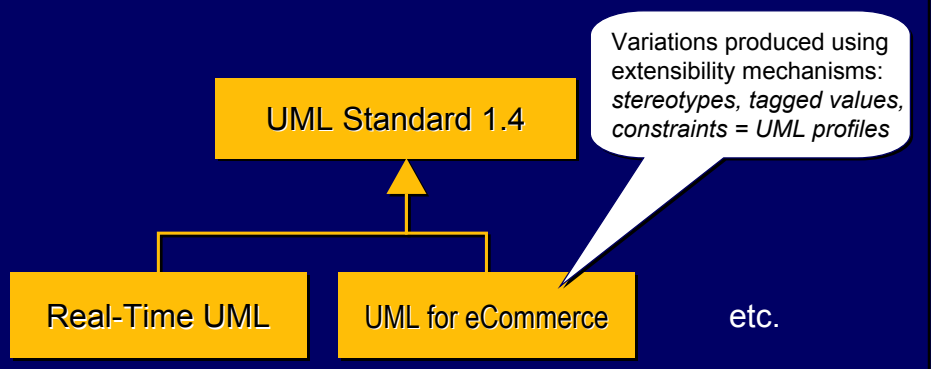
1. Models, software models, and software physics
2. A review and critique of UML
3. An introduction to the standard real-time UML profile
4. Modeling resources and quality of service
5. Modeling time in UML
6. Performance analysis with UML
7. Performance (engineering) oriented development
8. Summary and conclusions

Semantic Variation in UML

- Semantic aspects that are:
  - undefined (e.g., scheduling discipline), or
  - intentionally ambiguous (multiple, mutually-exclusive, interpretations)
- Why?
  - Different domains require different specializations
  - The applicability and usefulness of UML would have been severely constrained if it could not support such diversity
- The scope and semantic impact of semantic variation choices must be strictly limited

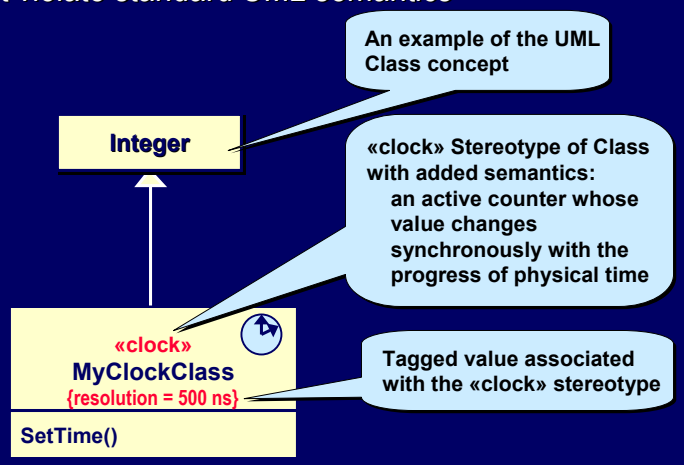
UML is a Family of Languages

- Standard UML is like an abstract class that can be refined into multiple more concrete forms



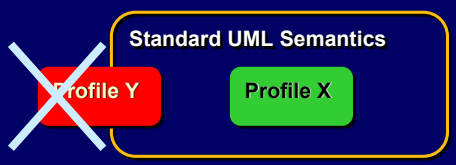
Specializing UML: Stereotypes

- We can add semantics to any standard UML concept
  - Must not violate standard UML semantics



UML Profiles

- A package of related specializations of general UML concepts that capture domain-specific variations and usage patterns
  - *A domain-specific interpretation of UML*
- Fully conformant with the UML standard
  - additional semantic constraints cannot contradict the general UML semantics
  - within the "semantic envelope" defined by the standard



Requirements for a Real-Time UML

- *"UML profile for scheduling performance and time"*
  - Adopted as an official OMG standard in March 2002 (ptc/2002-03-02)
- Defines standard methods for using UML to model:
  - Physical time
  - Timing specifications
  - Timing services and mechanisms
  - Modeling resources (logical and physical)
  - Concurrency and scheduling
  - Software and hardware infrastructure and their mapping
  - ..including specific notations for the above

Important Caveat

- The standard does *not* ask for new concepts or methods
- The intent is to support existing and future modeling techniques and analysis methods in the context of UML
  - ⇒ *response should not be biased towards any particular technique or method*

RT Profile: Guiding Principles

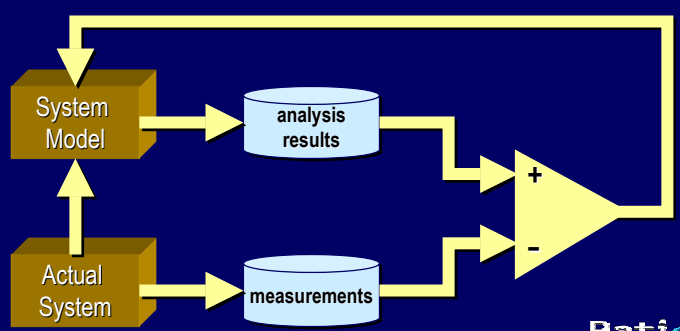
- Ability to specify quantitative information directly in UML models
  - key to quantitative analysis and predictive modeling
- Flexibility:
  - users can model their RT systems using modeling approaches and styles of their own choosing
  - open to existing and new analysis techniques
- Facilitate the use of analysis methods
  - eliminate the need for a deep understanding of analysis methods
  - as much as possible, automate the generation of analysis models and the analysis process itself

Quantitative Methods for RT Systems

- Once we have included QoS information in our models, we can use *quantitative methods* to:
  - predict system characteristics (detect problems early)
  - analyze existing system
  - synthesize elements of the model
- Methods considered for the profile:
  - Schedulability analysis
    will the system meet all of its deadlines?
  - Performance analysis based on queueing theory
    what kind of response will the system have under load?

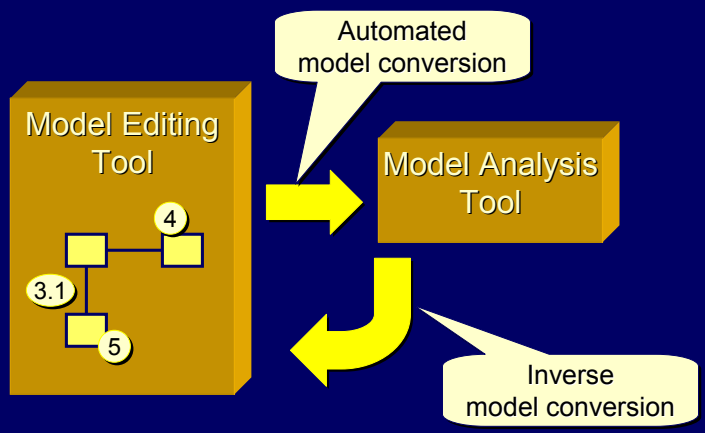
Issues with Quantitative Methods

- Require uncommon and highly-specialized skills
- Software is notoriously difficult to model
  - highly non-linear (detail often matters)
  - models are frequently severely inaccurate and not trustworthy
  - typical modeling process is highly manual:

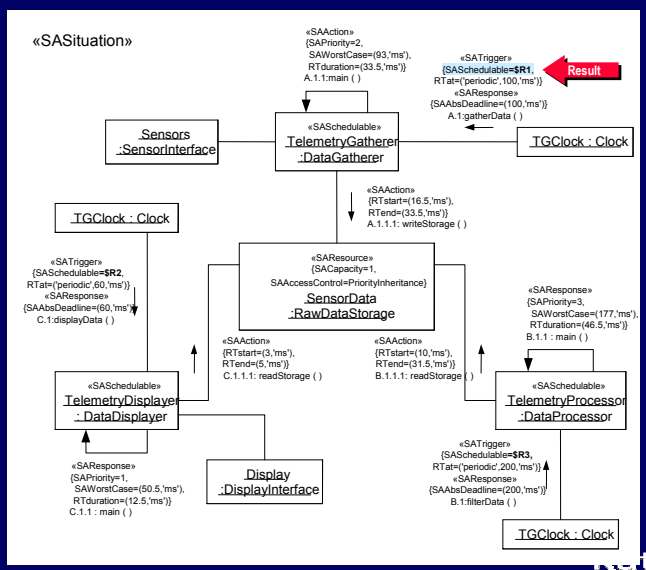


Desired Development Model

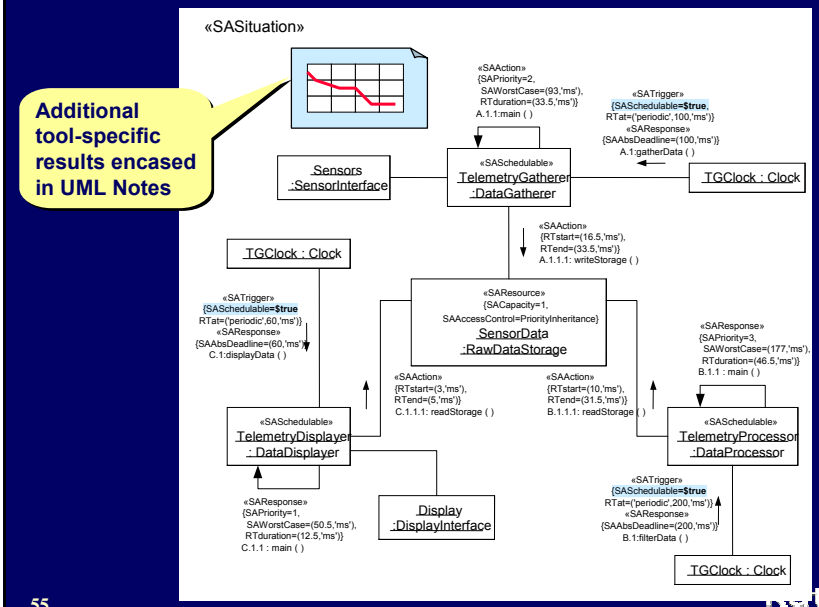
- Seamless integration of technologies and tools based on standards for real-time modeling



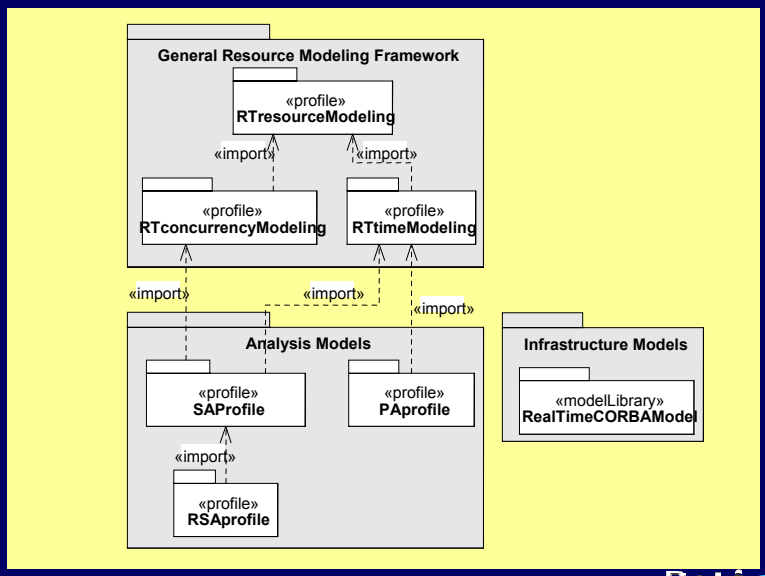
Example: Schedulability Annotations



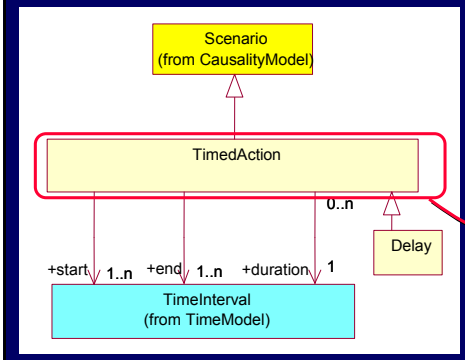
Example: Analysis Results



UML Real-Time Profile Structure



Format: Domain Model and Extensions



Domain model:
Identifies domain concepts and relationships using UML class diagrams

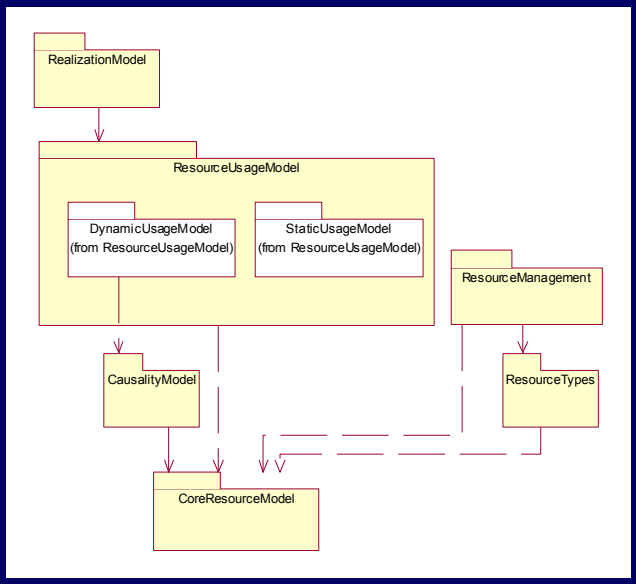
Corresponding UML extensions (tabular form)

Stereotype	Base Class	Tags
«RTaction»	Action	RTstart RTend RTduration
	ActionExecution	
	Message	
	Stimulus	
	Method	
	ActionSequence	
	ActionState	
	SubactivityState	
	Transition	
	State	

Tutorial Structure

1. Models, software models, and software physics
2. A review and critique of UML
3. An introduction to the standard real-time UML profile
4. Modeling resources and quality of service
5. Modeling time in UML
6. Performance analysis with UML
7. Performance (engineering) oriented development
8. Summary and conclusions

The General Resource Model



Quality of Service Concepts

- *Quality of Service (QoS):*

  a specification (usually quantitative) of how well a particular service is (to be) performed

  - e.g. throughput, capacity, response time

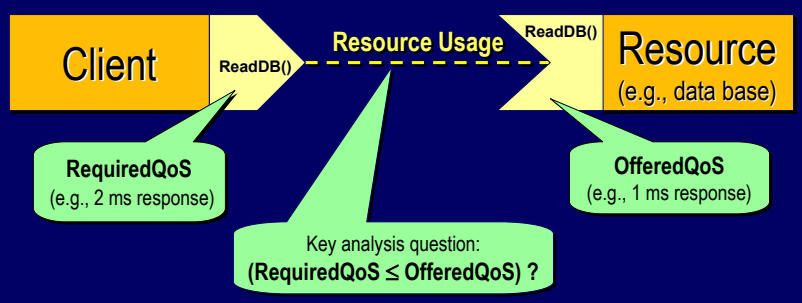
- The specification of a model element can include:
  - *offered QoS:* the QoS that it provides to its clients
  - *required QoS:* the QoS it requires from other components to support its QoS obligations

Resources and Quality of Service

- Resource:

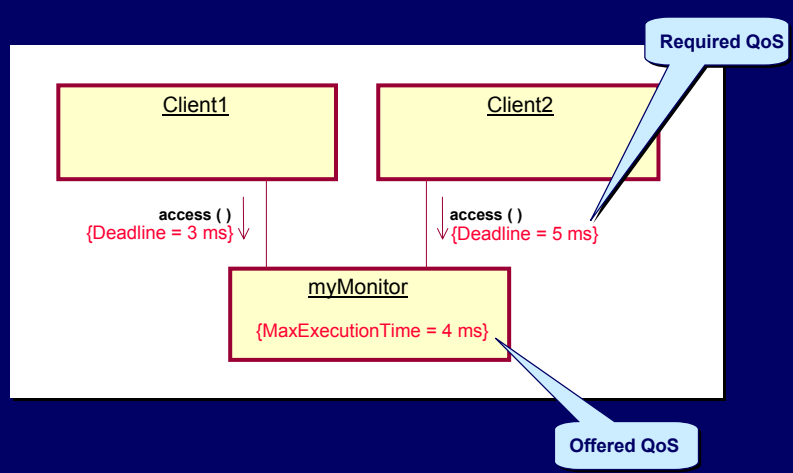
  an element whose service capacity is limited, directly or indirectly, by the finite capacities of the underlying physical elements

- These capacities are expressed as QoS attributes

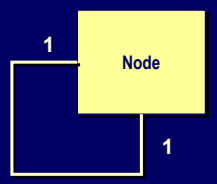


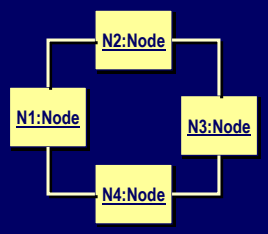
Simple Example

- Concurrent tasks accessing a monitor with known response time characteristics



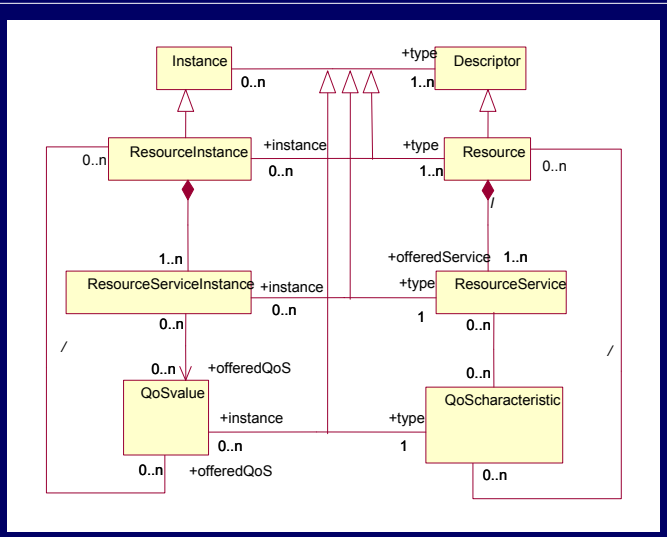
Instance- vs Class-Based Models





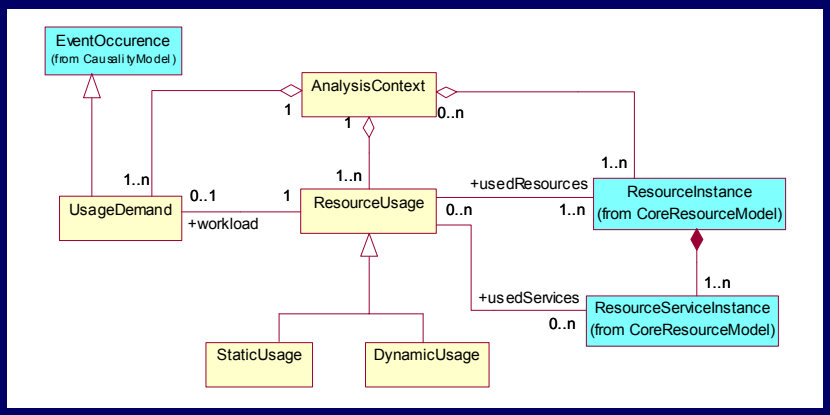
- Practically all analysis methods are concerned with instance-based models
- However, it is often useful to associate QoS characteristics with classes
  - Used to define default values that may be overridden for specific instances
- Need to apply a stereotype to both spec elements and instance elements

Core Resource Model (Domain Model)



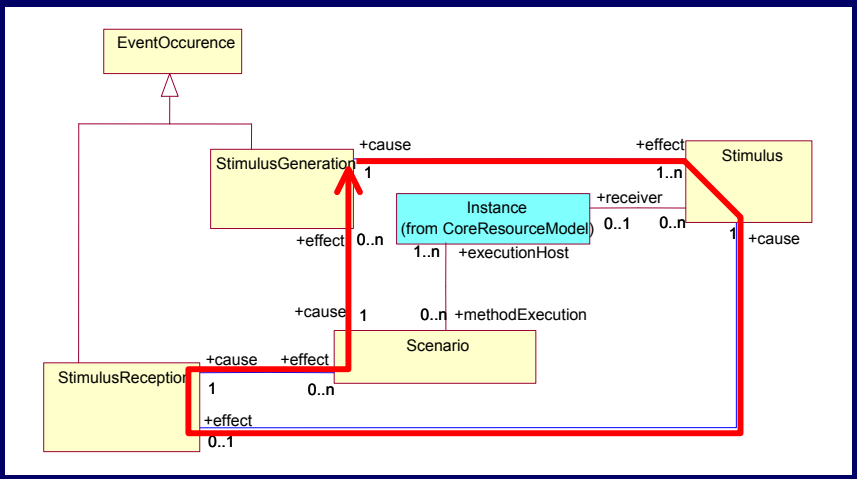
- *NB: This is just a model of the concepts! (domain model)*

Basic Resource Usage Model

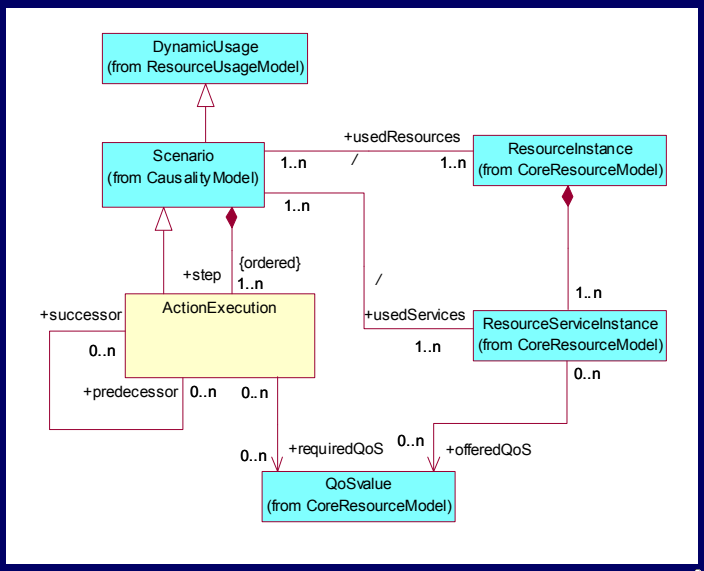


Basic Causality Loop

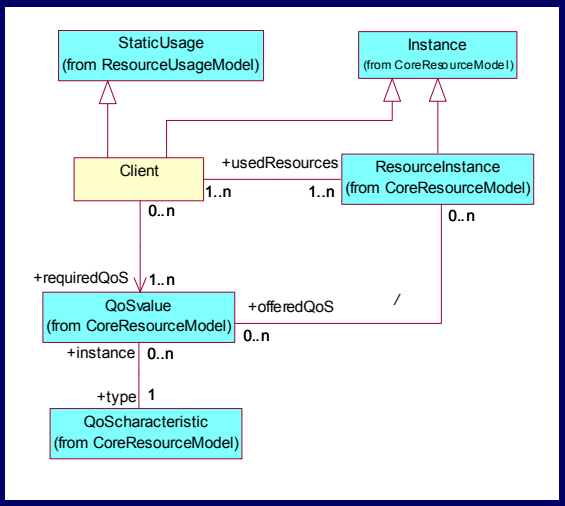
- Used in modeling dynamic scenarios



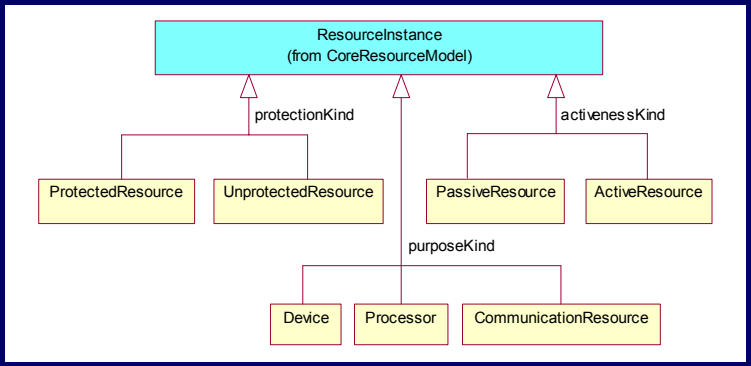
Dynamic Usage Model



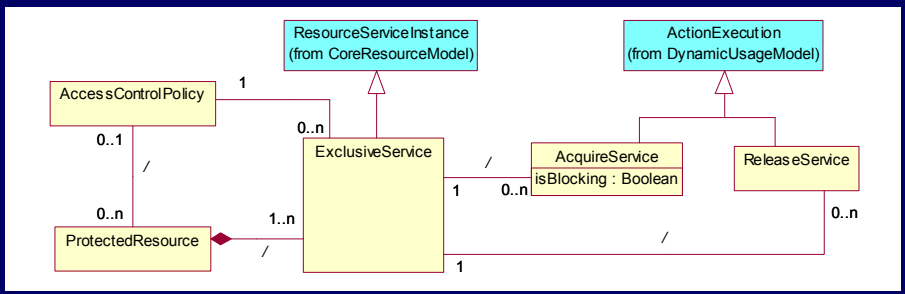
Static Usage Model



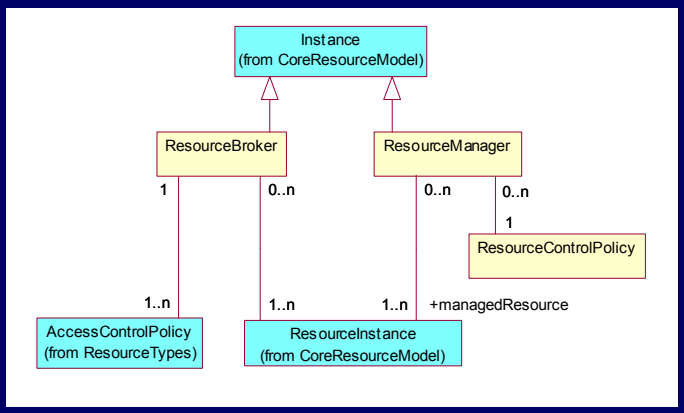
Resource Categorizations



Exclusive Use Resources and Actions

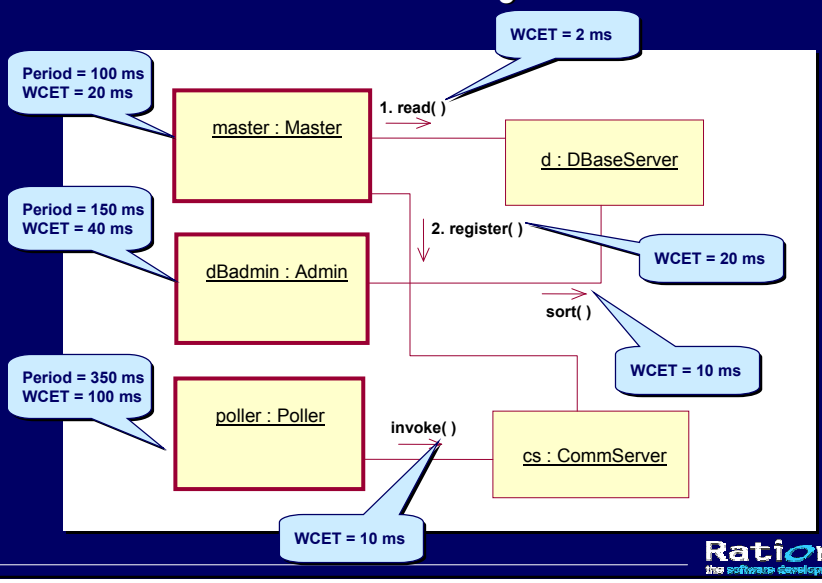


Resource Management Model



Example UML Model

- Periodic concurrent tasks sharing resources



Standard Stereotypes

- To allow an analysis tool to extract the necessary QoS information, we define a set of standard stereotypes and related tags*

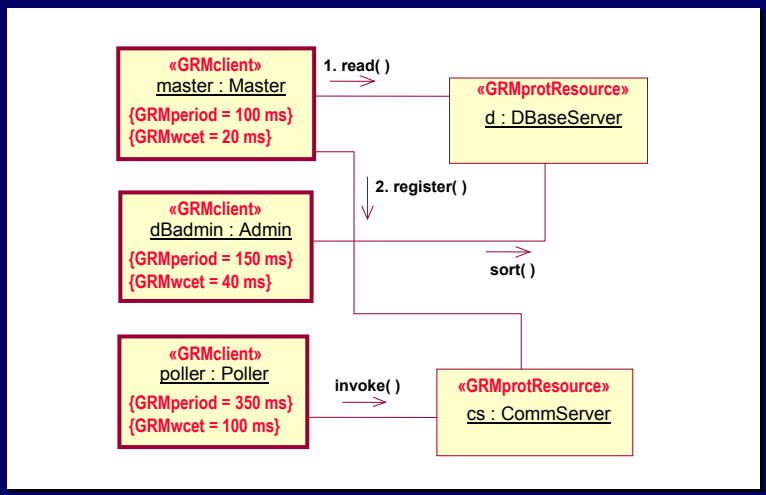
Stereotype	UML base concepts	Tags
GRMclient	Classifier, Instance	GRMperiod, GRMwcet
GRMprotResource	Classifier, Instance	N/A
GRMresService	BehavioralFeature	GRMwcet

Tag	Tag Type
GRMperiod	RTtimeString
GRMwcet	RTtimeString

* For clarity: the stereotypes and tags have been simplified from their actual form

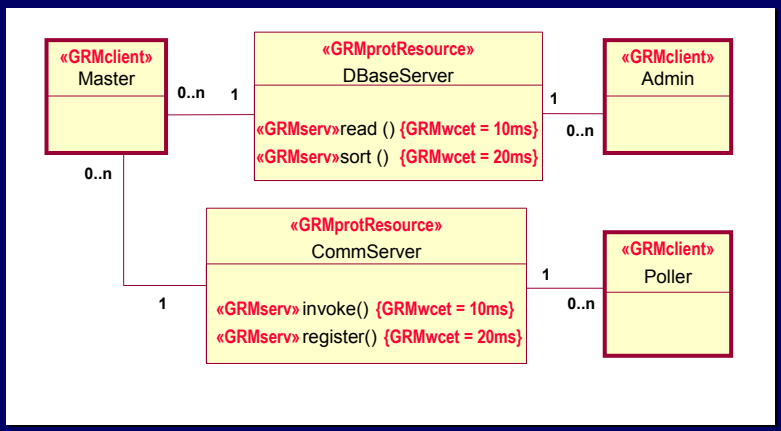
Example: QoS Annotations

- Using the standard stereotypes...



Example: Class Diagram

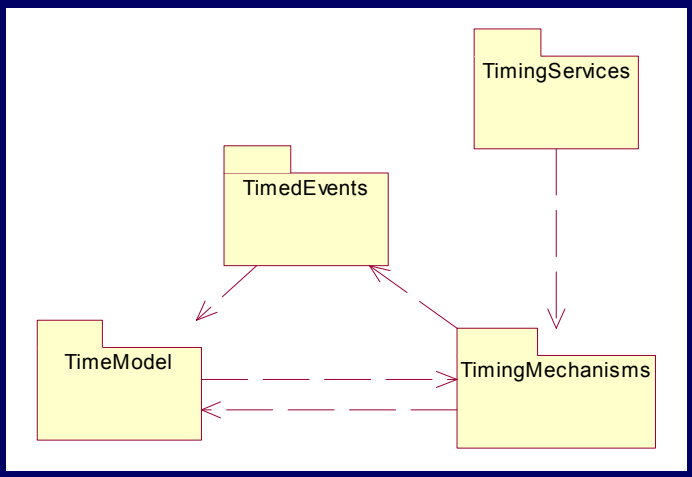
- QoS annotations can be added to classes as well



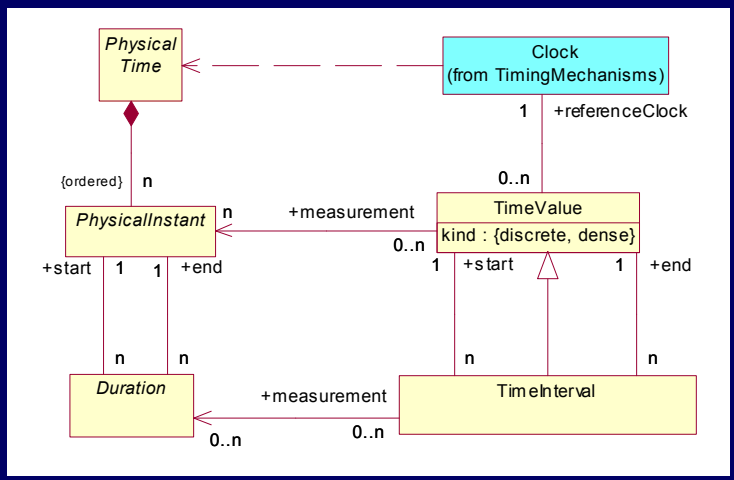
Tutorial Structure

1. Models, software models, and software physics
2. A review and critique of UML
3. An introduction to the standard real-time UML profile
4. Modeling resources and quality of service
5. Modeling time in UML
6. Performance analysis with UML
7. Performance (engineering) oriented development
8. Summary and conclusions

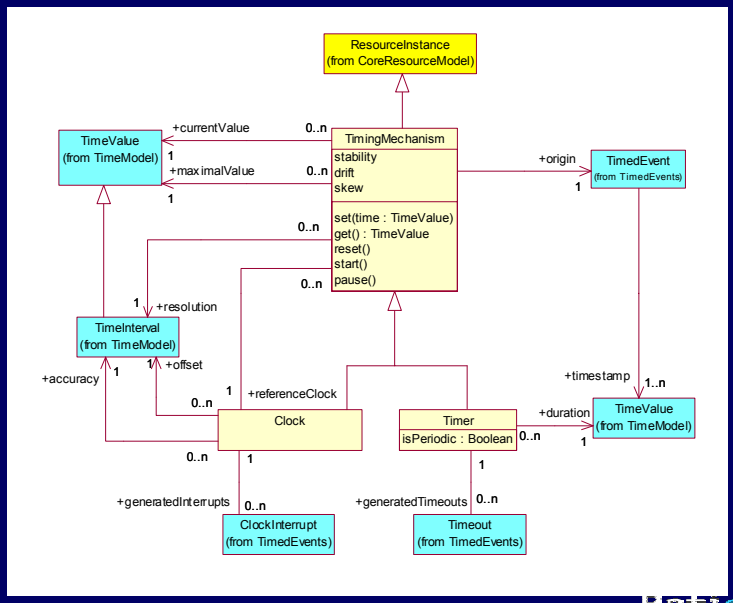
General Time Model



Physical and Measured Time



Timing Mechanisms Model

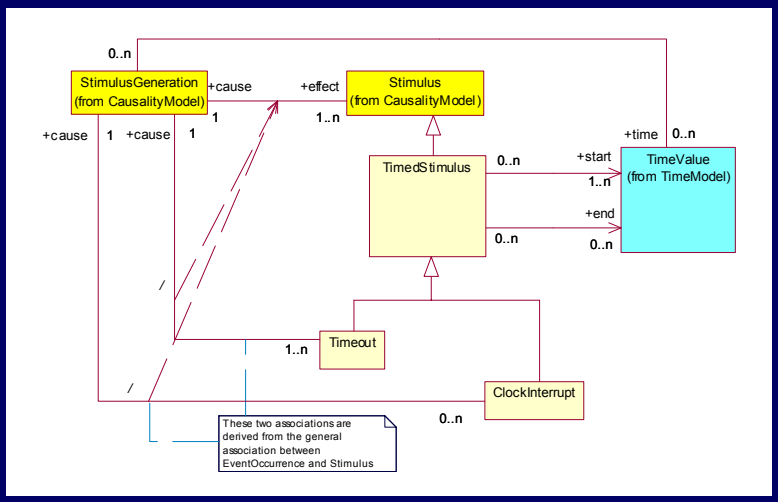


Example Timing Stereotype

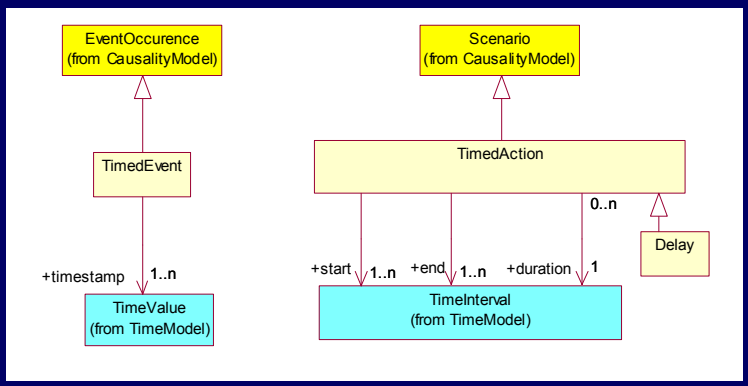
Stereotype	Base Class	Tags
«RTaction»	Action	RTstart RTend RTduration
	ActionExecution	
	Message	
	Stimulus	
	Method	
	ActionSequence	
	ActionState	
	SubactivityState	
	Transition	
	State	

Tag	Tag Type	Multiplicity	Domain Name
RTstart	RTtimeValue	[0..1]	TimedAction::start
RTend	RTtimeValue	[0..1]	TimedAction::end
RTduration	RTtimeValue	[0..1]	TimedAction::duration

Timed Stimuli

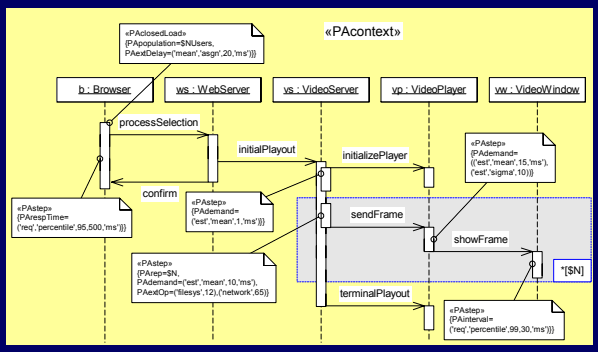


Timed Events and Timed Actions

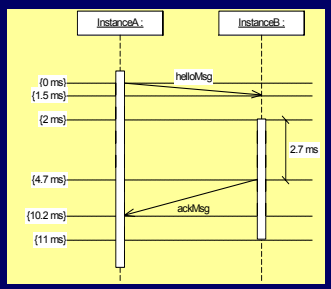


Time Annotations

- In various behavioral diagrams (sequence, activity, state)



More compact forms are also possible:



May be very sophisticated and express complex time values (instants and durations) including probability distributions, percentile values, etc.
(NB: tools can help reduce visual clutter)

Notation: Timing Marks and Constraints

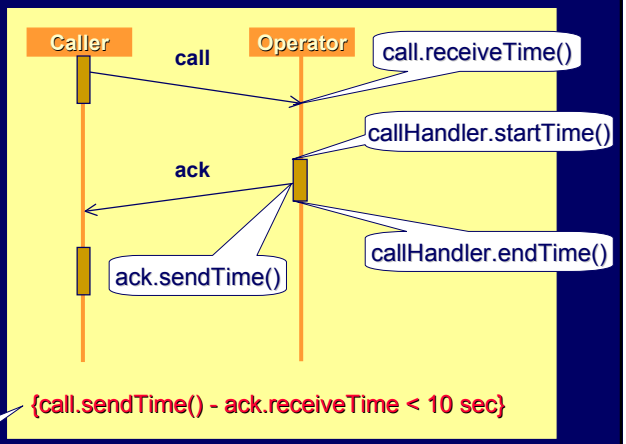
- A *timing mark* identifies the time of an event occurrence

- On messages:

  sendTime()
  receiveTime()

- On action blocks (new):

  startTime()
  endTime()



Timing constraint

Defined Stereotypes (1 of 3)

Stereotype	Applies To	Tags	Description
«RTaction»	Action, ActionExecution, Stimulus, Action, Message, Method…	RTstart [0..1] RTend [0..1] RTduration [0..1]	An action that takes time
«RTclkInterrupt» (subclass of «RTstimulus»)	Stimulus, Message	RTtimestamp [0..1]	A clock interrupt
«RTclock» (subclass of «RTtimingMechanism»)	Instance, DataType, Classifier, ClassifierRole…	RTclockId [0..1]	A clock mechanism
«RTdelay»	Action, ActionExecution, Stimulus, Action, Message, Method…	RTduration [0..1]	A pure delay activity
«RTevent»	Action, ActionExecution, Stimulus, Action, Message, Method…	RTat [0..1]	An event that occurs at a known time instant
«RTinterval»	Instance, Object, Classifier, DataType, DataValue	RTintStart [0..1] RTintEnd [0..1] RTintDuration [0..1]	A time interval

Defined Stereotypes (2 of 3)

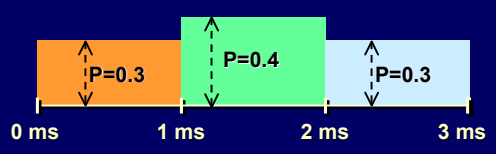
Stereotype	Applies To	Tags	Description
«RTnewClock	Operation	RTstart [0..1] RTend [0..1] RTduration [0..1]	An operation that creates a new clock mechanism
«RTnewTimer»	Operation	RTtimerPar [0..1]	An operation that creates a new timer
«RTpause»	Operation		A pause operation on a timing mechanism
«RTreset»	Operation		An operation that resets a timing mechanism
«RTset»	Operation	RTtimePar [0..1]	An operation that sets the current value of a timing mechanism
«RTstart»	Operation		An operation that starts a timing mechanism
«RTstimulus»	Stimulus, ActionExecution, Action, ActionSequence, Method	RTstart [0..1] RTend [0..1]	A timed stimulus

Defined Stereotypes (3 of 3)

Stereotype	Applies To	Tags	Description
«RTtime»	DataValue, Instance, Object, DataType, Classifier	RTkind [0..1] RTrefClk [0..1]	A time value or a time object
«RTtimeout» (subclass of «RTstimulus»)	Stimulus, ActionExecution, Action, ActionSequence, Method	RTtimestamp [0..1]	A timeout signal or a timeout action
«RTtimer» (subclass of «RTtimingMechanism»)	DataValue, Instance, Object, ClassifierRole, Classifier…	RTduration [0..1] RTperiodic [0..1]	A timer mechanism
«RTtimeService»	Instance, Object, ClassifierRole, Classifier		A time service
«RTtimingMechanism»	DataValue, Instance, Object,ClassifierRole, Classifier, DataType	RTstability [0..1] RTdrift [0..1] RTskew [0..1] RTmaxValue [0..1] RTorigin [0..1] RTresolution [0..1] RTaccuracy [0..1] RTcurrentVal [0..1] RToffset [0..1] RTrefClk [0..1]	A timing mechanism

Specifying Time Values

- Time values can be represented by a special stereotype of Value («RTtimeValue») in different formats; e.g.
  - 12:04 (time of day)
  - 5.3, 'ms' (time interval)
  - 2000/10/27 (date)
  - Wed (day of week)
  - $param, 'ms' (parameterized value)
  - 'poisson', 5.4, 'sec' (time value with a Poisson distribution)
  - 'histogram' 0, 0.3 1, 0.4 2, 0.3, 3, 'ms'



Specifying Arrival Patterns

- Method for specifying standard arrival pattern values
  - Bounded: *'bounded', <min-interval>, <max-interval>*
  - Bursty: *'bursty', <burst-interval> <max.no.events>*
  - Irregular: *'irregular', <interarrival-time>, [<interarrival-time>]**
  - Periodic: *'periodic', <period> [, <max-deviation>]*
  - Unbounded: *'unbounded', <probability-distribution>*
- Probability distributions supported:
  - Bernoulli, Binomial, Exponential, Gamma, Geometric, Histogram, Normal, Poisson, Uniform

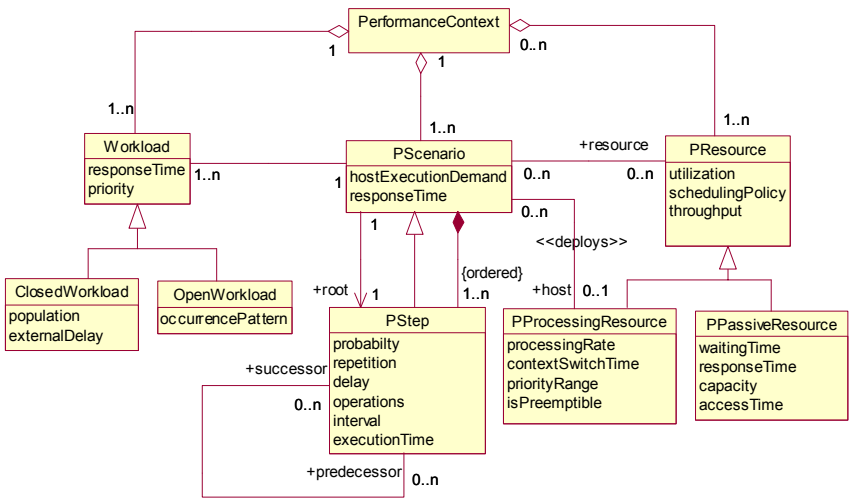
Tutorial Structure

1. Models, software models, and software physics
2. A review and critique of UML
3. An introduction to the standard real-time UML profile
4. Modeling resources and quality of service
5. Modeling time in UML
6. Performance analysis with UML
7. Performance (engineering) oriented development
8. Summary and conclusions

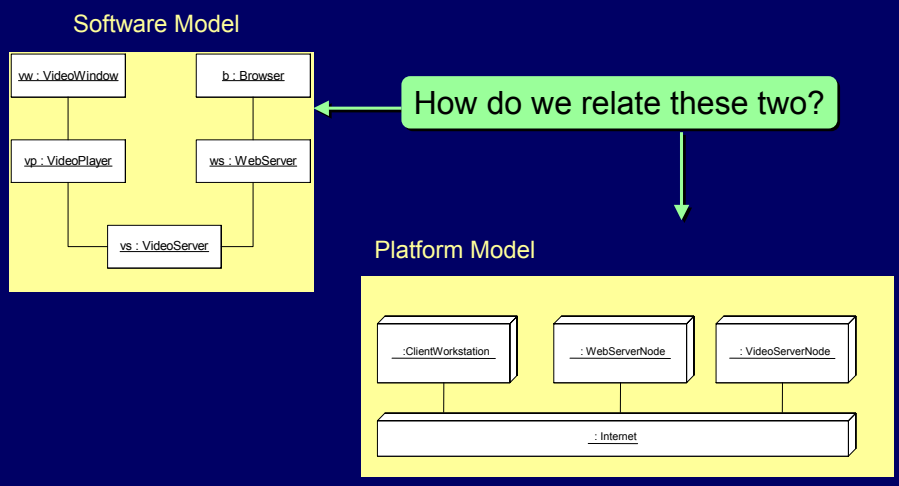
"Knowledge of what is possible is the beginning of happiness"

- George Santayana

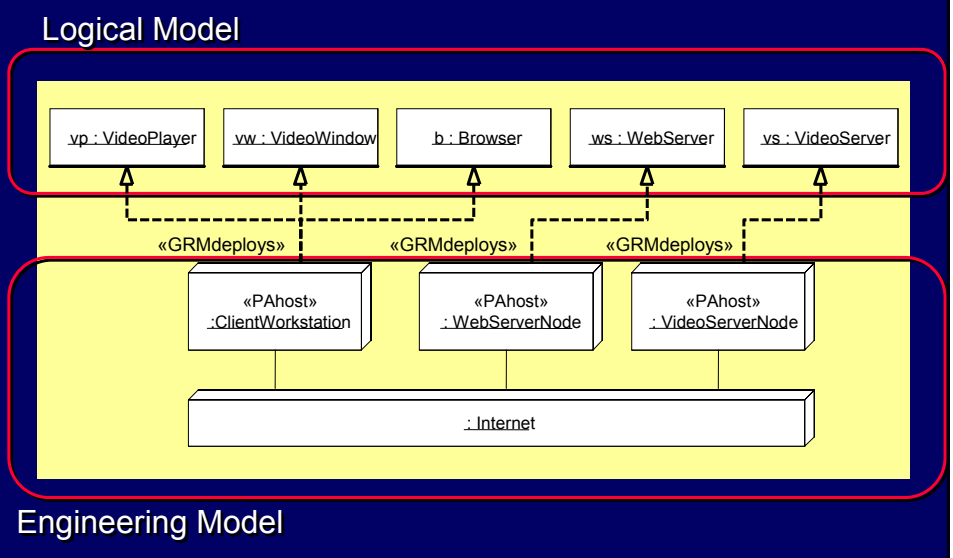
Performance Analysis Concepts



Software and Platform Models

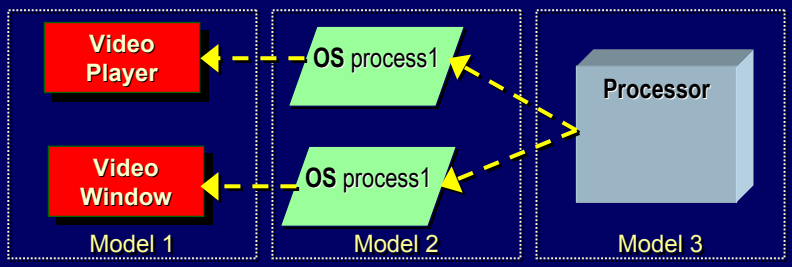


Realization Mappings



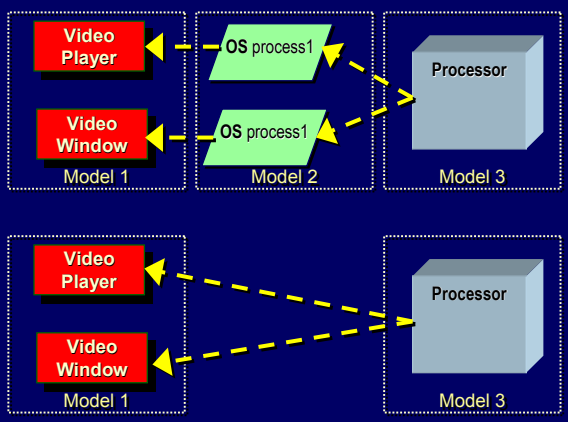
Realization Mappings

- A correspondence between elements of two models representing different viewpoints of a system (logical and engineering)
- Semantics: the logical elements are *implemented* by the corresponding engineering model elements
  - logical elements can be viewed as being deployed on the corresponding engineering elements



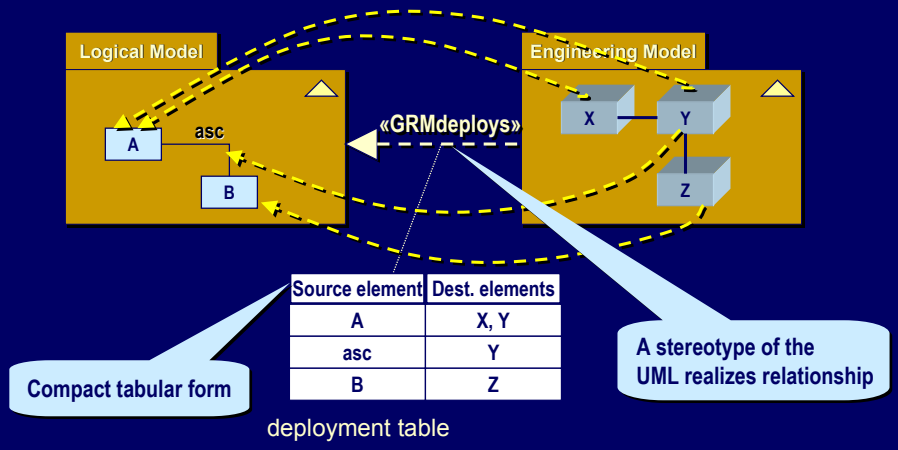
Choice of Level of Abstraction

- Intermediate levels may be abstracted out
  - depends on the desired granularity of modeling
  - affects the semantics of the realization relationship



Modeling Realization in UML

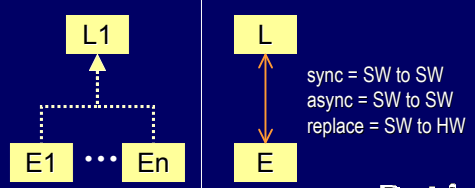
- An association between models with explicit realization mappings between model elements



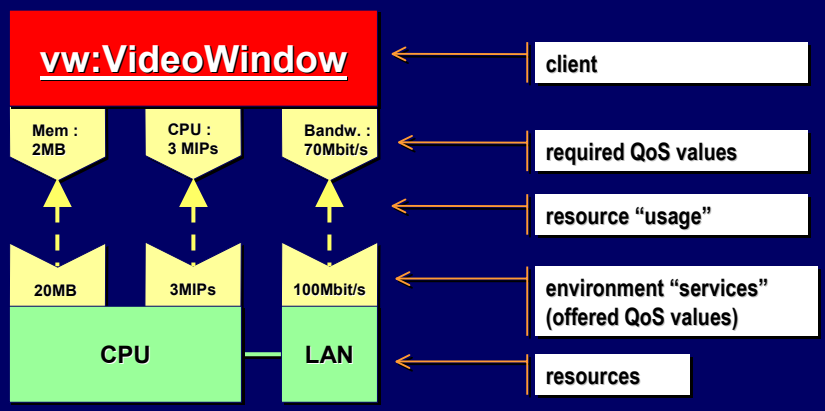
RT UML: Specifying Realization Mappings

- Captures the specifics of the realization mapping
  - Either as a string (tag value) or as a table

Logical Elements	Engineering Elements	Mode	Linkage	Additional Constraints
<List of logical model elements>	*<List of corresponding engineering model elements>*	*<If there are multiple engineering elements, one of:>* {inclusive, exclusiveStatic, exclusiveDynamic}	*<Interaction mode between levels, one of:>* {sync, async, replace}	*<Any additional constraints that apply to the mapping>*



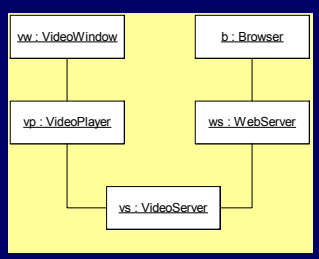
The Resource Model and Realization



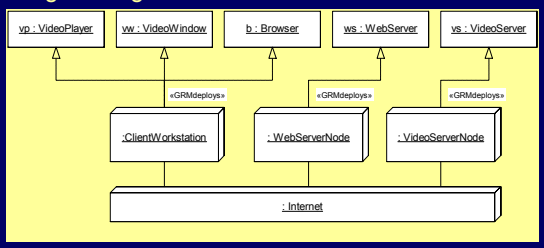
⇒*The same QoS framework used in the general resource model of the RT UML standard can be used for modeling realization*

Example: Web Video Application

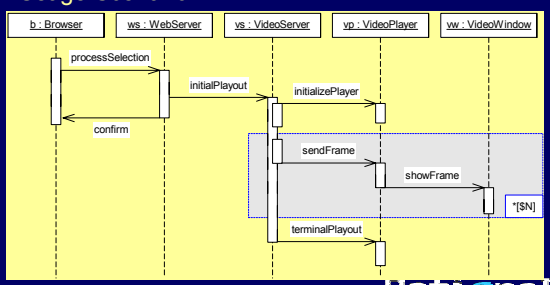
Logical Instance Model



Engineering Instance Model



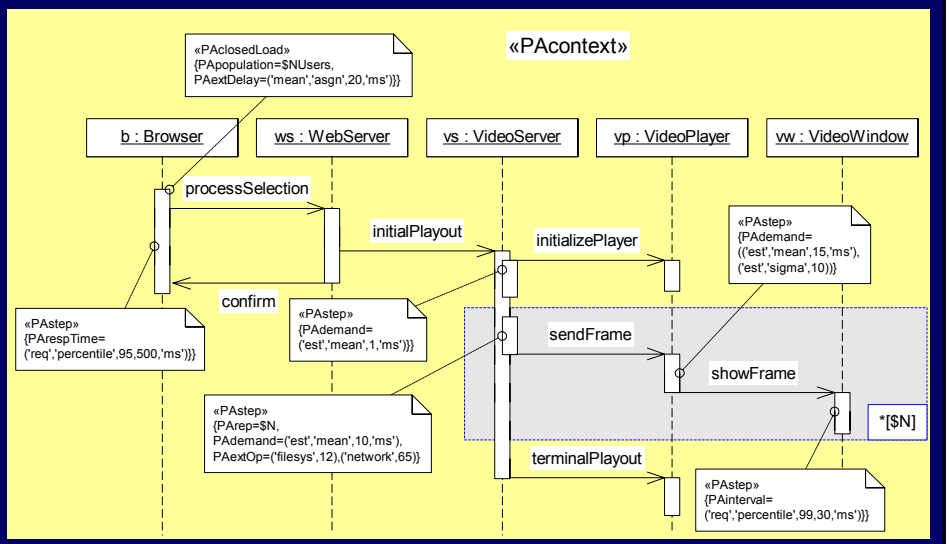
Usage Scenario



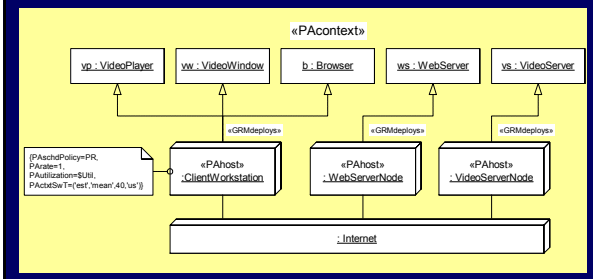
Example: Performance Requirements

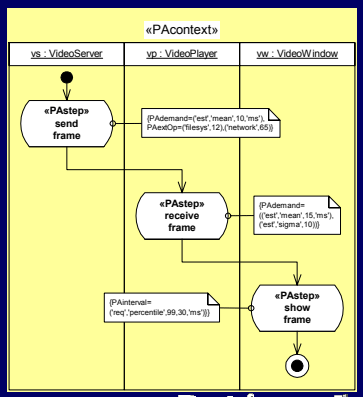
- Estimated video server processing demand per frame = 10 ms
- Estimated viewer processing demand per frame = 15 ms (dev = 20 ms)
- Assumed network delay distribution: exponential with mean = 10 ms
- Measured packets per frame (LAN) = 65
- Measured video server file operations per frame = 12
- Max. number of concurrent users = $Nusers
- Average inter-session times = 20 minutes
- Frames in a video $N
- Video frame intervval = 30 ms
- Required confirmation delay: 95% < 500 ms
- Required interval between frame displays = 99% < 30 ms

Example: Annotations for a Scenario



Example: More Annotations





Defined Stereotypes (1 of 2)

Stereotype	Applies To	Tags	Description
«PAclosedLoad»	Action, ActionExecution, Stimulus, Action, Message, Method…	PArespTime [0..*] PApriority [0..1] PApopulation [0..1] PAextDelay [0..1]	A closed workload
«PAcontext»	Collaboration, CollaborationInstanceSet, ActivityGraph		A performance analysis context
«PAhost»	Classifier, Node, ClassifierRole, Instance, Partition	PAutilization [0..*] PAschdPolicy [0..1] PArate [0..1] PActxtSwT [0..1] PAprioRange [0..1] PApreemptible [0..1] PAthroughput [0..1]	A deferred receive
«PAopenLoad»	Action, ActionExecution, Stimulus, Action, Message, Method…	PArespTime [0..*] PApriority [0..1] PAoccurrence [0..1]	An open workload

Defined Stereotypes (2 of 2)

Stereotype	Applies To	Tags	Description
«PAresource»	Classifier, Node, ClassifierRole, Instance, Partition	PAutilization [0..*] PAschdPolicy [0..1] PAcapacity [0..1] PAmaxTime [0..1] PArespTime [0..1] PAwaitTime [0..1] PAthroughput [0..1]	A passive resource
«PAstep»	Message, ActionState, Stimulus, SubactivityState	PAdemand [0..1] PArespTime [0..1] PAprob [0..1] PArep [0..1] PAdelay [0..1] PAextOp [0..1] PAinterval [0..1]	A step in a scenario

Specifying Performance Values

- A complex structured string with the following format
  - <kind-of-value> , <modifier> , <time-value>
- Where:
  - <kind-of-value> ::= ‘req’ | ‘assm’ | ‘pred’ | ‘msr’
  - Required, assumed, predicted, measured
  - <modifier> ::= ‘mean’ | ‘sigma’ | ‘kth-mom’ , <Integer> | ‘max’ | ‘percentile’ <Real> | ‘dist’
  - E.g.:

    {PAdemand = (‘msr’, ‘mean’, (20, ‘ms’))}

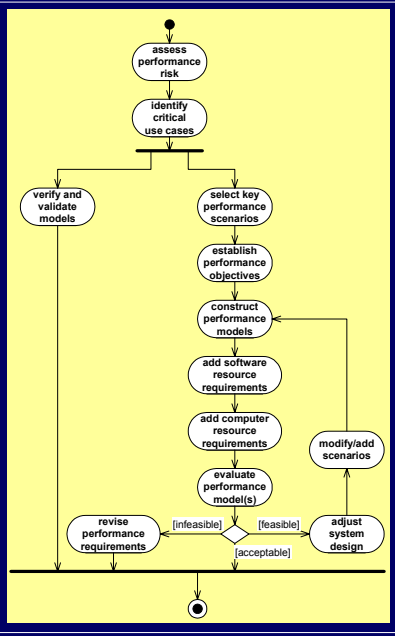
Tutorial Structure

1. Models, software models, and software physics
2. A review and critique of UML
3. An introduction to the standard real-time UML profile
4. Modeling resources and quality of service
5. Modeling time in UML
6. Performance analysis with UML
7. Performance (engineering) oriented development
8. Summary and conclusions

Software Performance Engineering (SPE)

- Modern software development process/method based on UML (but can be generalized)
- "A systematic, quantitative approach to constructing software systems that meet performance objectives"
- Developed by Connie Smith and Lloyd Williams
  - Recently published book by Addison Wesley

The SPE Process



SPE Activities (1 of 4)

- *Assess Performance Risk:* determine the likelihood that performance will be a problem (experience, models, etc.)
- *Identify Critical Use Cases:* use cases that are:
  - important to the user(s) of the system
  - most likely to be performance problems
- *Select Key Performance Scenarios:* subset of scenarios corresponding to critical use cases that are most likely to be performance problems
  - Define the scenarios (using sequence diagrams, activity diagrams, actions)

SPE Activities (2 of 4)

- *Establish Performance Objectives:* for each key performance scenario
  - Establish what is acceptable for the scenario (based on requirements)
  - Example objectives: desired response time, throughput, resource usage
  - Specify workload intensity (e.g., arrival rate) for each scenario based on experience, computation, or measurement
- *Construct Performance Models:* this may be automatically done with the right tools directly from suitably annotated UML models

SPE Activities (3 of 4)

- *Determine Software Resource Requirements:* amount of processing and software resources required for each scenario step
  - From a software perspective
  - Determine best and worst case estimates to determine range (potentially, on a choice of different platforms)
- *Add Computer Resource Requirements:* load imposed on devices used by scenario steps
  - From a hardware platform perspective
  - E.g., number of disk I/O operations; number of CPU instructions required for a step
  - Depends on the target environment

SPE Activities (4 of 4)

- *Evaluate Performance Models:* using the model and selected analysis method, compute the performance predictions
  - May be automated if suitable tools are provided
  - Feasible: choose alternative, more promising design approach
  - Infeasible: change product requirements
- *Verify and Validate the Model:* continuously evaluate the models to determine their accuracy
  - As our knowledge of the system increases

Real-World Real-Time Design Dilemma

If the physical world intrudes on the logical world, how can we separate the "logical" world of design from the "physical" world of implementation to achieve the benefits of platform independence (software portability, separation of concerns)?

Engineering-Oriented Design (EOD)

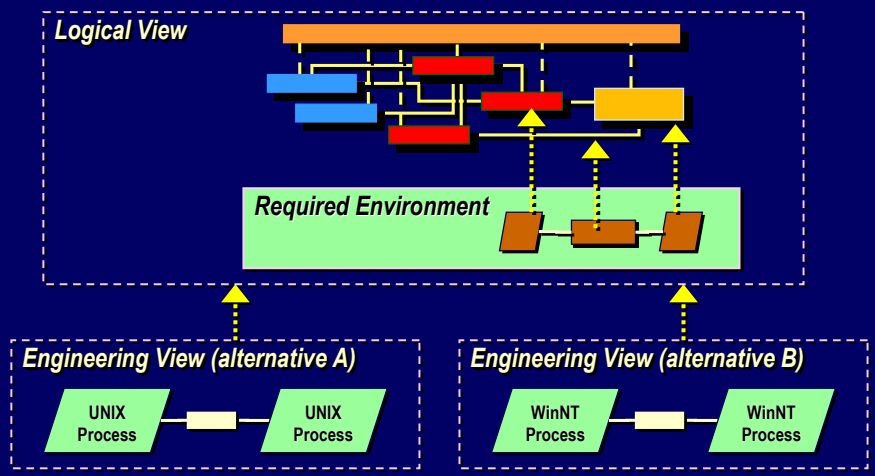
- *Analysis and design of software systems based on use of*
  - *Models*
  - *QoS specifications (accounting for physical properties)*
  - *Quantitative analysis techniques and simulation*
- Complements any model-based development method
- Advantages:
  - Higher reliability (simplification due to use of models)
  - Ability to predict system characteristics (and major design flaws) prior to full realization
  - Portability to different hardware configurations and technologies!

Achieving Platform Independence with EOD

- Approach: *Define an abstract technology-independent quantified specification of the required characteristics of the engineering model as part of the logical model*
  - i.e., design software for a specific *QoS environment*, rather than for a specific platform

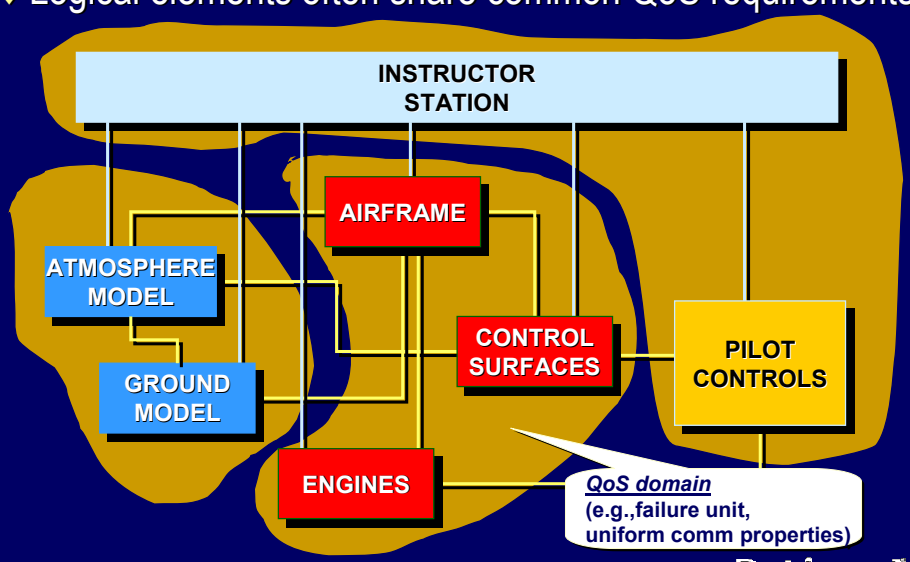
Viewpoint Separation

- *Required Environment:* a platform independent QoS environment specification required by the logical elements of a model



Required Environment Partitions

- Logical elements often share common QoS requirements

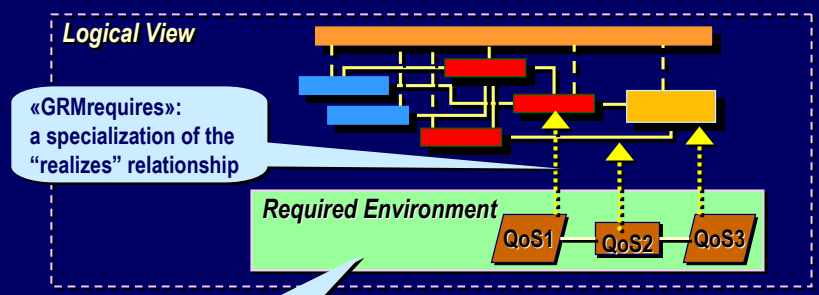


QoS Domains

- Specify a domain in which certain QoS values apply universally:
  - failure characteristics (failure modes, availability, reliability)
  - CPU performance
  - communications characteristics (delay, throughput, capacity)
  - etc.
- During realization, the QoS values of a domain can be compared against those of a concrete platform (engineering environment) to see if a given environment can support that model
- In essence, through the use of QoS domains, we are:
  - *Specifying explicitly the kind of environment expected by our program logic in a way that is independent of specific technologies or hardware configurations*

Modeling QoS Domains in UML

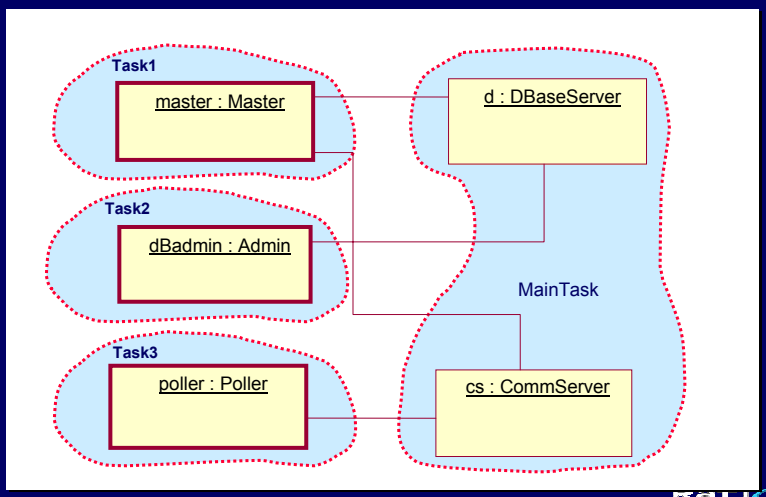
- Similar to realization: mapping of logical elements to a desired (required) engineering environment



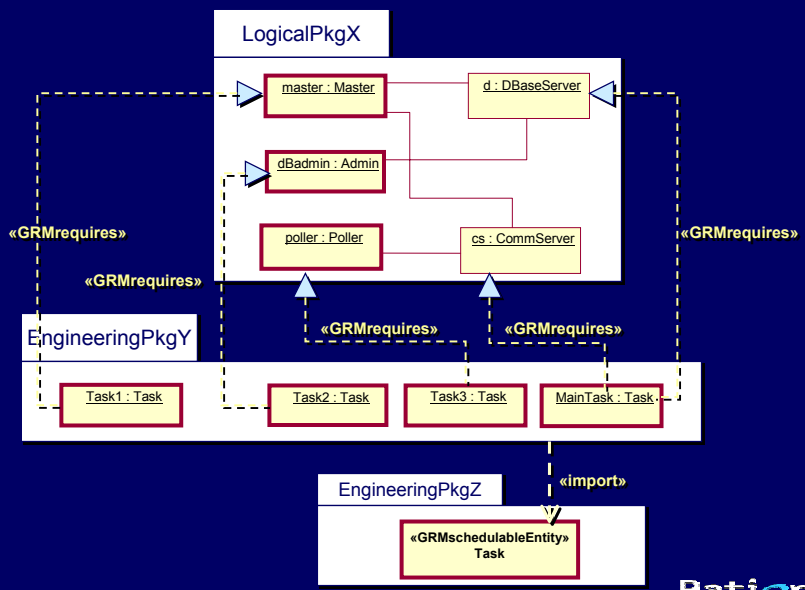
Stereotype	UML base concepts	Tags
GRMrequires	GRMrealizes	N/A

Example: Task Allocation

- The allocation of logical model to engineering model elements

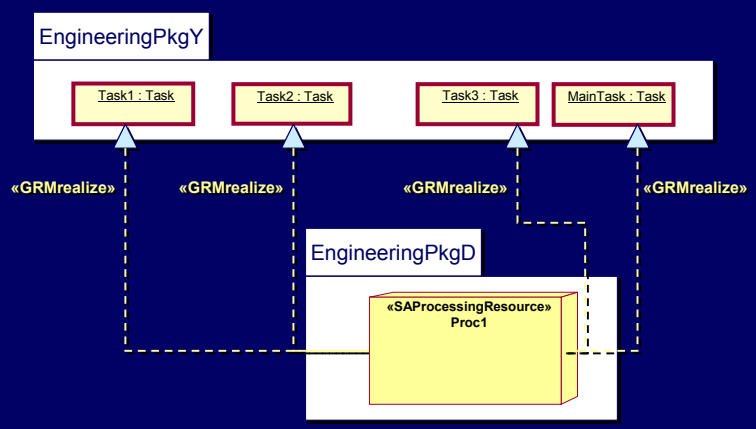


Example: UML Model of Allocation



Example: Completing the Mapping

- Mapping to hardware



Tutorial Structure

1. Models, software models, and software physics
2. A review and critique of UML
3. An introduction to the standard real-time UML profile
4. Modeling resources and quality of service
5. Modeling time in UML
6. Performance analysis with UML
7. Performance (engineering) oriented development
8. Summary and conclusions

Summary: Software, Modeling, and UML

- Software is traditionally viewed as a problem of logic rather than a problem of engineering
- However, software is realized by hardware and, hence, inherits its physical characteristics
- The physical characteristics of the supporting platform can have a major impact on application logic
- The Unified Modeling Language standard is used to model software systems, but
  - Has no standard facilities for modeling QoS characteristics
  - Has relatively weak facilities for modeling software platforms

Summary: The Real Time UML Profile

- The RT UML Profile defines a standard set of stereotypes for expressing platform-related concepts more precisely:
  - resources
  - concurrency mechanisms
  - time and timing mechanisms
- Furthermore, it allows the specification of quantitative aspects in the same models such that the models can be analyzed
  - predictive models that can be used to validate (risky) design approaches before major investments are made

Summary: Performance Analysis in UML

- The RT UML profile provides of a separate performance analysis sub-profile
  - Allows accurate specification of key performance concepts directly on UML models (i.e., a "performance analysis" overlay)
  - Eliminates the need for manual construction of a separate performance model (i.e., the model can be derived automatically)
- There is provision for further refinement of the performance sub-profile for specific requirements and modeling techniques

Conclusion

- With the availability of the following:
  - An industry standard modeling language for specifying software applications based on problem-domain concepts
  - A standard way of specifying quantitative characteristics of the application and its supporting platform
  - Computer-based tools conforming to the above standards which automate many error-prone design and verification tasks

…we can perhaps, at long last, raise the level of reliability of software engineering to that which we have come to expect of more traditional engineering disciplines

Bibliography

- Douglass, B., “Doing Hard Time,” Addison-Wesley, MA, 1999.
- Gomaa, H., “Designing Concurrent, Distributed, and Real-Time Applications with UML,”, Addison-Wesley, 2000.
- Klein, M., et al., “A Practitioner’s Handbook for Real-Time Analysis: Guide to Rate Monotonic Analysis for Real-Time Systems,” Kluwer Academic Publishers, The Netherlands, 1993.
- Kopetz, H., “Real-Time Systems: Design Principles for Distributed Embedded Applications, “Kluwer Academic Publishers, The Netherlands, 1997.
- Liu, J., “Real-Time Systems,” Prentice-Hall, NJ, 2000.
- Object Management Group, The, “UML Profile, for Schedulability, Performance, and Time,” OMG document ptc/2002-03-02 (http://www.omg.org/cgi-bin/doc?ptc/2002-03-02)
- Rumbaugh, J., Jacobson, I., and Booch, G., “The Unified Modeling Language Reference Manual,” Addison-Wesley, MA, 1999.
- Smith, C., and Williams, L., “Performance Solutions: A Practical Guide to Creating Responsive, Scalable Software,”, Addison-Wesley, MA, 2001.