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
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

Program Logic

HL Programming Languages

Computing Platform

The Case of Distributed Systems

♦ Possibility of out of date status information due to transmission delays

Processing Site

observer

"on"

State?

"on"

Processing Site

on

off

Copyright Bran Selić, 2002
More on Transmission Delays

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

![Diagram showing event orderings and inconsistent views of system state]

(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.

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

```
Airline
  owner 0..*

Airplane
designatedPlane 1

Flight
  route start duration
  0..* (ordered)

Pilot
crew 1..*

Captain

First Officer
```

```
Association

Association class
```
Object Instance Diagram

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

DecrepitAir : Airline
N1313: Airplane

CreakyAir : Airline

Link

Donald D.: Pilot
Mickey M.: Pilot

CA345 : Flight

CA123 : Flight

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

```
/CallProc
  ^   ^
 /   / 2.call() 3.sendTone()
|  |    |       |      |
| \
 V  V   1.OffHook() 4.dialtone() V

/IPhone IF

/ToneGen

Classifier role

Message: part of an Interaction
```

**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)

```
/Caller
call
ack
number
transfer

/Operator

/Callee
call
ack
talk

Rational
```

**time**
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

```
master : Master
d : DBaseServer
cs : CommServer

register( )
read( )

(register.receiveTime() - read.sendTime()) \leq 2 ms
```
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

![Diagram showing the hierarchy of UML languages: UML Standard 1.4, Real-Time UML, UML for eCommerce, and etc.](image.png)

Variations produced using extensibility mechanisms: stereotypes, tagged values, constraints = UML profiles

Specializing UML: Stereotypes

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

![Diagram showing the specialization of an Integer class into a MyClockClass with a «clock» stereotype and tagged value.](image.png)

An example of the UML Class concept

«clock» Stereotype of Class with added semantics: an active counter whose value changes synchronously with the progress of physical time

Tagged value associated with the «clock» stereotype

Copyright Bran Selic, 2002
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
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

**Model Editing Tool**

- Automated model conversion

**Model Analysis Tool**

- Inverse model conversion

Example: Schedulability Annotations
Example: Analysis Results

Additional tool-specific results encased in UML Notes

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)**

<table>
<thead>
<tr>
<th>Stereotype</th>
<th>Base Class</th>
<th>Tags</th>
</tr>
</thead>
<tbody>
<tr>
<td>«RTaction»</td>
<td>Action</td>
<td>RTstart, RTend</td>
</tr>
<tr>
<td></td>
<td>ActionExecution</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Message</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Stimulus</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Method</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ActionSequence</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ActionState</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SubactivityState</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Transition</td>
<td></td>
</tr>
<tr>
<td></td>
<td>State</td>
<td></td>
</tr>
</tbody>
</table>

**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
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

![Diagram showing resource usage and QoS attributes]

Simple Example

- Concurrent tasks accessing a monitor with known response time characteristics

![Diagram illustrating simple example with tasks and monitor QoS attributes]
♦ 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

♦ 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

- ResourceInstance (from CoreResourceModel)
  - protectionKind
  - activenessKind
  - purposeKind
    - ProtectedResource
    - UnprotectedResource
    - PassiveResource
    - ActiveResource
  - Device
  - Processor
  - CommunicationResource

Exclusive Use Resources and Actions

- AccessControlPolicy
  - 0..1
    - 0..n
    - ProtectedResource

- ResourceServiceInstance (from CoreResourceModel)
  - 1
  - 0..n

- ActionExecution (from DynamicUsageModel)
  - 1
  - 0..n
  - Blocking: Boolean
  - AcquireService
  - ReleaseService
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:

<table>
<thead>
<tr>
<th>Stereotype</th>
<th>UML base concepts</th>
<th>Tags</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRMclient</td>
<td>Classifier, Instance</td>
<td>GRMperiod, GRMwcet</td>
</tr>
<tr>
<td>GRMprotResource</td>
<td>Classifier, Instance</td>
<td>N/A</td>
</tr>
<tr>
<td>GRMresService</td>
<td>BehavioralFeature</td>
<td>GRMwcet</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tag</th>
<th>Tag Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRMperiod</td>
<td>RTtimeString</td>
</tr>
<tr>
<td>GRMwcet</td>
<td>RTtimeString</td>
</tr>
</tbody>
</table>

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

Example: QoS Annotations

Using the standard stereotypes...

```
«GRMclient»
master : Master
(GRMperiod = 100 ms) (GRMwcet = 20 ms)

«GRMclient»
#Badm : Admin
(GRMperiod = 150 ms) (GRMwcet = 40 ms)

«GRMclient»
operner : Poller
(GRMperiod = 350 ms) (GRMwcet = 100 ms)

1. read()

«GRMprotResource»
d : DBaseServer

2. register()

sort()

invokes()

«GRMprotResource»
cs : CommServer
```
Example: Class Diagram

- QoS annotations can be added to classes as well

```
«GRMclient»
  Master
    0..n
  1

«GRMprotResource»
  DBaseServer
    «GRMserver_read() (GRMWcet = 10ms)
    «GRMserver_sort() (GRMWcet = 20ms)

«GRMprotResource»
  CommServer
    «GRMserver_invoke() (GRMWcet = 10ms)
    «GRMserver_register() (GRMWcet = 20ms)

«GRMclient»
  Admin
    0..n

«GRMclient»
  Poller
    0..n
```

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

ResourceInstance (from CoreResourceModel)

Timeout (from TimedEvents)

Timer

isPeriodic : Boolean

1

0..n

+generatedTimeouts 0..n

Clock Interruption (from TimedEvents)

TimeInterval (from TimeModel)

Clock

1

0..n

+offset 1

0..n

1

+accuracy 1

0..n

+generatedInterrupts 0..n

TimedEvent (from TimedEvents)

TimeValue (from TimeModel)

1

0..n

+duration

10..n

TimingMechanism

stability

drift

skew

set(time : TimeValue)

generate() : TimeValue

reset()

start()

pause()

1

0..n

+resolution 1

0..n

1

+referenceClock 1

0..n

+generatedInterrupts

Clock

Timer

StartInterrupt (from TimeEvents)

Interrupt (from TimeEvents)

Example Timing Stereotype

<table>
<thead>
<tr>
<th>Stereotype</th>
<th>Base Class</th>
<th>Tags</th>
</tr>
</thead>
<tbody>
<tr>
<td>«RTAction»</td>
<td>Action</td>
<td>RTstart, RTend, RTduration</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tag</th>
<th>Tag Type</th>
<th>Multiplicity</th>
<th>Domain Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>RTstart</td>
<td>RTtimeValue</td>
<td>[0..1]</td>
<td>TimedAction:start</td>
</tr>
<tr>
<td>RTend</td>
<td>RTtimeValue</td>
<td>[0..1]</td>
<td>TimedAction:end</td>
</tr>
<tr>
<td>RTduration</td>
<td>RTtimeValue</td>
<td>[0..1]</td>
<td>TimedAction:duration</td>
</tr>
</tbody>
</table>
Timed Stimuli

These two associations are derived from the general association between EventOccurrence and Stimulus (from CausalityModel):

\[ \text{StimulusGeneration} \rightarrow \text{Stimulus} \rightarrow \text{StimulusGeneration} \]

(where \( n \) and \( m \) are natural numbers).

\[ \text{StimulusGeneration} \rightarrow \text{Stimulus} \rightarrow \text{StimulusGeneration} \]

\[ \text{Stimulus} \rightarrow \text{ClockInterrupt} \rightarrow \text{Stimulus} \]

Clock Interrupt

Timed Event

Timed Actions

Timed Events and Timed Actions

\[ \text{EventOccurrence} \rightarrow \text{TimedEvent} \rightarrow \text{Scenario} \rightarrow \text{TimedAction} \]

\[ \text{Scenario} \rightarrow \text{TimedAction} \rightarrow \text{Scenario} \]

\[ \text{TimeValue} \rightarrow \text{TimedEvent} \rightarrow \text{TimeValue} \]

\[ \text{ TimeInterval} \rightarrow \text{TimedAction} \rightarrow \text{TimeInterval} \]

Delay

Copyright Bran Selic, 2002
**Time Annotations**

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

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()`

  \{`call.sendTime()` - `ack.receiveTime` < 10 sec\}
## Defined Stereotypes (1 of 3)

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

## Defined Stereotypes (2 of 3)

<table>
<thead>
<tr>
<th>Stereotype</th>
<th>Applies To</th>
<th>Tags</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>«RTnewClock»</td>
<td>Operation</td>
<td>RTstart [0..1]</td>
<td>An operation that creates a new clock mechanism</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RTend [0..1]</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>RTduration [0..1]</td>
<td></td>
</tr>
<tr>
<td>«RTnewTimer»</td>
<td>Operation</td>
<td>RTtimerPar [0..1]</td>
<td>An operation that creates a new timer</td>
</tr>
<tr>
<td>«RTpause»</td>
<td>Operation</td>
<td>A pause operation on a timing mechanism</td>
<td></td>
</tr>
<tr>
<td>«RTreset»</td>
<td>Operation</td>
<td>A operation that resets a timing mechanism</td>
<td></td>
</tr>
<tr>
<td>«RTset»</td>
<td>Operation</td>
<td>RTtimePar [0..1]</td>
<td>An operation that sets the current value of a timing mechanism</td>
</tr>
<tr>
<td>«RTstart»</td>
<td>Operation</td>
<td>An operation that starts a timing mechanism</td>
<td></td>
</tr>
<tr>
<td>«RTstimulus»</td>
<td>Stimulus, ActionExecution, Action, ActionSequence, Method</td>
<td>RTstart [0..1]</td>
<td>A timed stimulus</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RTend [0..1]</td>
<td></td>
</tr>
</tbody>
</table>
Defined Stereotypes (3 of 3)

<table>
<thead>
<tr>
<th>Stereotype</th>
<th>Applies To</th>
<th>Tags</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>«RTtime»</td>
<td>DataValue, Instance, Object, DataType, Classifier</td>
<td>RTkind [0..1]</td>
<td>A time value or a time object</td>
</tr>
<tr>
<td>«RTtimeout» (subclass of «RTstimulus»)</td>
<td>Stimulus, ActionExecution, Action, ActionSequence, Method</td>
<td>RTtimestamp [0..1]</td>
<td>A timeout signal or a timeout action</td>
</tr>
<tr>
<td>«RTtimer» (subclass of «RTtimingMechanism»)</td>
<td>DataValue, Instance, Object, ClassifierRole, Classifier...</td>
<td>RTduration [0..1]</td>
<td>A timer mechanism</td>
</tr>
<tr>
<td>«RTtimeService»</td>
<td>Instance, Object, ClassifierRole, Classifier</td>
<td></td>
<td>A time service</td>
</tr>
<tr>
<td>«RTtimingMechanism»</td>
<td>DataValue, Instance, Object, ClassifierRole, Classifier, ObjectType</td>
<td>RTstability [0..1]</td>
<td>A timing mechanism</td>
</tr>
</tbody>
</table>

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)
- $\text{param}$, '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

Software Model

How do we relate these two?

Platform Model

Realization Mappings

Logical Model

Engineering Model

Copyright Bran Selic, 2002
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

![Diagram showing logical model with elements A and B and engineering model with elements X, Y, and Z. The logical model element A is associated with the engineering model elements X and Y, and B with Z via a stereotype of the UML realizes relationship.]

Compact tabular form:

<table>
<thead>
<tr>
<th>Source element</th>
<th>Dest. elements</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>X, Y</td>
</tr>
<tr>
<td>asc</td>
<td>Y</td>
</tr>
<tr>
<td>B</td>
<td>Z</td>
</tr>
</tbody>
</table>

RT UML: Specifying Realization Mappings

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

<table>
<thead>
<tr>
<th>Logical Elements</th>
<th>Engineering Elements</th>
<th>Mode</th>
<th>Linkage</th>
<th>Additional Constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;List of logical model elements&gt;</td>
<td>&lt;List of corresponding engineering model elements&gt;</td>
<td>&lt;If there are multiple engineering elements, one of: (inclusive, exclusiveStatic, exclusiveDynamic)&gt;</td>
<td>&lt;Interaction mode between levels, one of: (sync, async, replace)&gt;</td>
<td>&lt;Any additional constraints that apply to the mapping&gt;</td>
</tr>
</tbody>
</table>

- Diagram showing logical model elements L1 and E1 connected to engineering model elements L and E. The mode is sync = SW to SW, async = SW to SW, replace = SW to HW.
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

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 = $N_{users}$
- Average inter-session times = 20 minutes
- Frames in a video $N$
- Video frame interval = 30 ms
- Required confirmation delay: 95% < 500 ms
- Required interval between frame displays = 99% < 30 ms

Example: Annotations for a Scenario
Example: More Annotations

```
<table>
<thead>
<tr>
<th>Stereotype</th>
<th>Applies To</th>
<th>Tags</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>«PAclosedLoad»</td>
<td>Action, ActionExecution,</td>
<td>PArespTime [0..*]</td>
<td>A closed workload</td>
</tr>
<tr>
<td></td>
<td>Stimulus, Action, Message,</td>
<td>PApriority [0..1]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Method…</td>
<td>PApopulation [0..1]</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>PAextDelay [0..1]</td>
<td></td>
</tr>
<tr>
<td>«PAcontext»</td>
<td>Collaboration,</td>
<td></td>
<td>A performance analysis</td>
</tr>
<tr>
<td></td>
<td>CollaborationInstanceSet</td>
<td></td>
<td>context</td>
</tr>
<tr>
<td></td>
<td>ActivityGraph</td>
<td></td>
<td></td>
</tr>
<tr>
<td>«PAhost»</td>
<td>Classifier, Node,</td>
<td></td>
<td>A deferred receive</td>
</tr>
<tr>
<td></td>
<td>ClassifierRole, Instance,</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Partition</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>PAschedPolicy [0..1]</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>PArate [0..1]</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>PAactSwT [0..1]</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>PApreemptible [0..1]</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>PAthroughput [0..1]</td>
<td></td>
</tr>
<tr>
<td>«PAopenLoad»</td>
<td>Action, ActionExecution,</td>
<td>PArespTime [0..*]</td>
<td>An open workload</td>
</tr>
<tr>
<td></td>
<td>Stimulus, Action, Message,</td>
<td>PApriority [0..1]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Method…</td>
<td>PApopulation [0..1]</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
Defined Stereotypes (2 of 2)

<table>
<thead>
<tr>
<th>Stereotype</th>
<th>Applies To</th>
<th>Tags</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>«PAresource»</td>
<td>Classifier, Node, ClassifierRole, Instance, Partition</td>
<td>PAutilization [0..*] PA@schedPolicy [0..1] PA@capacity [0..1] PA@maxTime [0..1] PA@respTime [0..1] PA@waitTime [0..1] PA@throughput [0..1]</td>
<td>A passive resource</td>
</tr>
<tr>
<td>«PStep»</td>
<td>Message, ActionState, Stimulus, SubactivityState</td>
<td>PA@demand [0..1] PA@respTime [0..1] PA@prob [0..1] PA@delay [0..1] PA@extOp [0..1] PA@interval [0..1]</td>
<td>A step in a scenario</td>
</tr>
</tbody>
</table>

Specifying Performance Values

♦ A complex structured string with the following format
  - `<kind-of-value> , <modifier> , <time-value>`
  - `<kind-of-value>` ::= 'req' | 'assm' | 'pred' | 'msr'
  - `<modifier>` ::= 'mean' | 'sigma' | '<kth-mom' , <Integer> | 'max' | 'percentile' <Real> | 'dist'

♦♦ A complex structured string with the following format
  - `<kind-of-value> , <modifier> , <time-value> , <modifier> , <time-value>`
  - `<kind-of-value>` ::= 'req' | 'assm' | 'pred' | 'msr'
  - `<modifier>` ::= 'mean' | 'sigma' | '<kth-mom' , <Integer> | 'max' | 'percentile' <Real> | 'dist'

♦♦♦ A complex structured string with the following format
  - `<kind-of-value> , <modifier> , <time-value> , <modifier> , <time-value> , <modifier> , <time-value>`
  - `<kind-of-value>` ::= 'req' | 'assm' | 'pred' | 'msr'
  - `<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

- 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

**Logical View**

**Engineering View (alternative A)**
- UNIX Process

**Engineering View (alternative B)**
- WinNT Process

**Required Environment Partitions**

- Logical elements often share common QoS requirements

**QoS domain** (e.g., failure unit, uniform comm properties)
**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

![Diagram showing QoS Domains in UML](image)

**Logical View**

- **«GRMrequires»:** a specialization of the "realizes" relationship

**Required Environment**

- QoS1
- QoS2
- QoS3

- A local "engineering" model

<table>
<thead>
<tr>
<th>Stereotype</th>
<th>UML base concepts</th>
<th>Tags</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRMrequires</td>
<td>GRMrealizes</td>
<td>N/A</td>
</tr>
</tbody>
</table>
Example: Task Allocation

- The allocation of logical model to engineering model elements

Example: UML Model of Allocation

- LogicalPkgX
- EngineeringPkgY
- EngineeringPkgZ

- Task1
- Task2
- Task3
- MainTask

- master : Master
- d : DBaseServer
- dBadmin : Admin
- poller : Poller
- cs : CommServer
Example: Completing the Mapping

- Mapping to hardware

![Diagram showing tasks and packages related to 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 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