The Emerging Real-Time UML Standard: Theory and Practice

Bran Selic
Rational Software Canada
bselic@rational.com
Tutorial Objectives

♦ To clarify the relationship between the object paradigm and real-time systems
  ■ match or mismatch?

♦ Describe and analyze UML from a real-time designer’s perspective

♦ To introduce the “UML Profile for Schedulability, Performance, and Time”

♦ To introduce an engineering-oriented design approach for real-time systems
Tutorial Overview

♦ Real-Time Systems and the Object Paradigm
♦ UML as a Real-Time Modeling Language
♦ The Real-Time UML Profile
♦ Engineering-Oriented Design of Real-Time Systems
♦ Summary and Conclusions
Real-Time Systems and the Object Paradigm

- Real-Time System Essentials

- UML as a Real-Time Modeling Language

- The Real-Time UML Profile

- Engineering-Oriented Design of Real-Time Systems

- Summary and Conclusions
Real-Time System

- Systems that maintain an ongoing timely interaction with its environment

• outputs = f (inputs, state)
• coordination

Real-Time System (state)
A persistent structure that provides a framework for behavior

- Real-Time System (state) signals sources
- Controlled entities inputs signals sources
- Outputs = f (inputs, state)
- Coordination

Concurrency conflict
Classifications of RT Systems

- Based on nature of key inputs
  - *time-driven*: for continuous (synchronous) inputs
  - *event-driven*: for discrete (asynchronous) inputs

- Based on time criticality
  - *hard RT systems*: every input must have a timely response
  - *soft RT systems*: most inputs must have timely response

- Based on load:
  - *static*: fixed deterministic load
  - *dynamic*: variable (non-deterministic) load

- Many practical systems are combinations of these
Which procedure(s) describe this system?
Classical Approach: Cyclical Executive

The miscellaneous procedural slices are executed cyclically based on time resolution.

- **= 50 msec band**
- **= 100 msec band (2 parts: A and B)**
- **= 200 msec band (4 parts: A, B, C, D)**
- **= 400 msec band (8 parts: A, B, C, D, E, F, G, H)**

The system structure is almost completely obscured.
Real-Time Systems and the Object Paradigm

- Real-Time System Essentials
- Essentials of the Object Paradigm

UML as a Real-Time Modeling Language

The Real-Time UML Profile

Engineering-Oriented Design of Real-Time Systems

Summary and Conclusions
Objects and Real-Time Systems

- The structure of real-time systems tends to persist through time because it reflects the physical entities of the real world.
- This structure is the framework through which (infinitely) many different behavior threads are executed.
- Hence, the focus is on structure rather than behavior.
- *The structural focus of the object paradigm is better suited to real-time systems than the procedural paradigm.*
Yes, But What About...

- **Performance?**
  - the cost of abstraction (encapsulation, automatic garbage collection, dynamic binding, etc.)

- **Modeling real-time specific phenomena?**
  - time and timing mechanisms
  - resources (processors, networks, semaphores, etc.)

- **Exploiting current real-time system theory?**
  - schedulability analysis (e.g., rate-monotonic theory)
  - performance analysis (queueing theory)
Performance of OO Technology

- Hardware is becoming ever faster (Moore’s law)
  - previously unacceptable response times may now be acceptable

- OO software technologies are becoming real-time aware
  - bounded dynamic binding techniques
  - tunable automatic garbage collection (bounded latency)
  - real-time variants of popular OO languages (e.g., EC++, RT Java)
The Essence of the Object Paradigm

- Combines all the various features of a logical unit (procedures and data) into a single package called an object.

Defines a software system as a structure of collaborating objects.
Objects

- Conceptual units with
  - a unique identity (dedicated memory)
  - a public interface
  - a hidden (encapsulated) implementation

Telephone1:
- busy : boolean
- offHook()
- onHook()
- ring()

void:offHook ();
{busy = true;
 reqDialtone();
 ... }

Data Attributes

Operations
The object paradigm allows us to create our own (virtual) reality!

- Not all objects necessarily require a physical underpinning
- For example, the “telephone call” object

```
 Telephone1  Telephone2

 Telephone Call Object

 abortCall ()
 addParty (t:Telephone)
 reportDuration ()
```
Object Behavior

♦ In essence, an object is a *server*

- generic object lifecycle:

  - Handling depends on specific request type and object state
  - Invokes operations on other objects

```c
void:offHook ()
{busy = true;
 obj.reqDialtone();
 ... 
};
```

```
void:offHook ()
{busy = true;
 obj.reqDialtone();
 ... 
};
```
In essence, an object is a **server**

- Handling depends on specific request type and object state
- `void: offHook(); {busy = true; obj.reqDialtone(); ... };`
- Invokes operations on other objects
Higher-level behavior “emerges” through the interactions of individual objects.
Objects and Emergent Behavior

♦ One of the main problems of many current OO programming languages is that they do not provide a means for specifying high-level emergent behavior

- “keyhole” view of high-level behavior
- difficult to ensure desired high-level behavior will necessarily emerge

♦ A conflict between top-down and bottom-up design approaches

- re-usable component programming style defines objects independently of the sequences in which they may participate
More than one object can be constructed from the same specification—the class
- A design environment concept

Objects created from some class specification are called instances of that class
- A run-time concept
Inheritance and Polymorphism

- A generalization and re-use mechanism

**GenericTelephone**
- `busy : boolean`
- `offHook()`
- `onHook()`
- `dialDigit()`

**TouchTone Telephone**
- `busy : boolean`
- `offHook()`
- `onHook()`
- `dialDigit()`
- `playTone()`

**RotaryDial Telephone**
- `busy : boolean`
- `offHook()`
- `onHook()`
- `dialDigit()`

**Polymorphism:** different realizations of the same operation

**Generalization (inheritance) relationship**
Objects: Summary

- The object paradigm is very well adapted to real-time software systems because of its powerful structural modeling capability
  - networks of collaborating objects
- In addition, the object paradigm comes packaged with a number of well-established techniques:
  - modularity
  - information hiding
  - generalization/refinement mechanisms (e.g., inheritance)
  - genericity
Real-Time Systems and the Object Paradigm

- Real-Time System Essentials
- Essentials of the Object Paradigm

UML as a Real-Time Modeling Language

The Real-Time UML Profile

Engineering-Oriented Design of Real-Time Systems

Summary and Conclusions
The Unified Modeling Language

- A consolidation of proven ideas and practices based on the object paradigm into a general-purpose OO modeling language
  - Initiated by Rational Software (Booch, Rumbaugh, Jacobson)
- Standardized by the Object Management Group in 1997
- Major advantages:
  - widely adopted by software practitioners
  - widely taught in universities and technical seminars
  - supported by many software tool vendors
The Unified Modeling Language

- Combination of proven OO methods

- Harel (statecharts)
- Booch
- Rumbaugh (OMT)
- Jacobson (OOSE)
- + many others

UML 0.9 → UML 1.1 → UML 1.3 → UML 1.4

OMG standard  

June 2001
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)
Introducing Viewpoints

♦ **Viewpoint:** a model of a system based on a particular set of related concerns
  - abstracts out detail that is irrelevant for that set of concerns

Radio: Electronic design viewpoint
UML uses a model-based approach rather than a view-based approach.

Model-view consistency is enforced through the UML metamodel.
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

[Diagram]

Telephone Switch

- instructor
- actor
- trainee

Use case

- Fail one engine
- Prepare flight
- Land plane
- Run checklist

«include»
Use Cases and RT Systems

♦ As useful as in any other domain
  ■ fundamental drivers of 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,…
  ■ No standard way of associating such non-functional requirements with use cases
  ■ Use cases do not deal with many important “ilities” (availability, reliability, maintainability,…) that are critical in many real-time systems
Class Diagram

♦ Shows the entities in a system and their general relationships
Object Instance Diagram

♦ Shows object instances in a particular case

DecrepitAir : Airline

N1313: Airplane

Donald D. : Pilot

Mickey M. : Pilot

Link

CA345 : Flight

CA123 : Flight

DecrepitAir : Airline

CreakyAir : Airline
Class diagrams are very abstract and sometimes leave out crucial system information (e.g., topology)

- e.g., common class diagram for both systems
Object (instance) diagrams do show topologies

However...

- in principle, object diagrams only represent “snapshots” of a system at a particular point in time
- no guarantee that they hold throughout the lifetime of the system
- need “prototypical” object diagrams
- but, such semantics are not defined in the current standard
Collaboration Diagram

♦ Depict generic structural and behavioral patterns

P2:TTSet

1. offHook()
2. call()

P1:BusSet

3. sendTone()
4. dialtone()

/CallProc

/ToneGen

NB: It is possible to have collaboration diagrams without an Interactions 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
/Callee
  talk
/Operator
  call
  ack
```

Rational the e-development company™
Sequence Diagrams and RT Systems

♦ Sequence diagrams are extremely useful for showing object interactions
  ■ very common in many 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 timing information
  ■ UML sequence diagrams do not scale up very well for modeling large systems with complex sequences
Using Timing Marks with Sequence Diagrams

♦ Specifying constraints

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

(read.endTime( ) - read.startTime( )) ≤ 2 ms
(register.endTime( ) - register.startTime( )) ≤ 10 ms
```
Activity Diagrams

Different focus compared to sequence diagrams

- Activity Diagrams
- ♦ Different focus compared to sequence diagrams

- /Caller
- /Operator
- /Callee

- Contact Operator
- Contact Callee
- Notify Parties
- Respond
- Acknowledge
- Acknowledge

- activity
- swimlane
Better than sequence diagrams for

- showing concurrency (forks and joins are explicit)
- scaling up to complex systems

However...

- No standard way of denoting timing information
- Less well-suited for describing event-driven behavior
Each state corresponds to a selective receive action

- State machine
- State
- Trigger
- Action expression
- Final state
- Transition
- Initial pseudostate
- Start
  - start/^master.ready()
Hierarchical States and Transitions

- Allows step-wise refinement and viewing of complex behavior
State Machines and RT Systems

- Many real-time systems are event-driven
  - very well suited to those systems
  - scale up very nicely
- However...
  - not directly connected to time (except for time events)
  - e.g., run-to-completion paradigm
Implementing Time-Triggered Systems

♦ Periodic timers:
  - once initiated they repeatedly send TimeEvents at the appropriate intervals until explicitly stopped or cancelled

♦ In “steady-state” mode, active objects stimulated exclusively by periodic timers become periodic tasks
  - allows rate-monotonic scheduling policies
  - schedulers use the priorities of periodic timers to make scheduling decisions
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
The Active Objects of UML

- Single thread of execution
- Behavior defined by state machines (event driven)
Active Object Semantics

♦ Concurrent incoming events are queued and handled one-at-a-time regardless of priority
♦ run-to-completion (RTC) execution model

RTC eliminates potential concurrency conflicts
RTC Semantics

- A high priority event for another active object will preempt an active object on the same processor that is handling a low-priority event
  - Limited priority inversion can occur

![Diagram showing RTC semantics with high (hi) and low (lo) priority events, including queuing and priority inversion.]
RTC Analysis

♦ Advantages:
  ■ Eliminates concurrency conflicts for all passive objects encapsulated by active objects
  ■ No explicit synchronization code required
  ■ Low-overhead context switching (RTC implies that stack does not need to be preserved)

♦ Disadvantage:
  ■ Limited priority inversion can occur (higher priority activity may have to wait for a lower-priority activity to complete)
  ■ Can be circumvented but at the expense of application-level complexity
Example: Active Objects

♦ Active object ≠ OS thread
  ■ two-tier scheduling scheme
  ■ event priorities vs thread priorities

OSProcess

OSThread1

OSThreadN

Active object scheduler
Active object scheduler
... 

Operating System
(schedules process and threads)

Lightweight specialized multitasking

Heavyweight general-purpose multitasking
Active objects are the major concurrency mechanism of UML
- automatically resolve certain concurrency conflicts

However...
- The priority inversion inherent in RTC may be unacceptable in some cases
- How does this map to concurrency mechanisms that are used in the real-time domain (processes, threads, semaphores, real-time scheduling methods, etc.)?
- No clear way of exploiting real-time analyses methods (e.g., schedulability analysis)
Scheduling in UML

- Scheduling is based on events, not threads
  - Priorities typically attached to events
  - Timing events allow realization of time-triggered systems
- The actual scheduling policy is unspecified
  - A semantic variation point
  - Can be customized to suit application requirements
The Model of Time in UML

- Unbiased and uncommitted (i.e., it does not exist):
  - Time data type declared but not defined (could be either continuous or discrete)
  - No built-in assumptions about global time source (open to modeling distributed systems)

- Related concepts:
  - Time events: generated by the occurrence of a specific instant
  - Assumes some kind of run-time Timing Service
Implementation Diagrams

♦ Implementation focus

Component

reservations

Node

update

:Scheduler

:Planner

:GUI
Implementation Diagrams and RT Systems

♦ Probably the weakest part of UML

- not sophisticated enough to capture the various complex aspects of deployment common to real-time 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)

- ......
UML Summary

♦ An industry standard for analysis and design of object-oriented systems
  - based on extensive experience and best practices
  - gaining rapid acceptance (training, tools, books)
♦ Comprises:
  - set of modeling concepts
  - a standard graphical notation
♦ Represented through 8 different diagram types
  - class, state machine, collaboration, use case, sequence, activity, component, deployment
Using UML for real-time 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 in the real-time domain
- missing or non-standard concepts
- ability to create predictive models for real time
Real-Time Systems and the Object Paradigm
- Real-Time System Essentials
- Essentials of the Object Paradigm

UML as a Real-Time Modeling Language

The Real-Time UML Profile

Engineering-Oriented Design of Real-Time Systems

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
Specialization of UML

- Avoiding the PL/I syndrome ("language bloat")
  - UML standard as a basis for a "family of languages"

Variations produced using built-in extensibility mechanisms: stereotypes, tagged values, constraints

UML Standard 1.4

Real-Time UML

UML for eCommerce

.....etc.
How Do We Specialize UML?

- Typically used to capture semantics that cannot be specified using UML itself.

But, how can we specify a clock?

Semantics:
- an active counter whose value changes synchronously with the progress of physical time.

Specialization through regular inheritance:

- Integer
- Counter
- ResetCounter()
Example: a “clock” stereotype based on the generic UML Class concept

Semantics:
an active counter whose value changes synchronously with the progress of physical time
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
UML Extensibility and RT Systems

♦ The extensibility mechanisms of UML provide an excellent opportunity to fill in the missing bits for real-time applications

♦ If we can define a standard set of extensions ("real-time profile") then these could provide a common facility for real-time UML modelers and tool builders
The Real-Time A&D Group in OMG

◆ An OMG working group
  - mission: to investigate and issue requests (RFPs) for standard ways and means to apply UML to real-time problems

◆ Three principal areas of investigation:
  - Time-related modeling issues
  - General quality of service/fault tolerance modeling issues
  - Architectural modeling issues

◆ Status:
  - first RFP issued (April 1999)
  - second RFP being drafted
The Real-Time UML RFP

First request for proposal (RFP.1): “UML profile for scheduling performance and time”
- initial proposal submitted in August 2000 (ad/2000-08-04)
- revised proposal due June 18, 2001

Requirements (modeling of):
- Physical time
- Timing specifications
- Timing services and mechanisms
- Modeling resources (logical and physical)
- Concurrency and scheduling
- Software and hardware infrastructure and their mapping
- Notation
The RFP does not ask for new real-time concepts or methods.

Instead, 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.
Response to the RFP

♦ Single submission

♦ Consortium team:
  - ARTiSAN (UML tool vendor)
  - I-Logix (UML tool vendor)
  - Rational (UML tool vendor)
  - Telelogic (UML tool vendor)
  - TimeSys (RT tool vendor)
  - Tri-Pacific (RT tool vendor)

♦ In consultation with many of the top real-time system experts in academia and industry
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

Current quantitative analysis methods:

- 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:

![Diagram showing a model of system analysis and results](image)
Desired Development Model

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

Model Editing Tool

Model Analysis Tool

Automated model conversion

Inverse model conversion
An abstract, technology-independent representation of the engineering model can be specified using the general concept of **Quality of Service (QoS):**

*a specification (usually quantitative) of how 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
Resource: an element whose service capacity is limited, directly or indirectly, by the finite capacities of the underlying physical computing environment.

The services of a resource are characterized by one or more quality of service (QoS) attributes:
- capacity, reliability, availability, response time, etc.
Simple Example

♦ Concurrent tasks accessing a monitor with known response time characteristics

Client1

access ( )
{Deadline = 3 ms}

Client2

access ( )
{Deadline = 5 ms}

myMonitor

{MaxExecutionTime = 4 ms}

Required QoS

Offered QoS
The General Resource Modeling Framework

♦ A conceptual domain model (not a metamodel)
More Resource Concepts

- **QoSCharacteristic**
  - offeredQoS
  - 0..n

- **Resource**
  - protection
  - activeness
  - 0..n

- **AccessControl Policy**
  - 1..n

- **Protected Resource**
  - 0..n

- **Unprotected Resource**
  - 1..n
  - 0..1

- **ActiveResource**
  - 0..1

- **PassiveResource**
  - 0..n
Resource Access and Management

- **Resource** manages **ResourceManager** (1..* \(\rightarrow\) 0..1)
- **ResourceManager** allocates **ResourceBroker** (1..* \(\rightarrow\) 0..1)
- **Resource** has **SharedResource** and **ExclusiveResource**
  - **SharedResource** has `+capacity : Integer`
  - **ExclusiveResource**
Elements of the general resource model are represented as stereotypes (with tags) of base UML concepts:

- **GRMresource**: Classifier, Instance, N/A
- **GRMresSrvc**: BehavioralFeature, GRMexTime

```
«GRMresource»
BufferPool

«GRMresSrvc» GetBuffer() {GRMexTime = 5}

«GRMresource»
apPool : BufferPool
```
Example System

♦ Periodic concurrent tasks sharing resources

- master : Master
  - Period = 100 ms
  - WCET = 20 ms

- dBadmin : Admin
  - Period = 150 ms
  - WCET = 40 ms

- poller : Poller
  - Period = 350 ms
  - WCET = 100 ms

- d : DBaseServer
  - WCET = 2 ms

- cs : CommServer
  - WCET = 10 ms

Example System

Period = 100 ms
WCET = 20 ms

Period = 150 ms
WCET = 40 ms

Period = 350 ms
WCET = 100 ms

WCET = 2 ms

1. read()

2. register()

sort()

invoke()

invoke()
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>

* The stereotypes and tags have been simplified for this presentation
Example: QoS Annotations

♦ Using the standard stereotypes...

```
«GRMclient»
master : Master
{GRMperiod = 100 ms}
{GRMwcet = 20 ms}

«GRMclient»
dBAdmin : Admin
{GRMperiod = 150 ms}
{GRMwcet = 40 ms}

«GRMclient»
poller : Poller
{GRMperiod = 350 ms}
{GRMwcet = 100 ms}

«GRMprotResource»
d : DBaseServer

1. read( )

2. register( )

sort( )

«GRMprotResource»
cs : CommServer

invoke( )
```
QoS annotations can be added to classes as well.
Example: Model Analysis

{isSchedulable= $.?

Schedulability Analyzer (RMA)

Result: a new model with analysis variable values set

May include additional tool-specific results encased in UML Notes

{isSchedulable= True

Result: a new model with analysis variable values set

May include additional tool-specific results encased in UML Notes
«RTtime»: a stereotype of Classifier (and Instance)
- supports both continuous and discrete time representations
- e.g. as a kind of real value

Stereotyping identifies any use of this class as representing time
Specifying Time Values

Time values can be represented by a special stereotype of Value («RTtimeString») 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”
A General Model of Time and Timing Mechanisms

- Guidelines for modeling time and timing facilities
  - Based on the generic resource model

Diagram:
- Resource
  - Timing Mechanism
    - Clock
    - Timer
Modeling Timers

Resource that generates events when a particular instant in time has been reached.
Modeling Clocks

♦ Resource for telling the “time of day”

- Time Value
  - start
  - stop
  - currentValue
  - maximalValue
  - resolution

- Clock
  - precision
  - stability
  - set (Time)
  - read () : Time
  - stop ()
  - resume ()

- Duration
  - 1

- Event
  - origin

- QoS Characteristics
  - offset
  - skew
  - drift

- standard
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}
```
Concurrency at the Engineering Level

- Generic model of a multi-tasking environment
The RT UML Profile defines a set of extensions for directly expressing real-time domain concepts in UML:

- 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
Real-Time Systems and the Object Paradigm
- Real-Time System Essentials
- Essentials of the Object Paradigm

UML as a Real-Time Modeling Language

The Real-Time UML Profile

Engineering-Oriented Design of Real-Time Systems

Summary and Conclusions
Common Wisdom...

♦ When designing software, we are instructed to ignore details of the technology and similar “implementation” issues until we have a sound logical solution to the problem.

  ■ simplifies the design problem (separation of concerns)
  ■ software is portable to new/different technologies

♦ But, what about real-time systems?
The idealized “forms” of pure logic acquire the finite characteristics of the physical stuff out of which they are spun:
- limited speed, limited capacity, limited availability,...
Common Aspects of All RT Design

*The quantitative aspect is significant*
- Time sensitive (metric view of time is needed)
- Resource sensitive (memory size, channel capacity)

*Concurrency is an inherent part of the problem*
- The real world is concurrent
- Concurrent activities need to be coordinated
- Concurrent activities may be asynchronous (non-deterministic)
Timeliness

♦ Does not always imply micro-second response

♦ An output should occur within some acceptable time interval following the input:

\[ T_{\text{min}} \leq t_{\text{output}} \leq t_{\text{input}} + T_{\text{max}} \]

♦ More generally: the value of outputs is a function of time

It’s a game of numbers!
Reasoning About Concurrency

♦ Zeno’s fable: Achilles and the Tortoise

It seems as if Achilles will never overtake the Tortoise!
It is difficult to reason about concurrency
Complex RT Systems

- Real-time systems with requirements for:
  - supporting complex functionality
  - high dependability (availability, reliability, safety)
  - distribution
  - heterogeneity
  - evolvability
  - scalability

- While we now have pre-packaged solutions for many categories of small-scale real-time systems, we are only starting to learn how to construct complex real-time systems
Fault-Tolerance

♦ Example: using spare processor capacity
  - failure detection
  - fault diagnosis
  - failure recovery

Optimal recovery strategies may be based on current needs and state of the application!
Distributed software system:

A system with two or more *independent processing sites* that communicate with each other over a medium whose *transmission delays* may exceed the time between successive state changes.
Real System Design Issues (1)

- Possibility of out-of-date state information due to lengthy (and variable) transmission delays

It’s a game of numbers!
Inconsistent views of system state:
- different observers see different event orderings

It’s a game of numbers!
Communication Media Failures

- **Message loss**
  - due to hardware failures
  - due to software failures (e.g., buffer overflow)

- **Message reordering**
  - due to different paths
  - due to variable delays (e.g., due to variable message lengths)
  - retransmission due to fault-tolerant protocols

- **Message duplication**
  - due to faulty hardware
  - retransmission due to fault-tolerant protocols
Distributed System Characteristics

- Key characteristics:
  - concurrency and asynchrony
  - need for communication and synchronization between sites
  - communication delays
  - possibility of partial failure

- Each of these adds significant “weight” to the programming problem

- Distributed programming is different from and much more complex than conventional programming
Real-World Real-Time Design Issues

♦ Much of the complexity associated with these systems is the result of the “intrusion” of the inherently complex physical world into the idealized logical world of software.

♦ The 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 portability?
Logical (Conceptual) Viewpoint

- A technology-independent view of the software
  - a “virtual” mechanism realized by a computer
The realization of a specific set of logical components using facilities of the run-time environment
Viewpoints and Mappings

Logical Viewpoint

- INSTRUCTOR STATION
  - AIRFRAME
    - ATMOSPHERE MODEL
    - GROUND MODEL
    - ENGINES
  - CONTROL SURFACES
  - PILOT CONTROLS

Engineering Viewpoint

- Processor
  - OS process
    - stack
  - Ethernet LAN
    - TCP/IP socket
  - OS process
    - stack
    - TCP/IP socket

Realization mappings
Realization Mappings

- A correspondence between elements of two distinct models (logical and engineering)
- Semantics: the logical elements are *implemented* by the corresponding engineering model elements
  - logical elements can be viewed as “residing” on the corresponding engineering elements
Selecting a Level of Abstraction

- Intermediate levels may be abstracted out
  - depends on the desired granularity of modeling
  - affects the semantics of the realization relationship
The engineering viewpoint represents the “raw material” out of which we construct the logical viewpoint.

- The quality of the outcome is only as good as the quality of the ingredients that are put in.

- As in all true engineering, the quantitative aspects are often crucial (How long will it take? How much will be required?...)

115
Engineering-Oriented Design (EOD)

- **Analysis and design of software systems based on**
  - *Models*
  - *QoS specifications (accounting for physical properties)*
  - *Quantitative analysis techniques and simulation*

- Complements any model-based development method

- **Advantages:**
  - Higher reliability (simplification due to modeling)
  - Ability to predict system characteristics (and major design flaws) prior to full realization
  - Portability!
Dilemma: How can we account for the engineering aspects of the system without prematurely and possibly unnecessarily committing to a particular technology?

Approach: Provide an abstract technology-independent but quantified specification of the required characteristics of the engineering model as part of the logical model.
**Viewpoint Separation**

- **Required Environment**: a technology-neutral environment specification required by the logical elements of a model

![Diagram of Logical Viewpoint, Required Environment, and Engineering Viewpoints](image)

- Engineering Viewpoint (alternative A): UNIX Process
- Engineering Viewpoint (alternative B): WinNT Process
Logical elements often share common QoS requirements

- Required Environment Partitions
- INSTRUCTOR STATION
- AIRFRAME
- CONTROL SURFACES
- ENGINES
- PILOT CONTROLS
- ATMOSPHERE MODEL
- GROUND MODEL

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.

- The QoS values of a domain can be compared against those of a concrete engineering environment to see if a given environment is adequate for a specific model.
Two Interpretations of Resource Model

♦ The peer interpretation

♦ The layered interpretation (the 2-viewpoint model)
The Layered Interpretation

⇒ The same QoS framework can be used for both the layered and peer modeling situations

Airframe

Mem : 2MB
CPU : 3 MIPs
Bandw. : 70Mbit/s

20MB
3MIPs
100Mbit/s

CPU
LAN

client
required QoS values
resource “usage”
environment “services” (offered QoS values)
resources
Example: Task Allocation

- The allocation of logical model to engineering model elements

Master

Poller

DBaseServer

Admin

CommServer
Example: Completing the Mapping

Mapping to hardware

- Task1: Task
- Task2: Task
- Task3: Task
- MainTask: Task

EngineeringPkgY

EngineeringPkgD

«GRMrealize»

«SAProcessingResource»

Proc1
In complex RT systems, the logical design is strongly influenced by the characteristics of the engineering environment.

In such systems, it is often crucial to formally determine if a system will meet its non-functional requirements (throughput, response time, availability, etc.).

The QoS-based approach described here can serve as a basis for:

- quantitative analysis of UML-based models
- a real-time modeling standard that will facilitate automated exchange between design and analysis tools
Complexity!

The design of real-time systems is influenced significantly by the physical properties of:

- the environment in which the system exists
- the implementation technology

Most of them stem from the physical world

- the physical dimension plays a major role in the design of real-time software since it imposes limitations on the logical design
Summary: The Solution (1 of 2)

♦ **The object paradigm**
  - reduces incidental complexity
  - its structural bias is better suited to the real-time domain than the procedural paradigm
  - additional key features (encapsulation, inheritance, polymorphism, etc.) add further expressive power

♦ **Engineering-oriented design**
  - accounts for the physical dimension during logical design
  - based on a quality of service (QoS) framework as represented in the generic resource model
  - allows de-coupling from actual implementation technologies (through required environment specifications)
  - suitable for analysis and synthesis
  - enables early detection of critical design flaws
SUMMARY: THE SOLUTION (2 OF 2)

- **UML provides a common and standardized underpinning that supports all the components of our solution**
  - for object-oriented modeling
  - for predictive QoS modeling (via the real-time profile)
  - for design analysis and synthesis (tool interchange)
  - for architectural definition
  - for implementation (through full automatic code generation)
- **Furthermore, as a standard, it enables model interchange between specialized tools and is a basis for significant automation of the RT software development process**
Bibliography