Runtime Management of Quality Specification for QoS-Aware Components

Miguel A. de Miguel
Escuela Técnica Superior de Telecomunicaciones
Universidad Politécnica de Madrid
mmiguel@dit.upm.es

M. Teresa Higuera
Facultad Informática, Universidad Complutense de Madrid
mthiguer@dacya.ucm.es
Contents

1. QoS Requirements and Components
2. UML QoS Specification
3. QoS Component Framework
   3.1 Integration of QoS specifications in component framework
4. Conclusions
1. Extra-Functional Requirements

- **Generic service:**
  \[ \sqrt{x} \rightarrow y \]

- **Functional Requirement:**
  \[ y^2 \equiv x \]

- **Extra-functional requirement:**
  - **Accuracy:** \[ |y^2 - x| \]
  - **Response time:** \( t \) seconds
  - **Precision of \( x \):** \([\text{double, float}]\)
1. QoS Requirements

- Extra-functional requirements related with the quality of a service or function

- In the previous case:
  - QoS characteristics: accuracy, time response, precision

- Other QoS characteristics:
  - Robustness, efficiency

- Time properties are quality characteristic
  - Response time, deadlines:
    - Can be hard or soft
  - Jitter . . .
1. Example: QoS on a Multitarget-Radar

**Antenna Capacity**
- Antenna1
- Antenna2
- Antenna3

**Antenna Accuracy**
- maxError

**Multitarget Radar**

**Radar Human Interface**

**Target Scalability**
- maxNumberOfTargets

**Trajectory Accuracy**
- singleTrajectoryAccuracy
- mixingDifferentTrajectories

**Trajectory Latency**
- maxUpdateDelay
- minUpdateDelay

**Whole Space Availability**
- availabilityOfAllSpaces
1. QoS on a Multitarget-Radar

- **Antennas**
  - Sample accuracy
    - Maximum errors
1. QoS on a Multitarget-Radar

◆ Antennas
  - Sample accuracy
    - Maximum errors
  - Capacity
    - Maximum number of samples
    - Maximum number of targets
  - Availability
    - Mean time between failures
    - Mean time to repair
  - Latency
    - Worst case response time
1. QoS on a Multitarget-Radar

- **Multi-Radar**
  - Accuracy of single trajectory
    - Maximum interpolation error

- **Antennas**
  - Sample accuracy
    - Maximum errors
  - Capacity
    - Maximum number of samples
    - Maximum number of targets
  - Availability
    - Mean time between failures
    - Mean time to repair
  - Latency
    - Worst case response time
1. QoS on a Multitarget-Radar

- **Multi-Radar**
  - Accuracy of simple trajectory
    - Maximum interpolation error
  - Accuracy of multiple trajectory
    - Probability of mixing targets
  - Scalability
    - Maximum number of targets
  - Availability
    - Space availability
    - Maximum space non-available
  - Latency
    - Worst response time of target representation

- **Antennas**
  - Sample accuracy
    - Maximum errors
  - Capacity
    - Maximum number of samples
    - Maximum number of targets
  - Availability
    - Mean time between failures
    - Mean time to repair
  - Latency
    - Worst case response time
2. UML QoS Specification (1)

- **QoS Modeling Framework** includes a modeling language for the QoS-Aware models
  - It includes the modeling elements of a QoS modeling language and their relations
- Defines the abstract syntax of this QoS modeling language
  - A MOF Metammodel for QoS modeling language
2. UML QoS Specification (2)

- **UML QoS Profile:**
  - UML extensions for the representation of QoS-Aware models in UML
  - Integration of concepts that describe QoS Modeling Framework in UML notation

- **QoS Profile Model**
2. UML QoS Specification (3)

- Mapping from UML + Profile to QoS
- Abstract Syntax of QoS Modeling Framework
  - Specification of semantics of UML + Profile models
  - Automatic construction of QoS abstract models from UML models

```plaintext
QoSCharacteristicsSpt: QoSCharacteristic

stereotype
extended

QoSCharacteristics: Class

ownedAttribute
dimensions: Property [*]
templateParameter

parameters: ParametrableElement [*]

QoSCharacteristicsTgt: QoSCharacteristic

invariant={context QoSCharacteristicSpt self.invariant}
dimensionsTgt: QoSDimension [{context dimensions self.size()}]=
{context dimensions self->iterate(
d : QoSDimension; acc : Bag(QoSDimension)=Bag{} | let nd : QoSDimension=QoSDimension.make(d.direction,
d.statisticalQuantifier,d.unit) in
acc->including(nd)).oclAsType(Bag(QoSDimension))}

parametersTgt: QoSParameter [{context parameters self.size()}]=
{context parameters self->iterate(
p : ParametrableElement; acc : Bag(QoSParameter)=Bag{} |
let np : QoSParameter in acc->
including(np)).oclAsType(Bag(QoSParameter))}

Parameter
Type

QoSCharacteristicsTgt: QoSCharacteristic

invariant={context QoSCharacteristicSpt self.invariant}
DimensionOf

Quantiifier
dimensionsTgt: QoSDimension [{context dimensions self.size()}]=
{context dimensions self->iterate(
d : QoSDimension; acc : Bag(QoSDimension)=Bag{} |
let nd : QoSDimension=QoSDimension.make(d.direction,
d.statisticalQuantifier,d.unit) in
acc->including(nd)).oclAsType(Bag(QoSDimension))}
```
3. Component Infrastructure

- Container based component architecture:
  - Component external and internal interfaces
  - Component descriptor and generator of container
3. Integration QoS Metamodel in Container (1)

- QoS Container architectures and UML mapping rules integrates both approaches.

- Mapping rules provide support for:
  - Profile 2 QoS model transformation.
    - The UML + QoS Profiles models are represented in terms of QoS abstract syntax
  - Repository management in Java:
    - MOF, JMI and EMF support the automatic generation of Java code for the repository.
    - The QoS models generate code for the instantiation
  - XML storage of QoS models.
    - MOF, XMI defines the rules for the representation in XML of models and EMF is the implementation we use.
3. Integration QoS Metamodel in Container (2)

MOF Metamodel

QoS Metamodel

XML Schema

Java QoS Classes

UML+QoS Profile Model

SLQ Model

XMI data

Java QoS Object Instances

Mapping

QoSCharacteristic factory = QoSCharacteristicFactory.eINSTANCE;
QoSCharacteristic availability = factory.createQoSCharacteristic();
QoSContext cont_ava = factory.createQoSContext();
3. QoS-Aware Component Infrastructure

- **Component Container** integrates the QoS model repository for its business component
- **QoS Manager** in the container includes basic operation for the specifications management:
  - constraints monitoring,
  - negotiation of constraints
4. Conclusions

- **Metamodels** defines a repository for the management of QoS concepts
- **UML Profile** is a tool for the integration of QoS specification in UML tools
- **Mapping** from profile to metamodel is an approach to reuse UML modelling information
- **Container** provides the basic infrastructures for the automatic integration of QoS concepts in component frameworks
- **QoS Managers** in the container reuse the modelling information to support basic QoS operations
  - Negotiation
  - Monitoring
  - Adaptation