MARTE: The Future of the UML Profile for Schedulability, Performance and Time

Murray Woodside and Dorina C. Petriu
Carleton University
Department of Systems and Computer Engineering
Ottawa, Canada, K1S 5B

What is a UML Profile
- 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
- Does not extend the UML metamodel
- Uses only standard extension mechanisms: stereotypes, tagged values, constraints
- Additional semantic constraints cannot contradict the general UML semantics
- Within the “semantic envelope” defined by the standard

Specializing UML: Stereotypes
- We can add semantics to any standard UML concept (metaclass)
- The extensions are defined as stereotypes that apply to existing metaclasses.
- Must not violate standard UML semantics

Three OMG Profiles
- UML Profile for Schedulability, Performance and Time (SPT)
  - Current OMG standard for UML 1.4 / UML1.5
  - OMG adoption process:
    - Request For Proposals (RFP) – issued 2000
    - Initial and revised proposals – 2001, 2002
    - Finalization Task Force – makes minor corrections
- UML Profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms (QoS Profile)
  - Current status: Finalization Task Force
- UML Profile for Modeling and Analysis of Real-Time and Embedded Systems (MARTE)
  - Current status: RFP issued February 2005
### Stereotypes in UML 2.0
- A Stereotype extends an existing metaclass and enables the use of platform or domain specific terminology.
- A Stereotype may have properties (attributes), which are referred to as tag definitions.
  - when the stereotype is applied to a model element, the values of the properties are referred to as tagged values.
- A Stereotype may only generalize or specialize another Stereotype.
- According to the most recent UML 2.0 Superstructure specification (document ptc/04-10-12):
  - it is not possible to define an association between two stereotypes or between a stereotype and a metaclass.
  - such associations within profiles can be achieved in limited ways by specializing some existing associations from the reference metamodel.
  - BUT this restriction may be somehow relaxed by the Finalization Task Force, by allowing for a tagged value to refer to the name of other model elements.

### Constraints as an annotation mechanism
- A constraint is a condition expressed in natural language text or in a machine readable language for expressing semantics.
  - a constraint can be attached to more than one model element.
- A user-defined constraint is described in a specified language, whose syntax and interpretation is a tool responsibility.
  - OCL: a predefined language for writing constraints.
  - other languages are allowed (programming or natural).
- Two main rules:
  - the value specification for a constraint must evaluate to a boolean value.
  - evaluating the value specification for a constraint must not have side effects.

### Current UML SPT Profile Structure
- Current UML SPT Profile: the Performance Subprofile.

---

Murray Woodside and Dorina C Petriu
Carleton University
Performance Profile: fundamental concepts

- A Scenario defines a response path through the system, so it's the unit for which performance specifications and predictions are given.
  - however, there is no Scenario stereotype in the Performance Profile to be applied to a UML model element
  - scenario annotations are attached to its first Step instead
  - a Scenario is composed of Steps (which can be simple or composite)
  - a Step stereotype has tags that define performance specifications
  - workload intensity parameters, demands for resource usage, etc.
- Scenarios use the services of Resource instances
  - resource parameters: service policy, multiplicity, operation time
  - resource performance measure: utilization
  - quantitative resource demands given for each step
- Each scenario is executed by a Workload:
  - open workload: requests arriving at in some predetermined pattern
  - closed workload: a fixed number of active or potential users or jobs

Performance parameters and measures

- UML performance annotations define two types of information:
  - Performance parameters (inputs to a performance evaluation) describe:
    - workload
    - resource usage (measured or estimated)
    - behaviour of the program
  - Performance measures describe the performance itself:
    - response time, throughput, utilization, percentage of lost packets, etc.
    - may be given as required values
    - may be performance predictions - performance evaluation results
    - for the same measure we can have:
      - required and predicted values
      - more than one specified value (normal service, premium service)
      - more than one prediction

Specifying Time Values

- Time values can be represented by a special Value stereotype «RTtimeValue» in different formats:
  - 12:04 (time of day)
  - 5,3, ‘ms’ (time interval, unit)
  - 2000/10/27 (date)
  - Wed (day of week)
  - Sparam, ‘ms’ (parameterized value, unit)
  - ‘poisson’, 5,4, ‘sec’ (time value with a Poisson distribution)
  - ‘histogram’ 0,0.28, 1,0.44, 2,0.28, 3, ‘ms’
Specifying Performance Values

- A complex structured string with the following format

  \[
  \text{<performance-value>} ::= \text{<kind-of-value>} \cdot \text{<modifier>} \cdot \text{<time-value>}
  \]

  where:

  - \text{<kind-of-value>} ::= \text{req} | \text{assm} | \text{pred} | \text{msr}
  - \text{required, assumed, predicted, measured}
  - \text{<modifier>} ::= \text{mean} | \text{sigma} | \text{kth-mom} | \text{integer} | \text{max} | \text{percentile} | \text{dist}
  - \text{time-value} \text{ is a time value described by the RTtimeValue type}

- A single characteristic may combine more than one performance values:

  \[
  \text{<PAcharacteristic>} ::= \text{<performance-value>} \cdot \text{<performance-value>} \cdot \text{...}
  \]

  Example:

  \[
  \text{PAdemand} = (\text{pred}, \text{mean}, (20, \text{ms}))
  \]

  \[
  \text{PArespTime} = (\text{req}, \text{mean}, (1, \text{sec})) \cdot (\text{pred}, \text{mean}, S\text{RespT})
  \]

  \text{required predicted => analysis result}

Specifying Arrival Patterns

- Method for specifying standard arrival pattern values:
  - Bounded: \text{bounded}, \text{<min-interval>}, \text{<max-interval>}
  - Bursty: \text{bursty}, \text{<burst-interval>}, \text{<max.no.events>}
  - Irregular: \text{irregular}, \text{<interarrival-time>}, \text{<interarrival-time>}
  - Periodic: \text{periodic}, \text{<period>}, \text{<max-deviations>}
  - Unbounded: \text{unbounded}, \text{<probability-distribution>}

- Probability distributions supported:
  - Bernoulli, Binomial, Exponential, Gamma, Geometric, Histogram, Normal, Poisson, Uniform

- What happens when other distributions are needed?
  - tradeoff between
    - flexibility (allow users to introduce their own definitions)
    - simplicity/convenience of expression (i.e., provide pre-packaged definitions).

Performance Stereotypes (1)

<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, Stimulus,</td>
<td>PArespTime [0..1]</td>
<td>A closed workload</td>
</tr>
<tr>
<td></td>
<td>Message, Method...</td>
<td>PApopulation [0..1]</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>PAutilization [0..1]</td>
<td></td>
</tr>
<tr>
<td>«PAcontext»</td>
<td>Collaboration, CollaborationInstanceSet, ActivityGraph</td>
<td>PAintervalDelay [0..1]</td>
<td>A performance analysis context</td>
</tr>
<tr>
<td>«PAhost»</td>
<td>Classifier, Node, ClassifierRole,</td>
<td>PAutilization [0..1]</td>
<td>A deferred receive</td>
</tr>
<tr>
<td></td>
<td>Instance, Partition</td>
<td>PAmaxTime [0..1]</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>PAoccurrence [0..1]</td>
<td></td>
</tr>
<tr>
<td>«PAopenLoad»</td>
<td>Action, ActionExecution, Stimulus,</td>
<td>PArespTime [0..1]</td>
<td>An open workload</td>
</tr>
<tr>
<td></td>
<td>Message, Method...</td>
<td>PApriority [0..1]</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>PApredTime [0..1]</td>
<td></td>
</tr>
</tbody>
</table>

Performance Stereotypes (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,</td>
<td>PAutilization [0..1]</td>
<td>A passive resource</td>
</tr>
<tr>
<td></td>
<td>Instance, Partition</td>
<td>PAschedPolicy [0..1]</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>PACapacity [0..1]</td>
<td></td>
</tr>
<tr>
<td>«PAsleep»</td>
<td>Message, ActionState, Stimulus,</td>
<td>PAload [0..1]</td>
<td>A step in a scenario</td>
</tr>
<tr>
<td></td>
<td>SubactivityState</td>
<td>PAstepTime [0..1]</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>PApred [0..1]</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>PApost [0..1]</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>PAutilization [0..1]</td>
<td></td>
</tr>
</tbody>
</table>

Murray Woodside and Dorina C Petriu
Carleton University
Applying the SPT Profile

Case Study – Building Security System (BSS)
- Select key-performance use cases

Deployment Diagram (UML 1.4)

DD: Strategy and changes for UML2
- Strategy:
  - elements that are identified in behaviour diagrams by lifelines or swimlanes must be deployed
  - other resources can be identified here (e.g. Buffer)
- UML2:
  - “artifacts” that “manifest” components are deployed
  - Activity Diagram “partitions” group together actions with “some characteristic in common”
    - actions performed by an instance may be grouped in a partition
    - actions may reference operations of instances
Current SPT Limitations

- Users cannot define a delay measure between an arbitrary pair of events (an “end-to-end delay”)
  - currently, delay measures are associated to scenarios and steps (i.e., a delay is associated to a single model element)
  - if the beginning and end events are defined by different model elements, a delay should be associated to two model elements)
- Missing annotations for the size of messages passed between processes
- Network and communication delays are related to message length.
- Lack of annotations on state machine behaviours
- Lack of support for component-based software engineering
  - no Service and service-based QoS attributes are used now
  - challenge: one interface definition can be mapped to different service realizations, each of which should be modelled separately (different behaviour, resource demands, performance results, etc.)

UML Profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms (QoS Profile)
Goals of QoS Profile

- QoS Profile has a broader scope than the SPT Profile, allowing for user defined QoS and Fault-Tolerance concepts.
  - based on ISO Quality of Service Framework (1998)
  - SPT Profile: focused on schedulability and performance analysis
- QoS Profile contains:
  - QoS subprofile - extends the General Resource Model (GRM) from the SPT profile
  - QoS model library
  - Risk Assessment subprofile
  - Fault Tolerance subprofile
- Current status of the QoS Profile
  - The QoS profile was adopted by OMG in June 2004:
    - OMG document ptc/2004-06-01, June 2004
    - currently in the finalization phase, which precedes the formal adoption.

QoS Modeling Elements

- QoS Characteristic = a quantifiable aspect of QoS, defined independently of the means by which it is represented or controlled
  - quantified with some specific parameters and methods, or with other characteristics with a lower abstraction level.
  - can be grouped into categories
  - QoS Value instantiate QoS Characteristic and gives it specific values
  - QoS Dimension: a characteristics may be characterized by many values
- QoS Constraints define restrictions on QoS characteristics:
  - QoSRequired, QoS Offered, QoS Contract, CompoundConstraint
- Different QoS Levels of Execution for the same characteristics may have different constraints
- QoS Adaptation and Monitoring
  - for the transition from one execution mode to another requires
  - for the detection of faults

Catalog of QoS Categories

- Performance: makes reference to the timeliness aspects
  - QoS Characteristics included: latency and throughput
- Dependability:
  - QoS Characteristics included: availability, reliability, safety, integrity
- Security: covers protection of entities, and access to resources
- QoS Characteristics included: access control and confidentiality
- Integrity: the service provided is not the service expected (e.g., different levels of error or accuracy)
- Coherence: concurrent and temporal consistency
- Throughput
- Latency
- Efficiency: produce results with minimum resource consumption
- Demand
- Reliability
- Availability.

QoS Profile annotation process

- QoS Annotations are done in three steps:
  1. Define the QoS characteristics of interest for the analysis to be carried out in a specific domain - define template classes
  2. Define a Quality Model for the specific domain
    - the parameters of the QoS characteristic template classes specified in the first step are all resolved by bindings
  3. Annotate a UML given model - in three ways:
    - attach a note with an OCL constraint to a model element
    - connect the constrained element with an instance of a class stereotyped as <<QoSValue>> by a <<QoSContract>> dependency (it implies the creation of additional objects in the UML model)
    - stereotype the constrained model element with a QoS constraint and use AllowedValue and logicalOperator properties to reference a set of QoS Values and their relationships.
Example: Embedded Automation System

- System controlled by two cyclic control processes that are activated simultaneously and synchronized at the end of each cycle:
  - read a sample input from sensors
  - elaborate the future state
  - save the new state in memory and produce output for actuators

- Fault tolerance strategy includes:
  - fault masking during elaboration of future state:
    - hw/sw replication and voting
  - error detection during read/save operations: sw watchdog
  - error diagnosis and recovery or reconfiguration from failure

- Different non-functional requirements:
  - performance: “cycle-time at most 15 ms”
  - dependability: “mean availability at least 98%”

Example: Memory Fault Scenario

Step 1: Define QoS Characteristics

Legend
- from template library
- new template definition
**Step 2: Define Quality Model**

- Resolve all the parameters of the QoS characteristic template classes specified in Step 1 by bindings.
- Problem: cannot bind template parameters to variables or expressions, only to concrete values.

**QoS Profile: What it provides**

- it is only about defining QoS measures (required and offered)
  - very general measures... discussed later
  - object oriented structure
  - library of pre-defined measures
  - user extensible
- the QoSValue object defines the measure names
- Chatacteristic defines the value and units
- a measure must be a scalar
- thus, mean and variance of a delay are separate dimensions of the same characteristic
  - measures are defined by objects and associated with objects

**Compare measure definitions**

- SPT PAPerfMeasure has
  
  
  "(" <source-modifier> "," <type-modifier> "," <time-value> ")"

  Where:
  
  <source-modifier> := 'req' | 'assm' | 'pred' | 'msr'
  (means respectively: required, assumed, predicted, and measured)
  <type-modifier> := 'mean' | 'sigma' | 'kth-mom' , <Integer> | 'max' | 'percentile,' <real> | 'dist'
  <time-value> has Dimension slots
  a measure must be a scalar
  a "direction" of increasing quality indicating whether bigger is better or worse

- QoSCharacteristic has Dimension slots
  - units
  - statisticalQualifier, corresp to type-modifier above
  - a "direction" of increasing quality indicating whether bigger is better or worse
Strengths of QoS Profile

- Power and flexibility
  - Inheritance structure among constructs and among measures
- Precision
  - The class definitions can contain precise definitions of the measures, rather than depending on "well-known" meanings
- Aligned with ISO QoS standard (which perhaps is directed mainly to network QoS)

Limitations of QoS Profile

- Heavy weight
  - Requires a lot of special definitions to use it (overhead to the user)
- Values must be numeric
  - Cannot use variables like $N, $contextSwitch
  - Cannot use expressions for values, as provided by the Tag Value Language in SPT.
  - (but, can use expressions in constraints on values)
- Cannot express an "end-to-end" delay
  - More later...neither can SPT
- Does not express execution characteristics like branch probabilities, demands of steps.
  - Can express Open workload intensities, but not Closed

Annotation Style in the SPT and QoS Profiles

QoS Profile annotation: response time (steps 1, 2)

Step 1: use template from QoS Catalog if appropriate

- requestUnit: string
- resultUnit: string
- Unit: string

Step 2: bind template

- requestUnit: string
- resultUnit: string
- Unit: string
- (TemplateParameters (requestUnit -> ms, resultUnit -> ms))

<<QoSCharacteristic>>

<<QoSDimension>>

<<QoSDimension>>

<<QoSCharacteristic>>

context turn-around::turn-around

post resultOK: result = self.instant-of-result - self.instant-of-request

<<QoSCharacteristic>>

<<QoSDimension>>

<<QoSDimension>>

<<QoSCharacteristic>>

response-time

Murray Woodside and Dorina C Petriu
Carleton University
QoS Profile annotation: response time (step 3)

- A constraint can be attached to any model element:
  - structural: component, operation, interface, port, etc.
  - behavioural: action, message, etc.
- Convenient expression of requirements as OCL constraints
- Definition of response-time as QoSCharacteristics allows for defining attributes such as:
  - instant-of-request
  - instant-of-reply
- However, there is no easy way to:
  - identify the starting and ending events corresponding to a response time
  - define a placeholder variable for a response time result.

QoS Profile annotation: execution demand

Step 1: define new template

```plaintext
<<QoSCharacteristic>>
evaluation_time_demand
<<QoSDimension>>
+ mean: real
  { unit(Unit), statisticalQualifier(exponential) }
```

Step 2: bind template

```plaintext
<<bind>>
Unit: string
Distribution: string
```

Step 3: alternatives for execution demand

a) Attaching a constraint to a model element

```plaintext
<<QoSRequired>>
{ context response-time-inv: turn-around-value <= 35 }
```

b) Attaching a QoSValue object to a model element through a QoS dependency relationship

```plaintext
<<QoSValue>>
m1: demand
mean = 2.8
```

STP Profile annotation: response time

- response time for a scenario attached to its first step
- can express multiple values from different sources for the same measure by using qualifiers:
  - required, measured, predicted, assigned
- multiple performance value types, as defined in TVL
- can use variables and expressions, as defined in TVL
- cannot express requirements as logical expressions as in OCL.
STP Profile annotation: execution demand

- execution demand PAdemand is but one of the attribute of the stereotype <<PAstep>>
- in a sequence diagram, the stereotype <<PAstep>> can be attached to a message or to the corresponding action execution
- in an activity diagram, <<PAsupport>> can be attached to an activity
- variables and expressions can be used, as defined in TVL
- different time values can be used, as defined in TVL

```
<<PAstep>>
  { PAdemand = ("assm","mean", ($B * 0.9, ms')), ... }
```

Developing the SPT and QoS Profiles

- SPT PerfValues may be made a lightweight shorthand for a set of standardized QoSDimensions
- New concepts needed in both SPT and QoS profiles to express “end-to-end” delays
  - time intervals between two arbitrary events.
- Agreement is needed on the specification of complex timing values
  - stochastic time values
  - parametric models
    - symbolic variables (e.g., $C, $N, $B) and expressions in SPT Tag Value Language (TVL)
    - OCL cannot be used for this purpose
    - expressions: express CPU demand as $C*(log($N) + $B)
  - the QoS Profile does not have such a capability yet.

Importance of Symbolic Values and Expressions

- Parametric studies in the performance domain
  - parameters for demands and workload
  - parameters for varying the requirements
- Requirements that depend on context variables
  - required time to download is longer for a larger file
- Demands that depend on context variables
  - demand to sort depends on list length
  - demand is composed of variable quantities
  - several demands depend on just a few context variables
UML Profile for Modeling and Analysis of Real-Time and Embedded systems (MARTE)

MARTE Profile Structure

- Time and Concurrent Resources (TCR)
- Real-Time and Embedded Systems Modeling (RTEM)
- Schedulability and Performance Analysis (SPA)

Time and Concurrent Resource (TCR) Part: Requirements

- Semantics to support interpretation of behaviour in RT systems
- Support the following time models:
  - Asynchronous/Causal models
  - Synchronous/Clocked models, (not in SPT)
  - Real/Continuous time models
- Support a rich set of measures for schedulability and performance:
  - expressed by statistical measures (such as average, variance, percentile values, histograms, distributions, and deterministic values)
  - a delay measure shall be applicable to an interval bounded by a defined start event and a defined end event.
  - extensible types of measures to allow for user-defined distributions, statistics, bound definitions, etc.
  - any measure shall be expressible in multiple versions (e.g., a required value, an offered value, an estimated value, a test result value, etc.)
TCR Requirements (2)

- Provide a concrete syntax definition for time values modeling (e.g., RTtimeValue in SPT).
- Define modeling entities to specify tasking models including concurrency and scheduling mechanisms.
- Support for modeling of software deployment on heterogeneous platforms.
- Define a resource model to describe an abstract view of the hardware architecture including:
  - active/passive elements (e.g. CPU, Memories, FIFO, etc.)
  - communication media (e.g. Busses, Crossbar, etc.)
  - hierarchical models of hardware (e.g. SMP, multiprocessors, etc.).
  - not in SPT.

TCR: Synchronous Time Semantics

- Time proceeds in timeSteps (a logical clock: the duration of a step is usually supposed to be a fixed physical time).
- Several actions can be specified to occur within one timeStep:
  - see it as a clustering of behaviour to support analysis
  - the actions in one timeStep can have a sequential structure of their own which is defined in continuous time within the timeStep
  - behaviour in a timeStep can process several events (messages or signals) which were sent in a previous timeStep.
- Value is for analysis at the level of successive timeSteps.

The new RTE M Part: Motivation

- Provide support to model the following classes of RTES
  - embedded systems
  - reactive systems
  - control/command systems
  - intensive data-flow computation systems
- Ambitious goal: provide a common language for those involved in RTES development:
  - systems engineers - concerned with the overall architecture; usually make trade-offs between implementing functionality in hardware, software;
  - hardware engineers – language not meant for detailed circuit design, just for high-level description at block level
  - software engineers - language meant for detailed software development

RTE M goals

- To define a platform-independent design of a real-time and embedded system (RTES)
- To enable interoperability between different RTE development tools
- To improve communications between developers by providing a common way for modeling both hardware and software aspects of RTES and their allocation.
- To refine platform-independent designs
- To refine RTE platform-specific designs
- To enable the construction of schedulability/performance analysis models for quantitative predictions
- To be used for RTES software design in the MDA context
RTEM Requirements

- Embedded System Requirements
  - support at least static allocation of resources
  - define non-functional requirements related to:
    - real-time constraints (e.g. deadline, response time, etc.)
    - embedded constraints (e.g. power consumption, memory size)
- Behavior Requirements
  - specify computation models for data-flow and control-flow
  - support deterministic behavior modeling of RTES
- Structure Requirements
  - high-level modeling constructs for factorization of repetitive structures
  - allocation of software capabilities to an abstract view of the hardware
  - component-based modeling for RTES hardware architectures.

Schedulability and Performance Analysis (SPA) Requirements

- Harmonize the schedulability and performance analysis metamodels
- Base profile concepts on the current version of the UML 2 standard.
- Support the expression of predecessor-successor relationships between Steps/Actions, probabilities of branching, and loop counts for both performance and schedulability analysis.
- Provide support for parameterized expressions for annotations
- Support modeling of timed-utility functions which describe the cost of delay and the relative importance or urgency of responses
- Support analysis of component-based models, by attaching measures to services offered or required at interfaces/ports.
- Provide support for analyzing state machine based models.

Toward a Proposal for the Schedulability and Performance Analysis (SPA) Subprofile
Challenges

- Extend the domain model of the SPT Performance Profile
  - merge the schedulability and performance domain concepts
  - concepts: services, components, QoS at interfaces
  - Properties of OS
  - scenarios vs. state-machines
- Annotation style
  - compare SPT and QoS styles of profile (stereotypes vs. templates)
  - Take advantage of QoS profile
- Expressing and attaching performance measures
  - parametric annotations: variables, expressions, functions, late binding of values to parameters
  - end-to-end delays:
    - extend SPT's attachment of a measure to a single model element
- Openness to new user-defined measures

Domain Model Extension: Services

- Different meanings for the concept of "service"
  a) service offered by a Resource
     - processor: executes instructions
     - network: moves bytes
     - process: executes an operation
     - semaphore: ensures mutual exclusion to an operation
     - buffer: provides memory space to be used
  b) service offered by a Component at an Interface
  c) end-to-end or point-to-point service defined purely by a scenario
     - start: triggered by an event
     - end: satisfaction of a postcondition (completeness criterion)

Aspects of Defining a Service

In practice these aspects may be explicit or implicit:

- Identify the entity offering the service, and the interface used
- Define the service: what is done during the service
- Application services have a full definition
- Services by middleware and physical resources are often implied indirectly

Different Kinds of Services: a "levelled" view

Application Level:
- by objects and components, fully defined, with defined interface
- service is defined by behaviour (may return, may be end-to-end)
- can use extOp in SPT for application level services which are not defined

Middleware Level:
- services to link application objects
- entities, interfaces and service definition implied by annotations

Physical Level:
- services to execute operations or convey messages defined in application
- entities defined by deployment,
- service is defined:
  - partly by the nature of the device, and relationship (e.g., PArate)
  - partly by demands annotated on operations, messages, and by annotations to nodes describing hardware (e.g., PAhost)
**A "levelled" view (2)**

<table>
<thead>
<tr>
<th>The role of the Service</th>
<th>How use of the Service is indicated</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application Level:</td>
<td>message, call, signal, Step</td>
</tr>
<tr>
<td>by design</td>
<td>Drawn:</td>
</tr>
<tr>
<td>step</td>
<td>Step -&gt; call -&gt; extOp</td>
</tr>
<tr>
<td>Middleware Level:</td>
<td>(Services offered for scheduling, I/O and storage, memory management, and network messaging)... Implied by these operations, plus an indication of the OS and middleware</td>
</tr>
<tr>
<td>implied o'head</td>
<td>Indicated: Operation has demands and component, component has artifact, artifact has deployment that indicates physical host.</td>
</tr>
<tr>
<td>Physical Level:</td>
<td>execution of operations</td>
</tr>
</tbody>
</table>

**Services seen by Levels**

- **Application Layer:**
  - invoke app service (logical service)

- **OS/Middleware Layer:**
  - negotiate, connect (logical services)

- **Physical Layer:**
  - execute (physical)

**Characteristics of a Service**

- a service has Quality of Service attributes (requirements, measures, etc.)
- a service is described by a scenario, which has a workload
- a service may be associated with an interface, offered by a component
- a service has a trigger which may be a signal or message arriving to an interface (port)
- a service may have one or more end points; a criterion for completion may be a postcondition related to the end points
- progress points may be defined along the service with QoS attributes attached.

**Defining QoS of a Service**

- An application service is defined by a scenario.
  - QoS = delay (call-return or end-to-end)
  - with progress points, a set of delays
- A middleware service may be required by a Step
  - in principle we could identify m/w functions (too much detail?)
  - then, QoS = delay (calculated in model)
- Processor services are specified as host demands of steps
  - also Disk devices, other host devices
  - QoS = actual delay including contention, to give the service
  - or: define demands as operations, and an operation time.
  - QoS = operation delay
- higher level QoS depends on lower level
- QoS depends on lower contention: not a free-standing property
Information to support Service analysis

Two aspects:
- Identify the requests or demand levels for the Service.
- Define the content of the service so that its QoS can be calculated.
  + Give the delay directly as a parameter.
  + Give the demands generated by the service, for other services, so the delay can be calculated.
  + Give the internal structure, from which the demands can be calculated.

Demands for Services

- Application: "workload" data for the scenario.
- OS and Middleware: uses of services are implied by:
  + Message sending (ORB or protocol overhead).
  + File I/O.
  + Knowledge (outside the UML spec) of how the application uses the m/w, perhaps.

Thus frequency of use can be worked out from frequency of Step execution.

- Processor: the host demands of Steps define the demand for processor operations. Other devices are implied similarly.

Describing the internals of a Service

- An application service is defined by a scenario.
  + Steps describe details to support calculation.
- Middleware services are not described?
  + Maybe more in the MARTE profile on OS and m/w.
  + Demands per service operation could be stored.
- Processor services are implied by hosting of Steps/Components.
  + Also Disk devices, other host devices.
  + The demand is the internals.

Different Scenario Structures and "end-to-end" QoS ... beyond call-return delay

Delays for complex combinations of events:
- A progress point may be defined by a condition.
  + Each point has its own QoS spec.

Response object:
- toProgress1: delay, fraction, etc.
- toProgress2: delay...
- toProgress3: delay...
- toEnd: delay...

Murray Woodside and Dorina C Petriu
Carleton University
Problems with end-to-end QoS

- It is difficult to identify the events in the scenario context, for the start/end or progress points.
  - events have no graphical notation standard
  - events are associated to graphical entities as:
    - send or receive of a message, or
    - start of end of an executionSpecification (= Step)

- a Delay stereotype could perhaps be associated with multiple messages and executionSpecifications
  - not allowed?
- or, each message and executionSpecification could be stereotyped, and then these are associated.

PA Domain Model Extension: QoS at interfaces

- SPT Performance Subprofile is not using the stereotype Service
- Component and services in UML
  - an interface of a component contains a set of operations that define services provided or required by the component
  - an asynchronous service may be triggered by a signal arriving to a port
- Possible solution for MARTE:
  - apply the stereotype Service to an Operation defined at an interface of a component or to a Signal arriving to a port
- Issues to be considered:
  - an operation defined at an interface can be mapped to multiple operation realizations (e.g., due to polymorphism)
  - in a performance model, each service realization has to be represented separately, as it has different behaviour, resource demands, etc.
  - reporting of the performance results: not per service definition, but per service realization.

PA Domain Model Extension: Scenarios Vs State Machines

- Minimal changes are required to support attaching workload information to state machines
  - Steps on transitions, branching probabilities
  - must also connect behaviour through between state machines, when one invokes another

BUT,
- State machines are defined by class
  - different instances of an object may have different demands and probabilities
  - this will probably limit the use of SM definitions
- A scenario may use only part of the possible behaviour of a SM...
  - again, one SM would support multiple scenarios

SM Annotation: a Possible Solution

- Problem: one SM may require multiple annotations for different scenarios or instances
- Solution: where needed, define specializations purely for the purpose of annotation
  - the base class provides defaults, specializations provide overrides
  - identical SM classes, not necessarily used in the design
  - one specialization for each scenario
  - where only one instance or one scenario, this is not needed

NEGATIVES:
- problem of designating the relationship
- design clutter
Merging the Schedulability and Performance Subprofiles

- Many concepts are shared:
  - PAstep and SAaction both describe a unit of computation with processor demands, within a scenario
  - PAworkload and SAtrigger define intensity of load
  - sequence of steps has the same meaning
  - response time and deadline
- Differences are in emphasis, and usage of the information
  - PAstep has more general kinds of demands
    - not just CPU (PAhostDemand), but also "extOp" demands
    - not just min and max, but statistics (mean, variance), and distributions
  - SAaction has more time attributes

Schedulability and Performance: Common Aspects

Using the usual terms for performance...

- Step in a scenario (SAaction) (often called a task)
- Scenario (scheduling job, associated to an SAtrigger)
- Precedence relationships (sequence, branch, merge, fork, join, loop) (often called a task graph)
- Demands (SAworstCase) (execution time)
- Stimulus and response, response time (SAtrigger)
- Host Processor (execution engine)

Differences (SA / PA)

- The “task” (SAaction) is more central in SA than PAstep is in PA:
  - task is scheduled (vs process in PA, and in many systems)
  - task may be implemented as a process, for OS scheduling
  - jobs tend to be a single task, or a simple sequence
    - BUT: more complex jobs are becoming common (convergence)
  - Triggers tend to be more deterministic in SA (periodic, or min inter-arrival time)
  - CPU demands tend to be more deterministic in SA and more constrained (by design)
  - Passive resource demands are important in both
  - Deadlines are central in SA
    - BUT: they are important in PA too, just harder to deal with!
    - AND: soft deadlines are more common in SA.
    - convergence here too

Differences (SA/PA) Suggest a Combination...

SA parameters and attributes can be expressed within PA

- execution time bound is a max CPU demand
- periodic trigger is an arrival process
- used resources can be explicitly acquired/released
- deadline is a particular kind of response time requirement (required max value)
- some differences would require augmenting PA, as noted.

HOWEVER, SA users could see this as clutter...
### Detailed Differences: PAspec vs SAction

**CPU demand**, expressed as
- mean and perhaps variance
- distribution (exponential, Erlang, uniform, ..., histogram) with appropriate parameters (1 to n parameters)
- best-case, average-case and worst-case estimates of the mean
**Demand counts** for external operations identified by name
- other devices/services not in the software model
**Delay** (like a think time)
**Response time**, interval (between repetitions)

**CPU demand**
- worst-case completion time: the CPU time used in the analysis
- Execution attributes
  - priority, host, "isAtomic"
  - usedResources (passive resources)
- Constraints on execution
  - release time, ready time
  - abs deadline, relative deadline, "laxity" (= hard/soft deadline type
  - Achieved performance
    - delay time (waiting)
    - blocking time
    - pre-empted time

### Union of PA & SA

- **OpenWorkload/Trigger**
  - occurrencePattern
  - responseTime
  - priority (PA only)
  - isSchedulable (SA only)
- **ClosedWorkload**
- **Scenario/SchedulingJob**
  - scheduler (SA only)
- **Step/Action**
  - union of attributes seems wisest.

### Detailed Differences: PScenario vs SScheduling job

- **PAworkload** is a stereotype on the first step, giving parameters
  - open or closed
- **Precendence relationships: no differences**

**Scheduling is a host property**
**Calculated outputs:**
- from associated first Step, responseTime attribute

- **SAttriget stereotypes the stimulus, with property occurrencePattern**
  - nominally only open

**Scheduler is specified here. Outputs are spread over:**
- associated first SAction properties
- SAresponse stereotype on first Action (slack)
- SAttriget stereotype on stimulus (isSchedulable, endToEndTime)

### Union (2)

- **ProcessingResource/SAengine**
  - scheduling attributes (SA), rate, context switch cost,
    - utilization
- **SchedulableResource**
  - a process etc.
  - capacity (multi threaded)
  - priority (PA)
  - means a realignment of PA: split PAresource into schedulable and logical
- **LogicalResource**
  - a semaphore, buffer pool etc
  - units requested
  - capacity, time to acquire and release, consumable (?)
Expressing Performance Measures

- Complex types
  - expressed by statistical measures (such as average, variance, percentile values, histograms, distributions, and deterministic values)
  - extensible types of measures to allow for user-defined distributions, statistics, bound definitions, etc.
- Any measure shall be expressible in multiple versions (e.g., a required value, an offered value, an estimated value, a test result value, etc.)

Parametric Annotations

- Expressions and functions with variables of complex types
  - e.g., schedulability utility functions
- Late binding of values to parameters
- Ranges of values (e.g. for loops)
- Language issues: TVL versus OCL

Attaching performance measures

- In SPT and QoS Profile each performance measure is attached to a single model element
  - response time to a scenario
  - execution time demand to a step
  - utilization to a resource
- MARTE requirement: a delay measure shall be applicable to an interval bounded by a defined start event and a defined end event
  - why this is a challenge:
    - a UML stereotype can extend a single UML model element, so a delay measure cannot be defined as a stereotype

Openness to new user-defined measures

- Tradeoff between the simplicity of annotations (as in SPT) versus the flexibility to add user-defined measures (as in QoS Profile)
- Question - any difference between:
  - defining a new type for a given measure (e.g., a new distribution for execution demand), and
  - defining a new measure (e.g., a new definition for system availability)?
Conclusions

- Proposals in response to MARTE RFP are under preparation
  - Need to collect ideas from the software performance community
- How to merge the current Schedulability and Performance sub-profiles
- How to harmonize MARTE performance annotations with the existing QoS Profile
- Challenges in using the Performance Profile in the context of MDA
  - how to annotate UML models at different levels of abstraction:
    - platform-independent models (PIM)
    - platform-specific models (PSM)
    - what constitutes a platform: middleware, OS, hardware?
  - how to derive a performance model, which is inherently instance-based and platform dependent, by combining the PIM of an application with platform model(s).