Performance Analysis Based on the UML SPT Profile

Prof. Dorina C. Petriu
Carleton University
Department of Systems and Computer Engineering
Ottawa, Canada, K1S 5B6
http://www.sce.carleton.ca/faculty/petriu.html

Software Performance Engineering

- Software Performance Engineering (SPE) (Smith, 1990):
  - integrate performance evaluation into the software development process from the early stages throughout the whole life-cycle
  - requires the construction and analysis of performance models
- Why SPE is not frequently applied in practice:
  - cognitive gap between the software development domain and the performance analysis domain
  - pressure for “shorter time to market” leaves no time for SPE
- Performance analysis of UML models
  - OMG’s UML Profile for Schedulability, Performance and Time (SPT)
  - STP Profile sparked research efforts to:
    - automate the derivation of performance models from UML models
    - bridge the gap between software design and performance analysis

Layered Queueing Networks

Layered Queueing Network (LQN) model

http://www.sce.carleton.ca/rads/lqn/lqn-documentation

- LQN is an extension of QN
  - models both software tasks (rectangles) and hardware devices (circles)
  - represents nested services (a server is also a client to other servers)
  - software components have entries corresponding to different services
  - arcs represent service requests (synchronous and asynchronous)
  - multi-servers used to model components with internal concurrency
- What can we get from the LQN solver
  - Service time (mean, variance) including nested services
  - Waiting time
  - Throughput
  - Utilization
  - Probability of missing a deadline (obtained only by simulation)
Types of LQN Requests

a) Synchronous

Client

\[ s_1, s_2 \]

Server

\[ s_1, s_2 \]

b) Asynchronous

Client

\[ s_1, s_2 \]

Server

\[ s_1, s_2 \]

c) Forwarding

Client

\[ s_1, s_2 \]

Server1

\[ s_1, s_2 \]

Server2

\[ s_1, s_2 \]

\[ s_1, s_2 \]

LQN extensions: activities, fork/join

Current UML SPT Profile: the Performance Subprofile
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 entities
  - 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 Profile: the domain model

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
  \[ <\text{performance-value}> ::= \langle <\text{kind-of-value}> \cdot <\text{modifier}> \cdot <\text{time-value}> \rangle \]
  where:
  - \(<\text{kind-of-value}> ::= \text{'req'} | \text{'assm'} | \text{'pred'} | \text{'msr'}\)  -- required, assumed, predicted, measured
  - \(<\text{modifier}> ::= \text{'mean'} | \text{'sigma'} | \text{'kth-mom'} | \text{<integer>} | \text{'max'} | \text{'percentile'} | \text{'<Real>'} | \text{'dist'}\)
  - \(<\text{time-value}> \) is a time value described by the RTtimeValue type

- A single characteristic may combine more than one performance values:
  \(<\text{PAcharacteristic}> ::= <\text{performance-value}> [ <\text{performance-Value}> ]^* \)

Example:
- \(<\text{PAdemand} = (\text{'pred'}, \text{'mean'}, (20, \text{'ms'})) \rangle \)
- \(<\text{PArespTime} = (\text{'req'}, \text{mean}, (1, \text{'sec'})) (\text{'pred'}, \text{mean}, \$\text{RespT}) \rangle \)

Specifying Arrival Patterns

- Method for specifying standard arrival pattern values:
  - Bounded: ‘bounded’, \(<\text{min-interval}>, <\text{max-interval}>\)
  - Bursty: ‘bursty’, \(<\text{burst-interval}>, <\text{max.no.events}>\)
  - Irregular: ‘irregular’, \(<\text{interarrival-time}>, <\text{interarrival-time}>\)
  - Periodic: ‘periodic’, \(<\text{period}>, <\text{max-deviation}>\)
  - Unbounded: ‘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) and 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,</td>
<td>PAdemand [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>PAextDelay [0..1]</td>
<td>PAthroughput [0..1]</td>
<td></td>
</tr>
<tr>
<td>«PApcontext»</td>
<td>Collaboration, CollaborationInstanceSet, ActivityGraph</td>
<td>PAutilization [0..*]</td>
<td>A performance analysis context</td>
</tr>
<tr>
<td>«PAhost»</td>
<td>Classifier, Node, ClassifierRole, Instance, Partition</td>
<td>PAutilization [0..*]</td>
<td>A deferred receive</td>
</tr>
<tr>
<td></td>
<td>PAcount [0..1]</td>
<td>PAschedPolicy [0..1]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PAarate [0..1]</td>
<td>PAisContext [0..1]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PAarrivalRange [0..1]</td>
<td>PAisPreemptible [0..1]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PAisResponseTime [0..1]</td>
<td>PAisThroughput [0..1]</td>
<td></td>
</tr>
<tr>
<td>«PAopenLoad»</td>
<td>Action, ActionExecution,</td>
<td>PAdemand [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>PAoccurrence [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>«PAsource»</td>
<td>Classifier, Node, ClassifierRole, Instance, Partition</td>
<td>PAutilization [0..*]</td>
<td>A passive resource</td>
</tr>
<tr>
<td></td>
<td>PAschedPolicy [0..1]</td>
<td>PAschedCapacity [0..1]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PAarrivalRange [0..1]</td>
<td>PAarrivalTime [0..1]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PAisThroughput [0..1]</td>
<td>PAisMaxDelay [0..1]</td>
<td></td>
</tr>
<tr>
<td>«PAstep»</td>
<td>Message, ActionState,</td>
<td>PAarrivalTime [0..1]</td>
<td>A step in a scenario</td>
</tr>
<tr>
<td></td>
<td>Stimulus, SubactivityState</td>
<td>PAschedPolicy [0..1]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PAschedCapacity [0..1]</td>
<td>PAarrivalTime [0..1]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PAarrivalRange [0..1]</td>
<td>PAarrivalTime [0..1]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PAisMaxDelay [0..1]</td>
<td>PAarrivalTime [0..1]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PAisThroughput [0..1]</td>
<td>PAarrivalTime [0..1]</td>
<td></td>
</tr>
</tbody>
</table>
Transformation of UML Models to LQN Performance Models: High Level View

Tool interoperability

- Forward path (in black) - implemented
- Backward path (in gray) - not done yet

Well-formed annotated UML model

- key use cases described by representative scenarios
  - frequently executed, have performance constraints
- resources used by each scenario
  - resource types: active or passive, physical or logical, hardware or software
  - examples: processor, disk, process, software server, lock, buffer
  - quantitative resource demands for each scenario step
    - how much, how many times?
- workload intensity for each scenario
  - open workload: arrival rate of requests for the scenario
  - closed workload: number of simultaneous users

Case study: Document Exchange System (DES)
"Retrieve" Scenario with performance annotations

Client RetrieveI SOMTOD

- request document
- accept request
- read request
- update logfile
- write to logfile
- parse request
- get document
- read from disk
- send document
- recycle thread
- receive document

<<PAstep>>
{PAdemand=('msrd', 'mean', (220/$cpuS, 'ms'))}

<<PAstep>>
{PAdemand=('msrd', 'mean', (1.30 + 130/$cpuS, 'ms'))}

<<PAclosed Load>>
{Papopulation = $Nusers}

<<PAstep>>
{PAdemand=('asmd', 'mean', (0.5, 'ms'))
PAextOp=('net1', 1)
PArespTime=('req', 'mean', (1, 'sec'), ('pred', 'mean', $RespT)}

<<PAstep>>
{PAdemand=('asmd', 'mean', (1.5, 'ms'))}

<<PAstep>>
{PAdemand=('msrd', 'mean', (35/$cpuS, 'ms'))}

<<PAstep>>
{PAdemand=('msrd', 'mean', (25/$cpuS, 'ms'))}

<<PAstep>>
{PAdemand=('msrd', 'mean', ($scdC/$cpuS, 'ms'))
PAextOp=('net2', $DocS')}

<<PAstep>>
{PAdemand=('msrd', 'mean', (0.70, 'ms'))
PAextOp=('writeDisk', $RP')}

<<PAstep>>
{PAdemand=('msrd', 'mean', ($gcdC/$cpuS, 'ms'))}

Direct UML to LQN Transformation: our first approach

1. Generate LQN model structure (tasks, devices and their inter-connections) from:
   - UML model of the high-level software architecture
   - UML deployment diagram

2. Generate LQN detailed elements (entries, phases, activities and their parameters) from:
   - UML models of key scenarios with performance annotations
   - Scenarios can be represented in UML by the software designers as:
     - interaction diagrams
     - activity diagrams

UML→LQN Transformation Algorithm

1. Generate the LQN model structure
   1.1. determine LQN software tasks from high-level components
   1.2. determine LQN hardware devices from deployment diagram

2. Generate LQN entries, phases, activities from scenarios
   2.1. for each scenario, process the activity (interaction) diagram(s)
     2.1.1. match inter-component communication style from pattern with messages between components from the activity diagram
     2.1.2. divide the activity diagram into sub-graphs corresponding to different LQN entries, phases, and activities;
            create the respective LQN elements and compute their parameters;
   2.2. merge the LQN submodels corresponding to different scenarios;

3. Traverse the LQN graph and write out textual model description.

UML→LQN Transformation: Mapping the structure
Client Server Pattern

- **Structure (fig.a)** - the participants and their relationship
- **Behaviour**
  - Variant 1 (fig.b): Synchronous communication style - the client sends the request and remains blocked until the sender replies
  - Variant 2 (fig.b): Asynchronous communication style - the client sends the request and continues its work; later on it will accept the server's reply.

Client Server Pattern with blocking client: Mapping the Behavior

For each subset of scenario steps mapped to a LQN phase or activity, compute the execution time $S$:

$$S = \sum_{i=1}^{n} r_i s_i$$

where $r_i$ = number of repetitions and $s_i$ = execution time of step $i$.

Client Server Pattern with non-blocking client: Mapping the Behavior

Forwarding Server Chain Pattern

- **a) FwdServerChain Collaboration**
- **b) Behaviour of the FwdServerChain pattern**
Forwarding Pattern: Mapping the Behavior

<table>
<thead>
<tr>
<th>Client</th>
<th>FwdServer</th>
<th>RplServer</th>
</tr>
</thead>
<tbody>
<tr>
<td>e1, ph1</td>
<td>e2, ph1</td>
<td>e3, ph2</td>
</tr>
<tr>
<td>wait</td>
<td>work</td>
<td>complete</td>
</tr>
<tr>
<td>request</td>
<td>service and forward</td>
<td>service (opt)</td>
</tr>
<tr>
<td>Client is blocked waiting for reply</td>
<td>continue work</td>
<td>continue work</td>
</tr>
</tbody>
</table>

DES Case Study: generating the LQN structure

b) Mapping physical resources (processors and I/O devices) to LQN devices

effect of communication network

For every message transmitted over the network, insert a "network task" that represents the network delay suffered by the message.

In the case of synchronous communication, both the request and the reply suffer a network delay.

For each subset of scenario steps mapped to a LQN phase or activity, compute the execution time $S_i$:

$S = \sum_{i=1}^{n} r_i s_i$

where $r_i$ = number of repetitions

$s_i$ = execution time of step $i$. 
Performance Model Verification and Validation

Verification and validation

- Model verification and validation cover different aspects of assessing the “goodness” of a performance model
  - how close the model is to the real system?
- Verification: ensuring that the model is correctly built
  - consistent application of transformation rules produce correct models
  - similar to debugging
- Validation: ensuring that the model produces performance results close to that observed in the real system. Validate assumptions about system behaviour, input parameters and distributions, output results by:
  - using expert intuition
  - comparing with real system measurements
  - comparing with known theoretical results.

DES case study: model validation

a) UML-based analysis and design
b) Implement the system by using known reusable frameworks (ACE and JAWS)
c) Measure the system in a networked environment
d) Annotate the UML model with measured resource demands
e) Generate the LQN model automatically from the UML specification
f) Solve the LQN model
g) Compare LQN model results with overall performance measurements obtained from the real system
h) use the LQN model to gain some insights into the DES performance.
### Sensitivity to network delays for short messages

Sensitivity of the LQN model to network delay for 5 KB message size

![Graph showing sensitivity to network delays for short messages](image)

### Sensitivity to I/O time for long messages

Sensitivity of the LQN model to I/O time for 50 KB message size

![Graph showing sensitivity to I/O time for long messages](image)

### Sensitivity to network delays for long messages

Sensitivity of the LQN model to network delay for 50 KB message size

![Graph showing sensitivity to network delays for long messages](image)

### Using the LQN model: Effect of Multithreading

Effect of multithreading

![Graph showing effect of multithreading](image)
Using the LQN model: Effect of caching

![Graph showing effect of caching](image)

Using the LQN model for analyzing and improving system performance

Case Study – Building Security System (BSS)

- Use Cases

![Deployment Diagram](image)

Deployment Diagram
Sequence Diagram for Video Acquire/Store Scenario

Video Acquire/Store Scenario: top-level AD

Composite activity procOneImage
Sequence Diagram for Access Control Scenario

Using the Model: Improvements to BSS
- Base case system capacity: 20 cameras
  - Problem: software bottleneck at the buffers
- Adding software resources:
  - 4 Buffers and 2 StoreProc threads
  - 40 cameras supported, performance improvement 100%
  - hardware bottleneck at the processor
- Replicating the processor:
  - Dual Application CPU
  - 50 cameras supported, performance improvement 150%
- Increasing software concurrency
  - asynchronous messages - moving phase1 call to phase2
  - 100 cameras supported, performance improvement 400%
The PUMA project

- PERFORMANCE FROM UNIFIED MODEL ANALYSIS
  - NSERC strategic project: www.sce.carleton.ca/rads/puma/
- Goal: to develop a unified approach for building performance models from scenario-based software specifications
- PUMA proposed approach to support scenario-based performance engineering:

PUMA project highlights

- Scenario input based on the UML Profile for Schedulability, Performance and Time.
- Build models from the UML design specifications
- Performance modeling using existing tools.
- Design evaluation by assisted exploration of the design space and feedback to the designer as:
- identified hot spots
- software resource analysis
- design improvement suggestions.

Core Scenario Model

- Requirements:
  - Simpler than the UML model
  - Must contain all the data for generating performance models
- Reflects closely the elements of the Performance Profile
  - different kind of resources
  - scenarios and scenario steps and their resource demands
  - workload
- Challenges for the transformation from UML to CSM:
  - Extract only the UML model elements important for building the performance model
  - Process multiple UML diagrams and multiple scenarios
  - Recognize whether the UML model is incomplete/inconsistent from the performance annotation viewpoint
  - Generate a correct XML file conform to the CSM Schema
Generating Different Performance Models

- Challenges for the transformation from CSM to different performance models:
  - Realizing the mapping between CSM and the chosen performance model
    - different degrees of abstraction between input and output
    - generate the performance model structure
    - compute the performance model parameters
    - identify cases in which the chosen performance model cannot express some features of the system under study
  - Automatic or semi-automatic transformation?
    - designer guidance may be necessary
  - Flexibility of the transformation process
    - inflexible: mapping hard-wired in the transformation code
    - flexible: the performance analyst or software developer may be allowed to set some transformation rules.

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
  - under improvements by the Finalization Task Force, preceding 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:
  - QoS required, QoS offered, QoS contract, combined constraint
- 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

- Three steps are required:
  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.

Step 1: QoS Characteristics (a)

Efficiency category

Latency category

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.
### Step 1: QoS Characteristics (b)

- **Fault**
  - QoSCharacteristic: fault
  - Availability: availability
  - Fault-Tolerance: fault-tolerance

- **Availability**
  - QoSCharacteristic: availability
  - Reconfigurability: reconfigurability

- **Fault-Tolerance**
  - QoSCharacteristic: fault-tolerance
  - Recov: recoverability

### Step 2: Quality Model

#### Quality Model

- **Resource-Service-Time**
  - PA4ASResource-service-time

- **Alarm-Latency**
  - PA4ASAlarm-latency

- **Overhead**
  - PA4ASOverhead

- **Physical-Fault**
  - PA4ASPhysical-fault

### Step 3: Annotated Sequence Diagram

#### Annotated Sequence Diagram

- **System**
  - PC: Plant

- **Communication**
  - ACF2: Automation
  - AMF2: Automation
  - AC_ELAB: Elaboration
  - AC_MEM2: Memory

- **Failure**
  - PF: PA4ASPhysical-fault

- **Elaboration Function**
  - ELAB: Elaboration Function
  - MEM2: Memory Function

- **Resource**
  - GRMresource

- **QoS Models**
  - QoSDimension

- **QoSDimension**
  - QoSValue

- **QoSOffered**
  - QoSRequired

#### Extensions to QoS Profile

- **New concepts needed in both profiles to express time intervals between two arbitrary events.**
- **QoS and SPT profiles will need to reach a common agreement on the specification of complex timing values.**
  - QoS Profile lacks expression of stochastic time value
  - Another open problem: parametric models
    - fixed values for model parameters are not enough
    - SPT Profile supports symbolic variables and expressions in its Tag Value Language (TVL)
      - OCL cannot be used for this purpose
      - the QoS Profile does not have such a capability yet.
Upgrading the SPT Profile

Current SPT Limitations

- Missing the ability for the user to define a delay measure between an arbitrary pair of events.
  - 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 concepts are used now.

SPT Profile on Components and Interfaces

- Proposed extensions:
  - Apply the stereotype Resource to components
  - Apply the stereotype Service to interfaces corresponding to services provided or required by components
- Discussion:
  - So far the Performance Profile does not use the stereotype Service, but the General Resource Model of SPT (from which the Performance Profile inherits) defines resources and services, so the extension is straightforward.
  - An interface definition can be mapped to multiple interface 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.
    - Impact reporting of the results - not per service definition, but per service realization

Offered QoS

- In the present SPT Profile version, we can define required and offered performance characteristics (such as response time, throughput, etc.) at the scenario level only.
- Once we introduce provided and required services for components, the next step is to attach QoS characteristics to these services.
- Challenge due to the nature of systems with limited resources: "offered QoS" depends not only on the component capacity (which include all the underlying resources) but also on the workload offered to the component by its environment.
- Proposed extension: we should be able to express the QoS offered by a component as a function of the component’s workload.
- When a component offers multiple services, the service mix plays also a role.
**Required QoS**

- Proposed extension: express the "required QoS" as a function of component’s workload, as well.
  - a) required QoS has to be matched with the QoS offered by other components, which is workload dependent
  - b) the QoS required by a certain resource has to be taken into account when determining the QoS offered by the same resource for different workloads.

- **Workload:**
  - In the present Performance Profile version, the workload level is attached to the first step of a scenario.
  - With the proposed extensions, the workload has to be computed for every resource/service.

- **QoS contracts:**
  - More research in the performance domain is needed to find simple yet expressive ways of specifying/negotiating workload dependent QoS contracts based on workload dependent offered and required QoS.

---

**Conclusions**

- Upgrading the STP Profile for UML 2.0
  - OMG is in the process of issuing an new RFP for SPTv2

- Opportunities to both generalize and simplify the SPT profile by merging the Schedulability and Performance sub-profiles
  - the same Scenario/Step structure underlies the analysis
  - in general, the same performance measures are used

- Harmonize the SPT Profile with the QoS Profile

- Challenges in using the Performance Profile in the context of MDA
  - how to build UML models at different levels of abstraction:
    - platform-independent models (PIM) and platform-dependent (PSM)
  - how to combine PIM of an application with the model of a platform to generate performance models which are inherently instance-based and platform dependent.