

Quality of Service Engineering with UML and MDA

Torben Weis
weis@ivs.tu-berlin.de

Andreas Ulbrich
ulbi@ivs.tu-berlin.de

Prof. Dr. Kurt Geihs
geihs@ivs.tu-berlin.de

Intelligent Networks and Management of Distributed Systems
Berlin University of Technology

© Torben Weis, Andreas Ulbrich, and Kurt Geihs



Overview

- Introduction
- QoS-Concepts
- Modeling QoS-aware Applications
- QoS-aware Middleware
- Modeling & Middleware

© Torben Weis, Andreas Ulbrich, and Kurt Geihs



Overview

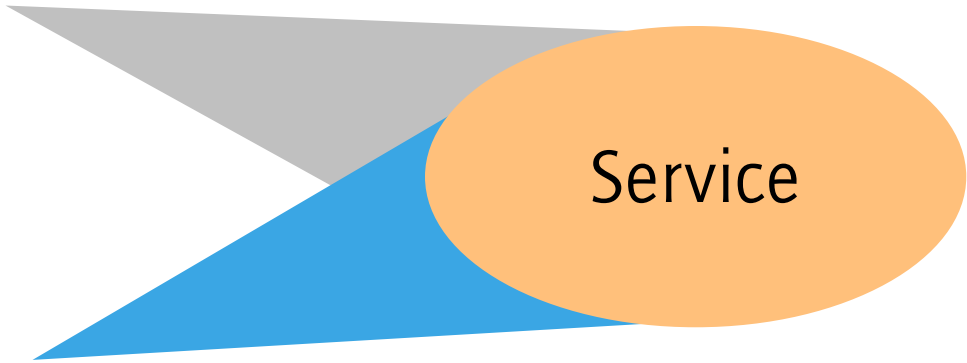
- Introduction
- QoS-Concepts
- Modeling QoS-aware Applications
- QoS-aware Middleware
- Modeling & Middleware

© Torben Weis, Andreas Ulbrich, and Kurt Geihs

Quality of Service (QoS)

What? *stock ticker, route finder, equation solver, …*

functional properties



non-functional properties

→ Quality of Service

How? *response time, throughput, precision, availability, security, …*

© Torben Weis, Andreas Ulbrich, and Kurt Geihs

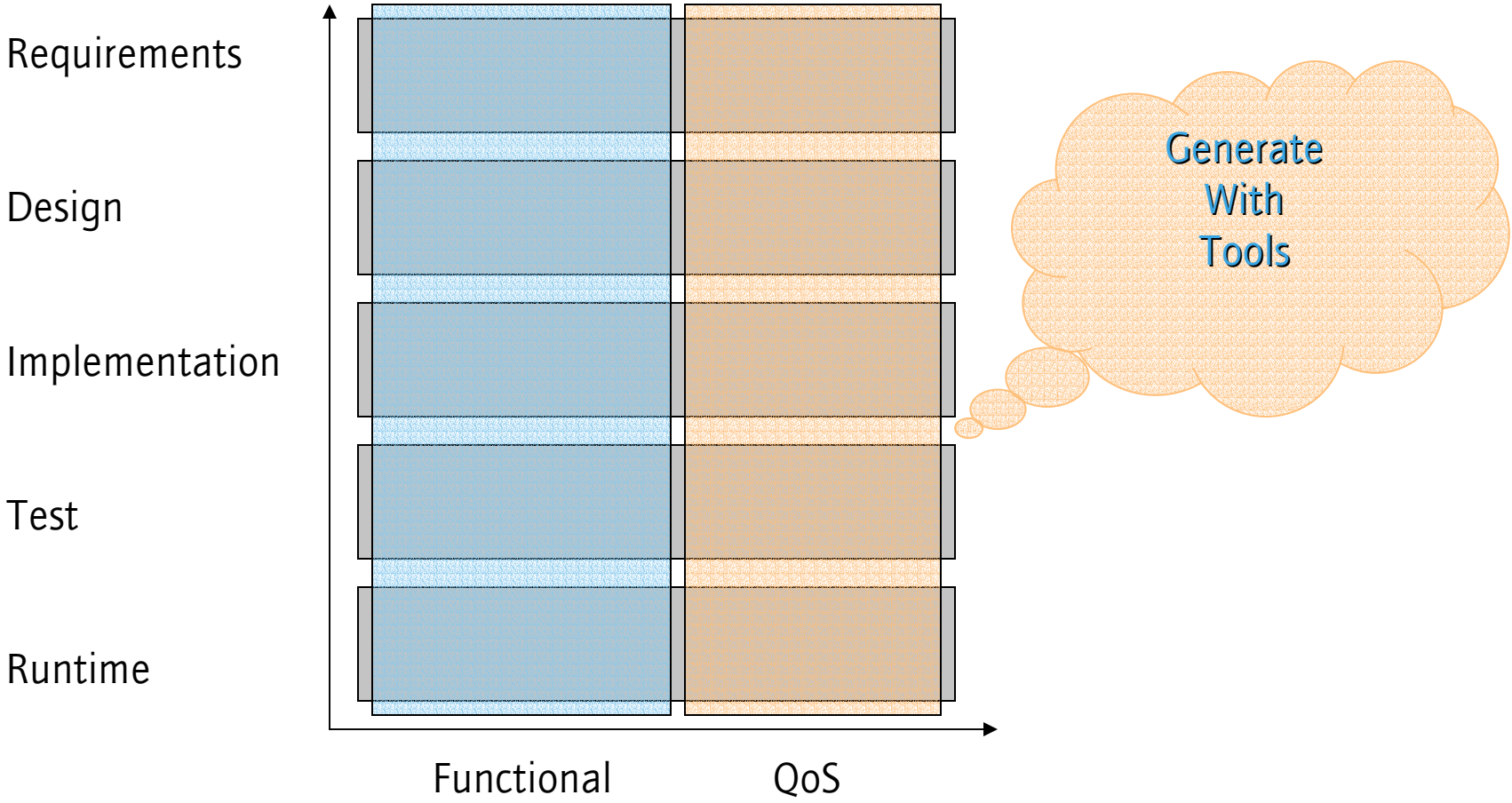
QoS – A Software-Engineering Problem

- Complexity
  - Requires expert knowledge
    - QoS-Terminology
    - QoS-Specification
    - QoS-Negotiation
    - QoS-Mechanisms
    - QoS-Monitoring
    - Billing / E-Payment
    - Middleware
  - ⇒ QoS is complicated



© Torben Weis, Andreas Ulbrich, and Kurt Geihs

Separation of Concerns (1/4)



© Torben Weis, Andreas Ulbrich, and Kurt Geihs

Separation of Concerns (2/4)

- Separation is (almost) possible for distribution
  "Distribution Transparency"
  Middleware generates
    - Stubs
    - Skeletons
- Separation is more difficult for QoS
  Cross-Cutting!
  Example: Repsone-Time with Replication
    - Shielding the development of business logic from replication related issues is very difficult
  Example: Authentification
    - The user name is determined by QoS-mechanisms but essential for the business logic

Full distribution transparency is impossible.
Partial Failures!

© Torben Weis, Andreas Ulbrich, and Kurt Geihs

Separation of Concerns (3/4)

- Aspect-oriented Programming (AOP)
  - Possibility to separate business logic from QoS-related code
  - Several drawbacks

    Requires non-trivial programming language extension

    Synchronization of QoS-aspects and business logic …

    QoS-aspects are (by default) not reusable

      – Aspects depend on the component with which they are woven

    Aspect-separation seems sometimes unnatural

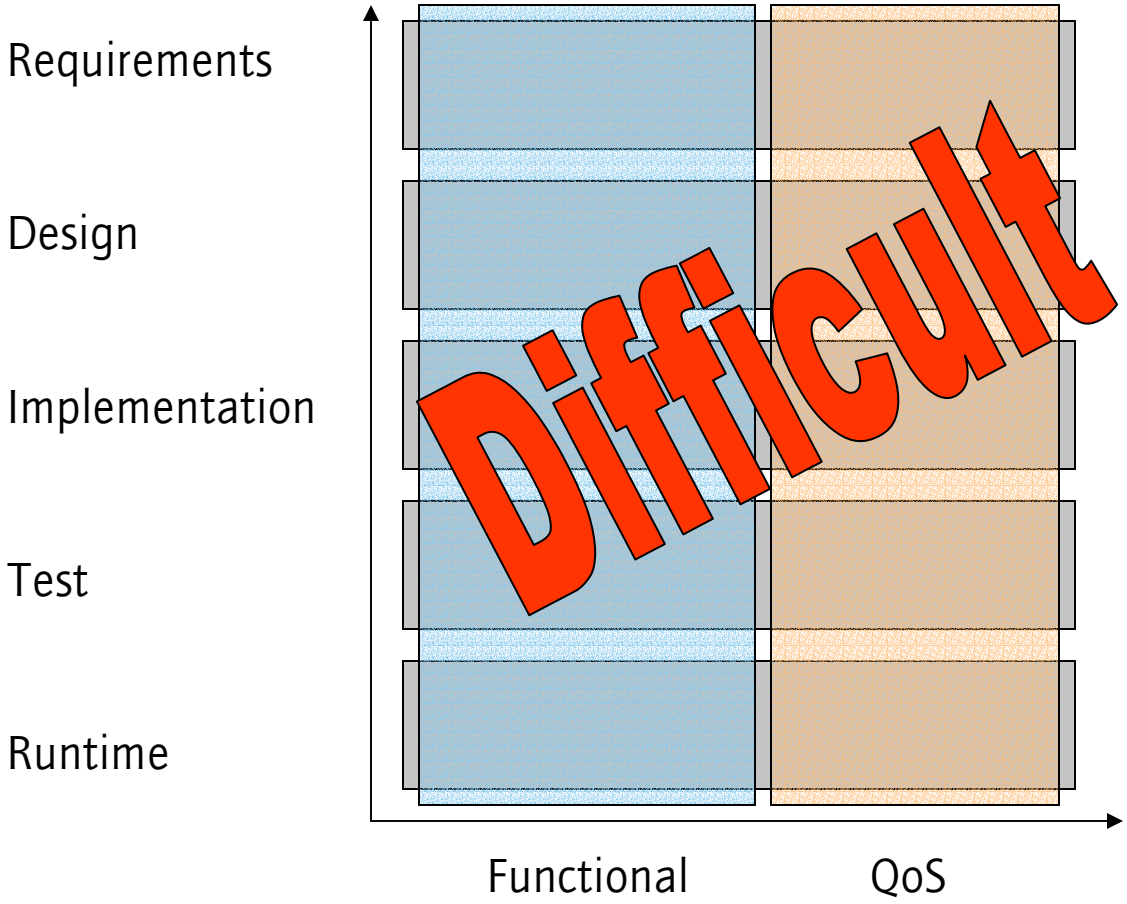
      – Example: Replication

    Classification into business logic / QoS-related sometimes fuzzy

      – Example: Authentication

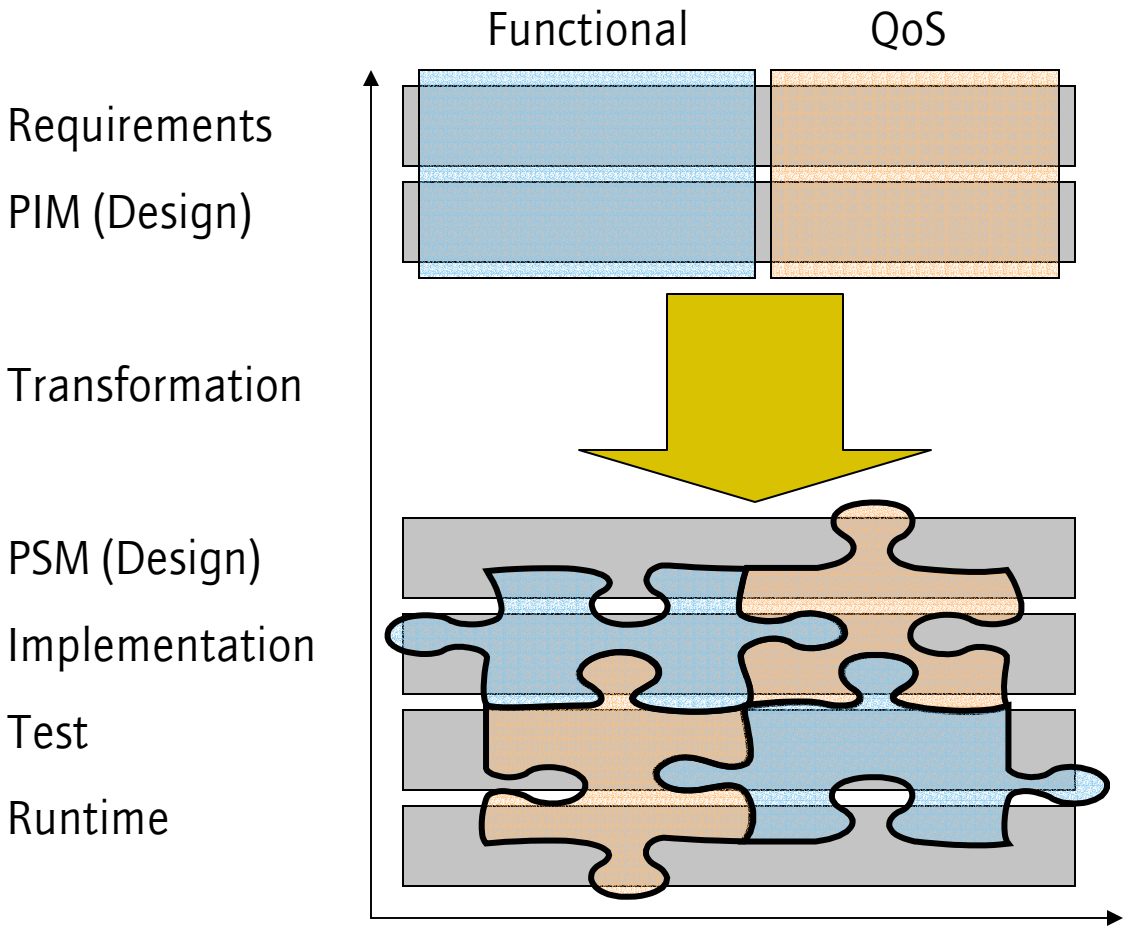
© Torben Weis, Andreas Ulbrich, and Kurt Geihs

Horizontal Separation



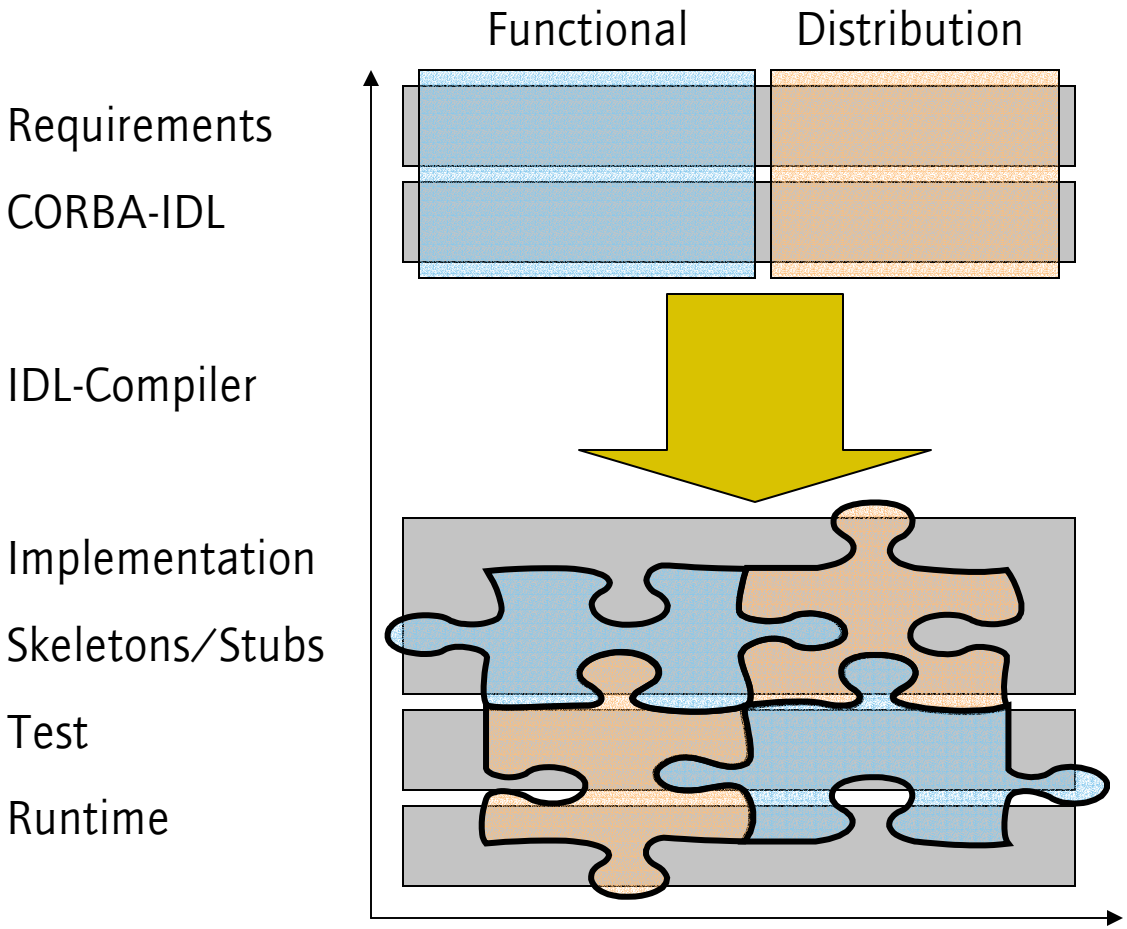
© Torben Weis, Andreas Ulbrich, and Kurt Geihs

Vertical Separation



© Torben Weis, Andreas Ulbrich, and Kurt Geihs

Seems familiar … ?



© Torben Weis, Andreas Ulbrich, and Kurt Geihs



Overview

- Introduction
- QoS-Concepts
- Modeling QoS-aware Applications
- QoS-aware Middleware
- Modeling & Middleware

© Torben Weis, Andreas Ulbrich, and Kurt Geihs



QoS-Concepts

- QoS-Basics
- QoS-Categories
- QoS-Agreements
- QoS-Contracts

© Torben Weis, Andreas Ulbrich, and Kurt Geihs

QoS ⇔ Non-Functional (1/3)

- Traditional QoS
  - Network centric
    - Bandwidth
    - Jitter
    - Round-trip time
  - Processing Centric
    - Performance / Response-time
    - Hard real-time
    - Soft real-time
    - Availability

Video/Audio streams
Block transfers

Important for X11 window system

Required for industrial automation and military

Useful for business applications.
More statistical

© Torben Weis, Andreas Ulbrich, and Kurt Geihs

QoS ⇔ Non-Functional (2/3)

- Non-functional aspects
  - Security

    Authentification

    Authorization

    Encryption

    Anonymity
  - Concurrency

    Control maximal concurrent calls at a time
  - Licensing

    Is the caller entitled to make this call?
  - Billing

© Torben Weis, Andreas Ulbrich, and Kurt Geihs

QoS ⇔ Non-Functional (3/3)

- Similarities between QoS and non-functional aspects
  - Specification of desired QoS/non-functional properties
  - Negotiation
  - Provisioning via specialized mechanisms
  - Mechanisms are injected in the message path
  - Extensions of the middleware are required
  - Monitoring
  - Logging
  - Adaptation

© Torben Weis, Andreas Ulbrich, and Kurt Geihs



QoS Terminology

QoS category	identifies a specific QoS concern (e.g. availability)
QoS parameter	quantifiable values which denote the state of a QoS characteristic (e.g. mean time to failure)
QoS agreement	result of a negotiation about QoS requirements
QoS mechanism	engineering artifacts (behavior and data structures) needed to implement QoS management
QoS framework	collection of interfaces, mechanisms, tools, services etc. to create a QoS-enabled infrastructure

© Torben Weis, Andreas Ulbrich, and Kurt Geihs

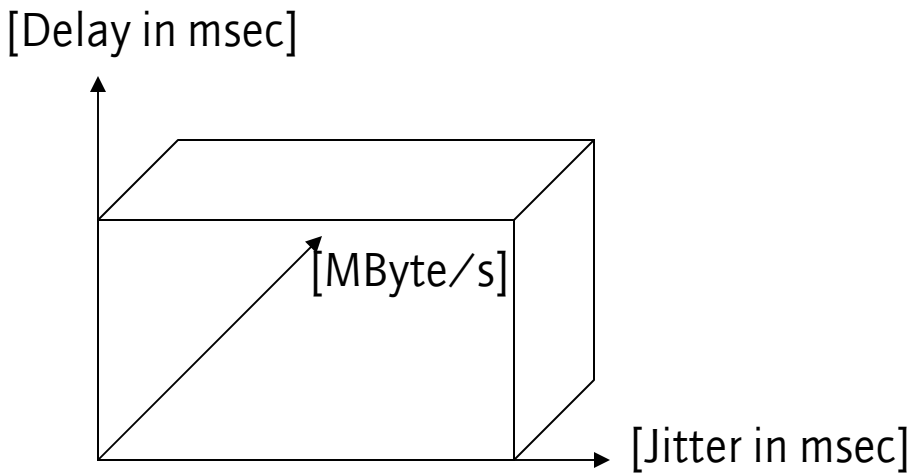
QoS-Concepts

- QoS-Basics
- QoS-Categories
- QoS-Agreements
- QoS-Contracts

© Torben Weis, Andreas Ulbrich, and Kurt Geihs

QoS-Categories

- A QoS-category has a name that implies its semantics
  - Bandwidth, Performance, Availability, …
- A QoS-category spans a multi-dimensional vector space
  - ⇒ QoS-Categories feature QoS-Dimensions



© Torben Weis, Andreas Ulbrich, and Kurt Geihs

QoS-Dimension

- A QoS-Dimension consists of
  - Storage-Type
    int, double, enumeration
  - Direction
    Up/Down
  - Physical unit
    Seconds, MByte, Percent, MByte/Second, Amount

Required for computation

Required for negotiation

Required for scaling
sec = 1000 msec

Required for formal correctness

© Torben Weis, Andreas Ulbrich, and Kurt Geihs

Example: Video Streaming

Category Throughput

{

⇑ Bandwidth : float [MByte/sec]

⇓ Jitter : float [Percent]

⇓ Delay : int [msec]

⇓ Packet Drop : float [Amount/min]

}

© Torben Weis, Andreas Ulbrich, and Kurt Geihs

Example: E-Shop

Enum AlgorithmKind {

    DES, Blowfish, …

}

Category Security {

    ⇑ KeyLength : int [Amount]

    * Algorithm : AlgorithmKind [None]

}

© Torben Weis, Andreas Ulbrich, and Kurt Geihs

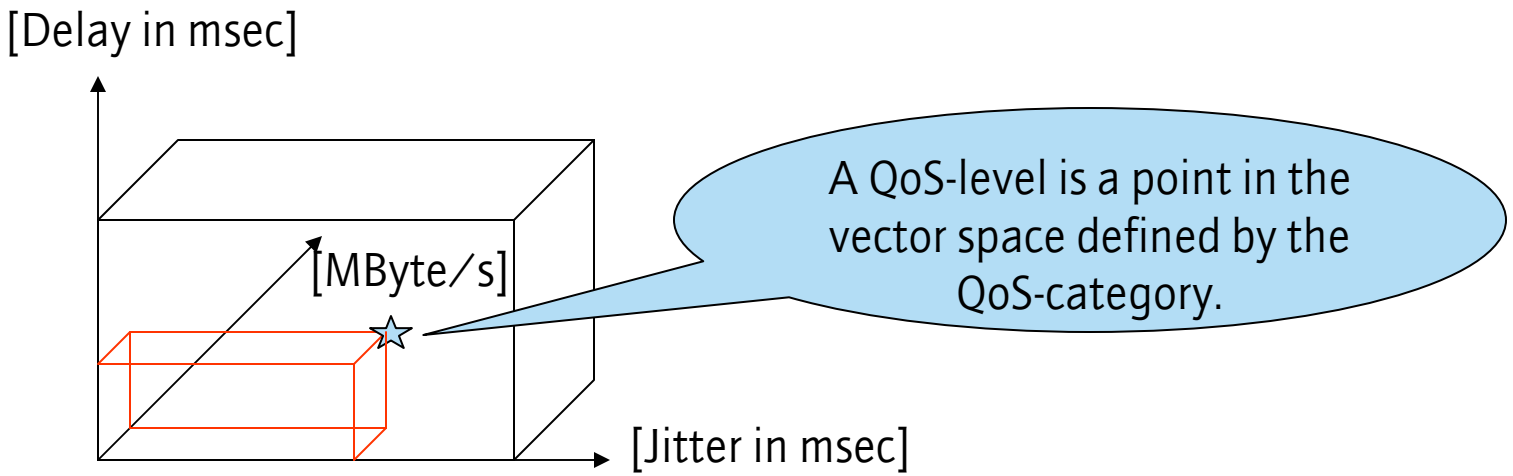
QoS-Concepts

- QoS-Basics
- QoS-Categories
- QoS-Agreements
- QoS-Contracts

© Torben Weis, Andreas Ulbrich, and Kurt Geihs

QoS-Level (1/2)

- QoS-categories can be treated as classes
- QoS-dimensions can be treated as attributes
- A QoS-level is an instance of a QoS-category
  - One value (QoS-Parameter) for each QoS-Dimension



© Torben Weis, Andreas Ulbrich, and Kurt Geihs

QoS-Level (2/2)

Category Throughput

{

⇑ Bandwidth : float [MByte/sec]

⇓ Jitter : int [msec]

⇓ Delay : int [msec]

⇓ Packet Drop : float [Amount/min]

}

QoSLevel : Throughput

{

Bandwidth = 0,1 MByte/sec

Jitter = 5ms

Delay = 30 ms

PacketDrop = 10 / min

}

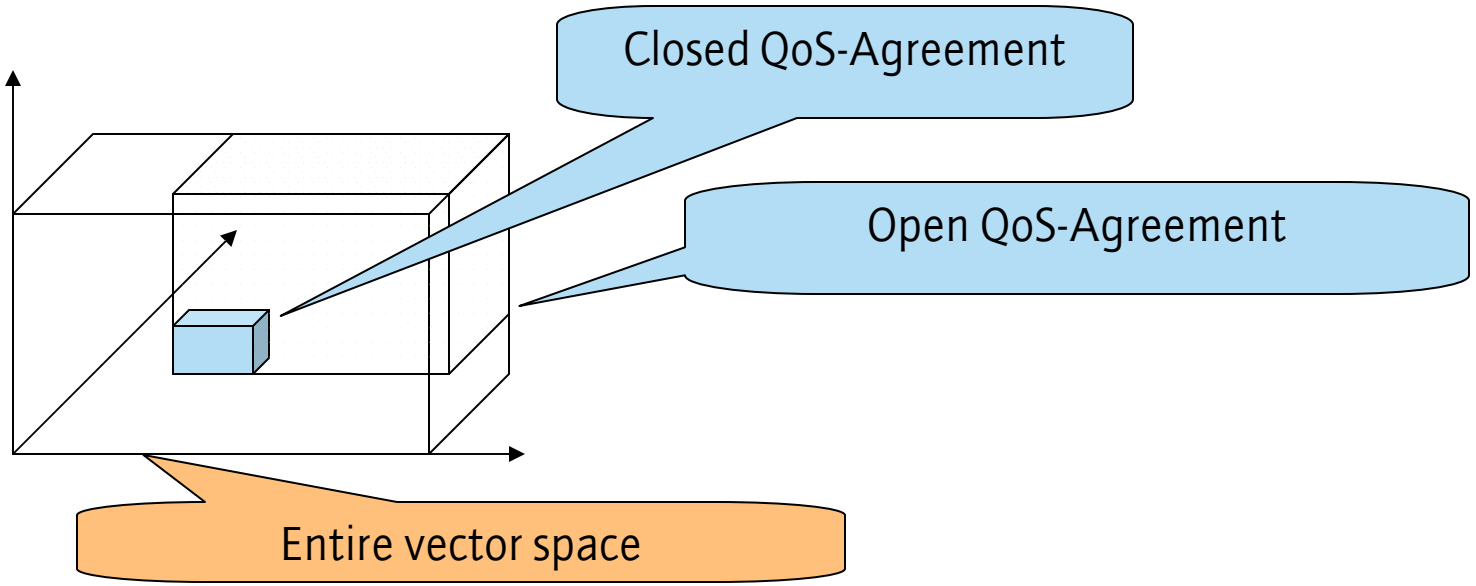
© Torben Weis, Andreas Ulbrich, and Kurt Geihs

QoS-Agreement (1/2)

> A QoS-agreement defines a subspace

> Each QoS-level inside this subspace complies with the agreement

> Every other QoS-level violates the agreement



© Torben Weis, Andreas Ulbrich, and Kurt Geihs

QoS-Agreement (2/2)

- A QoS-agreements consists of a set of constraints
- Each constraint is associated with one or multiple QoS-dimensions
- Examples
  - 100KB/Sec <= Bandwidth <= 200KB/Sec
  - Delay { percentile 10 < 20ms
    percentile 50 < 30ms
    percentile 100 < 40ms }

© Torben Weis, Andreas Ulbrich, and Kurt Geihs

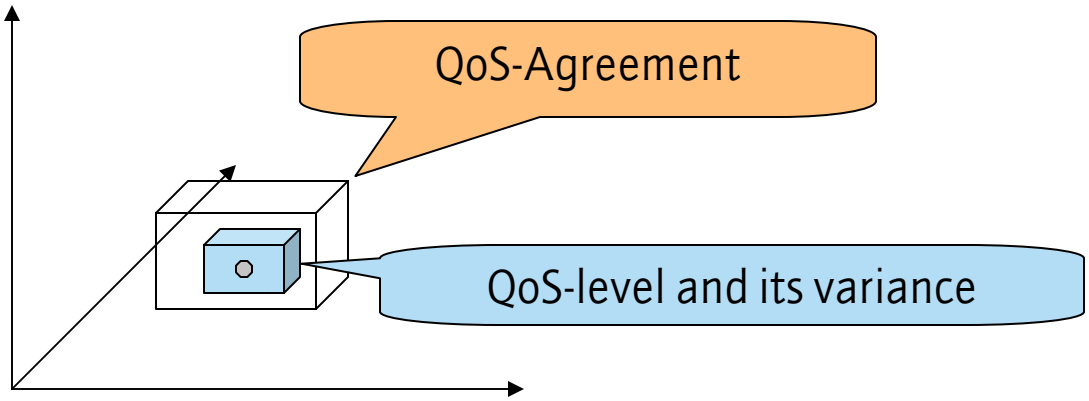
Monitoring QoS-Agreements (1/2)

- QoS-levels are monitored during runtime
- Their compliance with the QoS-agreement is checked
- Monitoring = Measuring
- Measurements are never precise
- $\Rightarrow$ Monitored QoS-levels are subject to variance

© Torben Weis, Andreas Ulbrich, and Kurt Geihs

Monitoring QoS-Agreements (2/2)

> QoS-Agreements must consider the variance implied by the measurement



© Torben Weis, Andreas Ulbrich, and Kurt Geihs

QoS-Concepts

- QoS-Basics
- QoS-Categories
- QoS-Agreements
- QoS-Contracts

© Torben Weis, Andreas Ulbrich, and Kurt Geihs

Contracts

1. Syntactic Contracts
   - > Interfaces, data-types
2. Behavioral Contracts
   - > Pre/post conditions
   - > Invariants
   - > Invocation Protocols
3. Synchronization Contracts
   - > Path-Expressions
4. QoS-Contracts

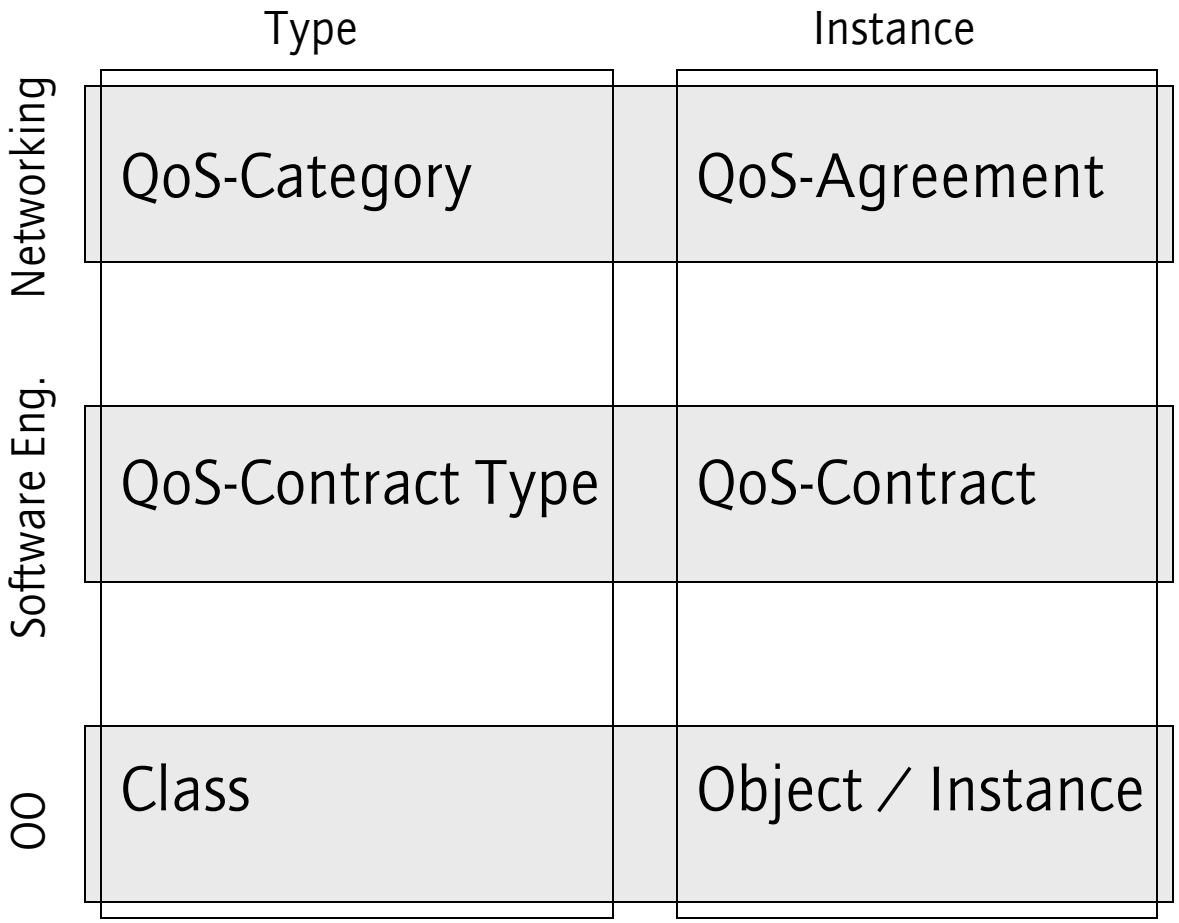
A. Beugnard, J.-M. Jézéquel, N. Plouzeau, and D. Watkins
Making components contract aware
IEEE Computer, 13(7), July 1999

Terminology (1/3)

- QoS-categories and QoS-agreements
  - "Conceptual"
  - Originate in the networking world
- QoS-contracts
  - "Computational"
  - Originates in the software engineering world
  - Can be bound to interfaces of object oriented software systems
  - Can be bound to tasks
  - Can be efficiently negotiated and monitored
  - Used to configure the QoS-framework and the QoS-mechanisms

© Torben Weis, Andreas Ulbrich, and Kurt Geihs

Terminology (2/3)



© Torben Weis, Andreas Ulbrich, and Kurt Geihs

QoS-Contract Specification (1/3)

- A QoS-contract type is an aggregation of QoS-dimensions
- Simple approach
  - A QoS-contract is an aggregation of values - one for each dimension
  - `Bandwidth = 100KB/s`
  - Shortcoming: QoS-contract = QoS-level (too weak)
- Very complex approach
  - A QoS-contract is an aggregation of constraints (formulas)
  - `100KB/S < Bandwidth < 200KB/S`
  - At least one constraint for each QoS-dimension
  - Shortcomings

    QoS-contracts are exchanged between processes.
    Parsing and evaluating the formulas is time consuming.

    Optimizations for every possible formula is difficult

© Torben Weis, Andreas Ulbrich, and Kurt Geihs

QoS-Contract Specification (2/3)

- The QML Approach
  - Support a fixed set of statistical characterizations …

    `Percentile`

    `Mean`

    `Variance`

    `Frequency`
  - … combined with comparison operators

    `>= , <= , > , < , ==`
  - Shortcomings

    Set of statistical characterizations is fixed

    One constraint can cover only one QoS-dimension

    What about categories that have interdependent QoS-dimensions?

© Torben Weis, Andreas Ulbrich, and Kurt Geihs

QoS-Contract Specification (3/3)

- Middle course
  - Hard-code the formula at design time and parameterize it
  - `a=100KB/S,b=200KB/S`, implied: `a < Bandwidth < b`
  - Pros

    No parsing of formulas at runtime

    QoS-contracts are just sets of values

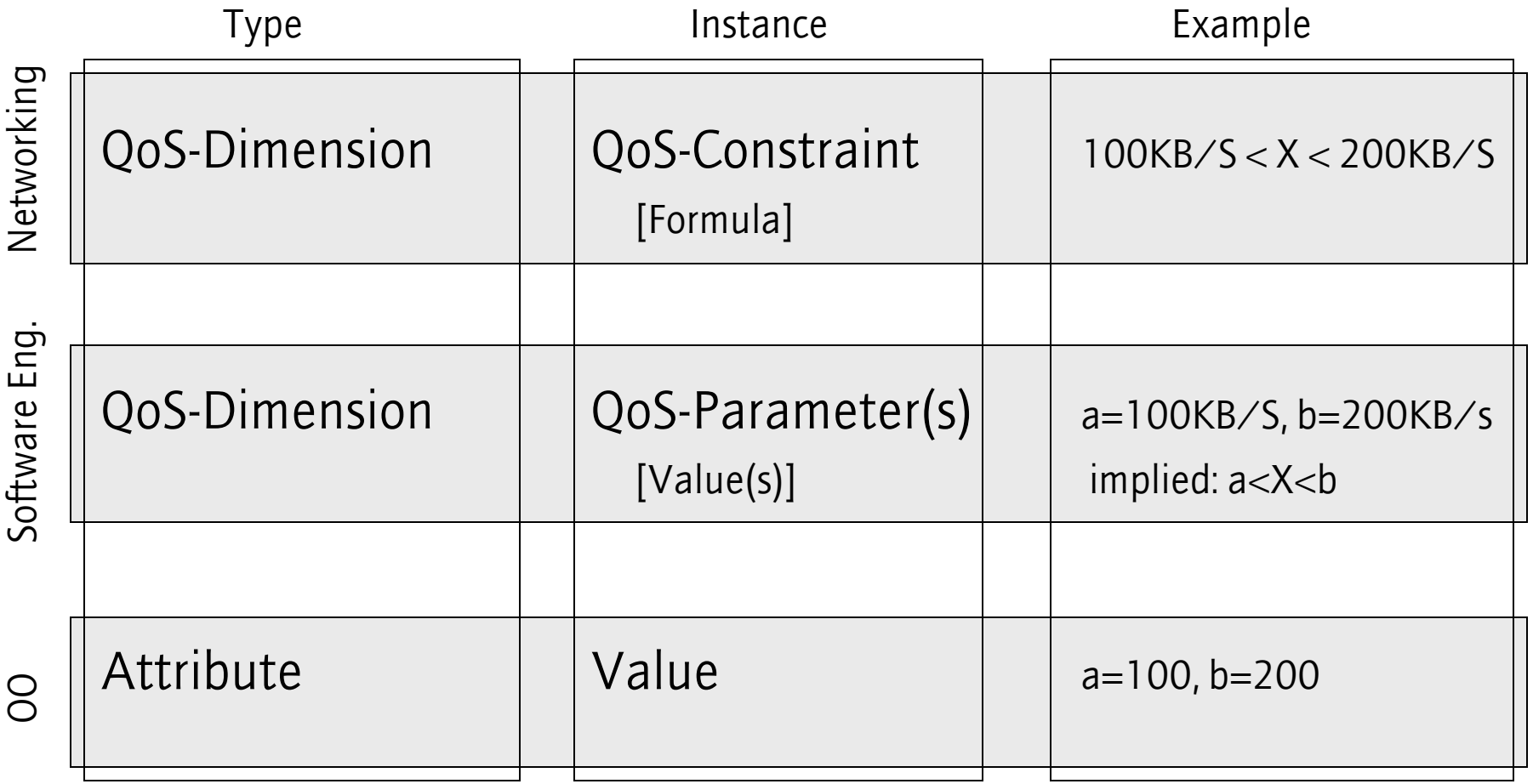
    Optimizations are easier to deal with

    More powerful than the simple approach
  - Cons

    Possible constraint formulas are limited at runtime

© Torben Weis, Andreas Ulbrich, and Kurt Geihs

Terminology (3/3)



© Torben Weis, Andreas Ulbrich, and Kurt Geihs



Overview

- Introduction
- QoS-Concepts
- Modeling QoS-aware Applications
- QoS-aware Middleware
- Modeling & Middleware

© Torben Weis, Andreas Ulbrich, and Kurt Geihs

Modeling QoS-aware Applications - Overview

- Motivation
- Model-driven Development
- QoS-support in platform independent models (PIM)
  - Components
  - QoS-contracts
  - Binding QoS-contracts to interfaces and tasks
  - Contract dependencies
- Model Transformation
- QoS-support in platform specific models (PSM)

© Torben Weis, Andreas Ulbrich, and Kurt Geihs

Motivation

- So far we know how to specify QoS-contracts
- How to build applications that adhere to these contracts?
- Three different techniques
  1. Formal Verification
  2. Testing
  3. Constructive Approach

© Torben Weis, Andreas Ulbrich, and Kurt Geihs

1. Formal Verification

- Pros
  - A complete verification covers all possible cases!
- Cons
  - Verification can only tell you “yes” or “no”
    Happens *after* design/implementation
  - Efficient automated verification is usually not possible (sigh)
  - Formal verification requires extra knowledge and time
  - Formal verification for real world applications is still a challenge
  - Verifications are conducted based on some model
    Does the model reflect the actual implementation?
    Who keeps code, model, and formal verification in sync?

© Torben Weis, Andreas Ulbrich, and Kurt Geihs



2. Testing

- Pros
  - Tests are conducted anyway
- Cons
  - Testing can only tell you “so far so good”
  - Testing happens *after* the implementation
  - Testing of large distributed systems is expensive
    Hardware
    Administration
  - Some QoS-categories are hard to test
    For example: availability

© Torben Weis, Andreas Ulbrich, and Kurt Geihs

3. Constructive Approach (1/2)

- Idea: Generate implementation (skeleton) and middleware configuration that adhere to the specified QoS-contracts
- Pros
  - Actively supports the development process
  - Expert knowledge can be encapsulated in the generators
  - Very useful if the same QoS-contract types are used in different projects
- Cons
  - Only partial implementations can be generated this way
  - One million ways of "messing it up"
  - Causes additional overhead for new QoS-contract types

© Torben Weis, Andreas Ulbrich, and Kurt Geihs

3. Constructive Approach (2/2)

- Requirement: Simple problem description, complex realization
- Example1: Distributed Objects
  - IDL Specification is simple
  - Ever looked inside the generated CORBA stubs and skeletons?
- Example 2: Quality of Service
  - Boss says: "We need ~100ms response times."
  - Developers cancel their holidays ☺

© Torben Weis, Andreas Ulbrich, and Kurt Geihs

Modeling QoS-aware Applications - Overview

- Motivation
- Model-driven Development
- QoS-support in platform independent models (PIM)
  - Components
  - QoS-contracts
  - Binding QoS-contracts to interfaces and tasks
  - Contract dependencies
- Model Transformation
- QoS-support in platform specific models (PSM)

© Torben Weis, Andreas Ulbrich, and Kurt Geihs



Model-driven Development

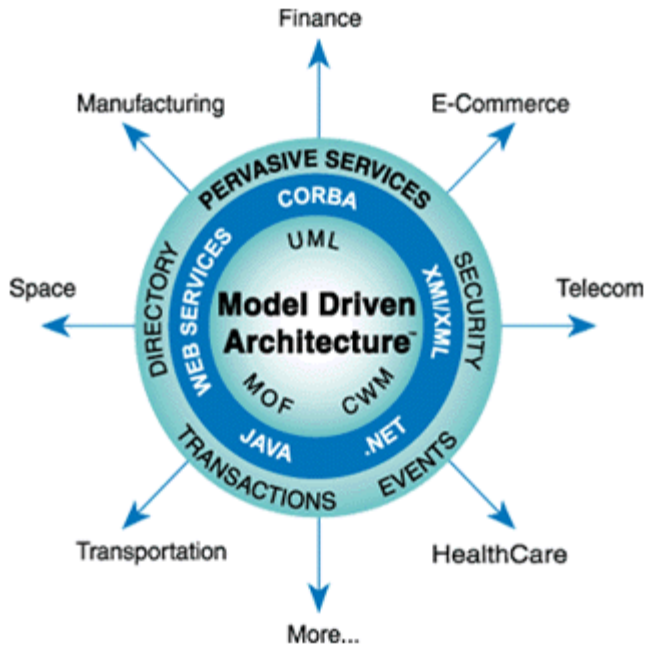
- Systems can be developed by
  1. Constructing abstract views of the system and
  2. Transforming (manually or automatically) the resulting models into
     - Other models
     - Source Code
- Transformations are either
  - Automatic
    - Model transformers, code generators, …
  - Manual
    - Models are degraded to pictures

© Torben Weis, Andreas Ulbrich, and Kurt Geihs

OMG's Model Driven Architecture (1/3)

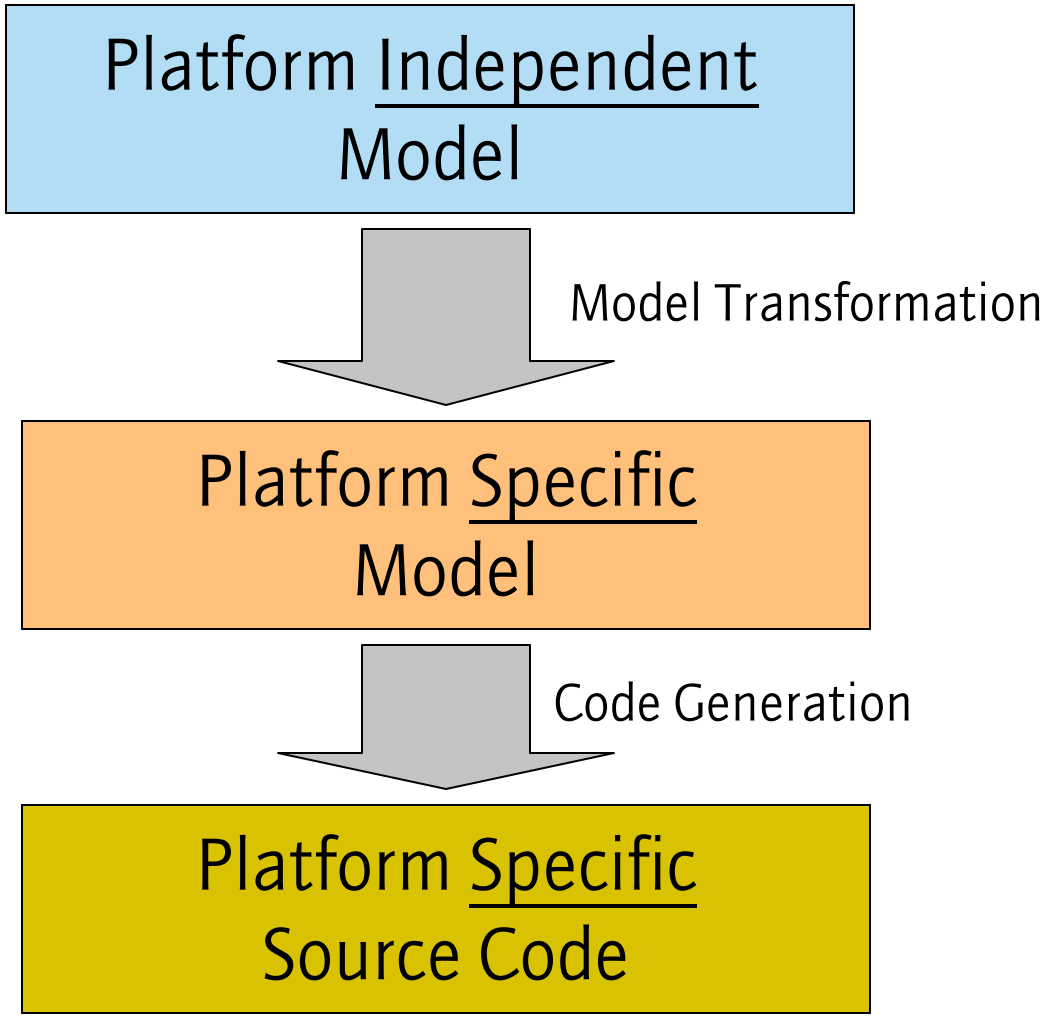
OBJECT MANAGEMENT GROUP

- Separates
- Business or application logic
- from
- Platform technology



© Torben Weis, Andreas Ulbrich, and Kurt Geihs

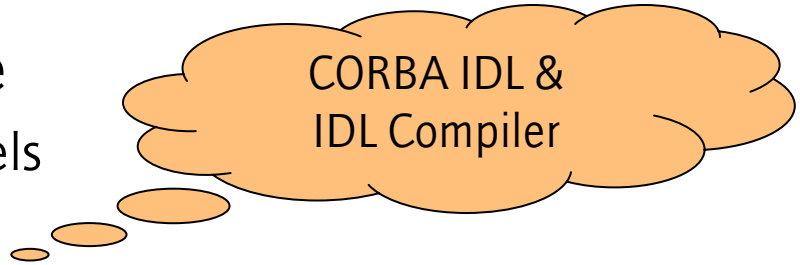
OMG's Model Driven Architecture (2/3)



© Torben Weis, Andreas Ulbrich, and Kurt Geihs

OMG's Model Driven Architecture (3/3)

- The key concepts of the MDA are
  - Platform independent/specific models
  - Generators / transformation tools



- MDA is useful in two kinds of settings

a) A single design has to be mapped to different platforms

   Required for very large enterprise applications

b) Possibility of exchanging the platform "on the fly"

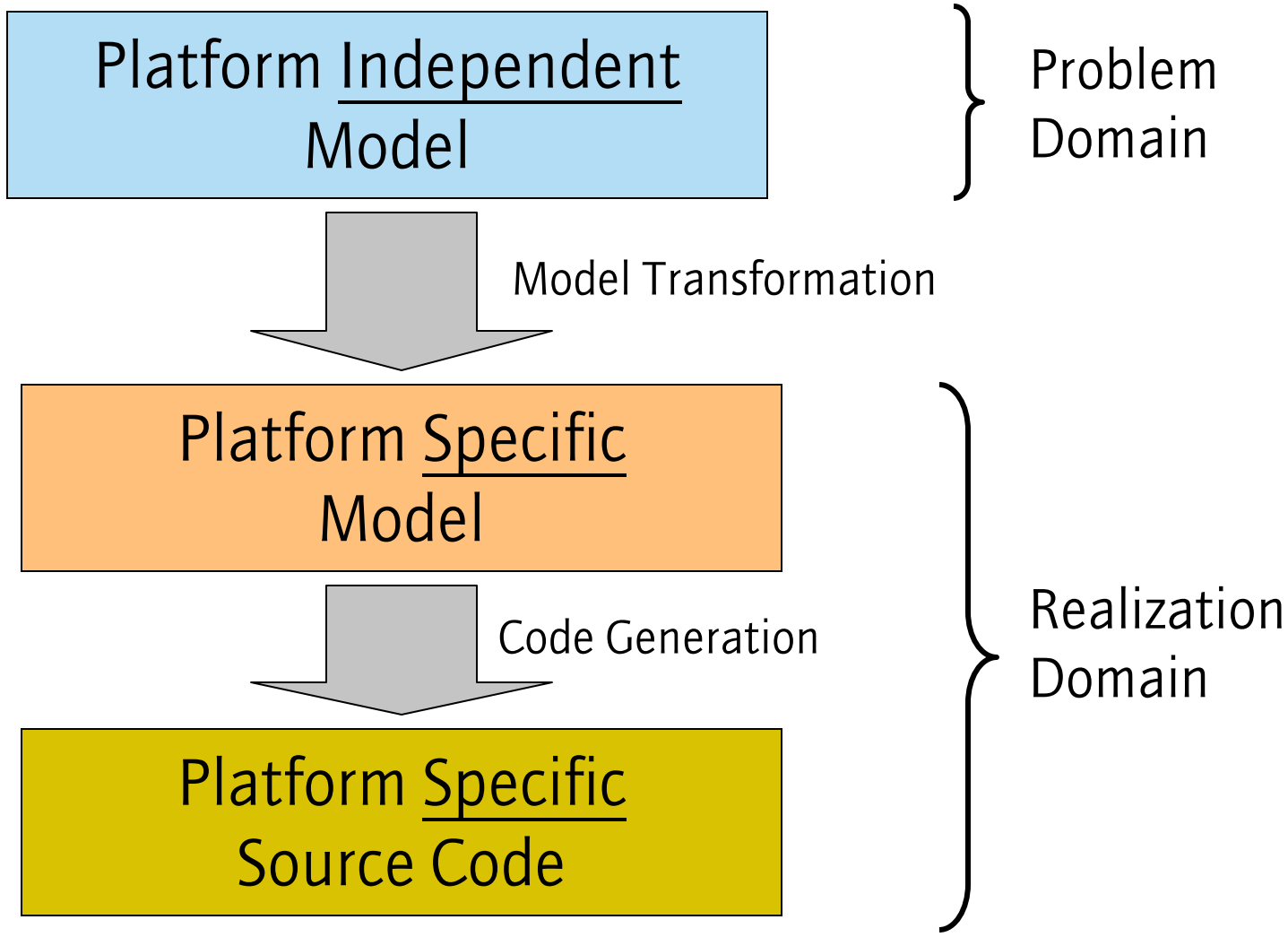
   Limited to selected application categories

c) The platform independent model is simpler than the platform specific model

   That is the case with QoS

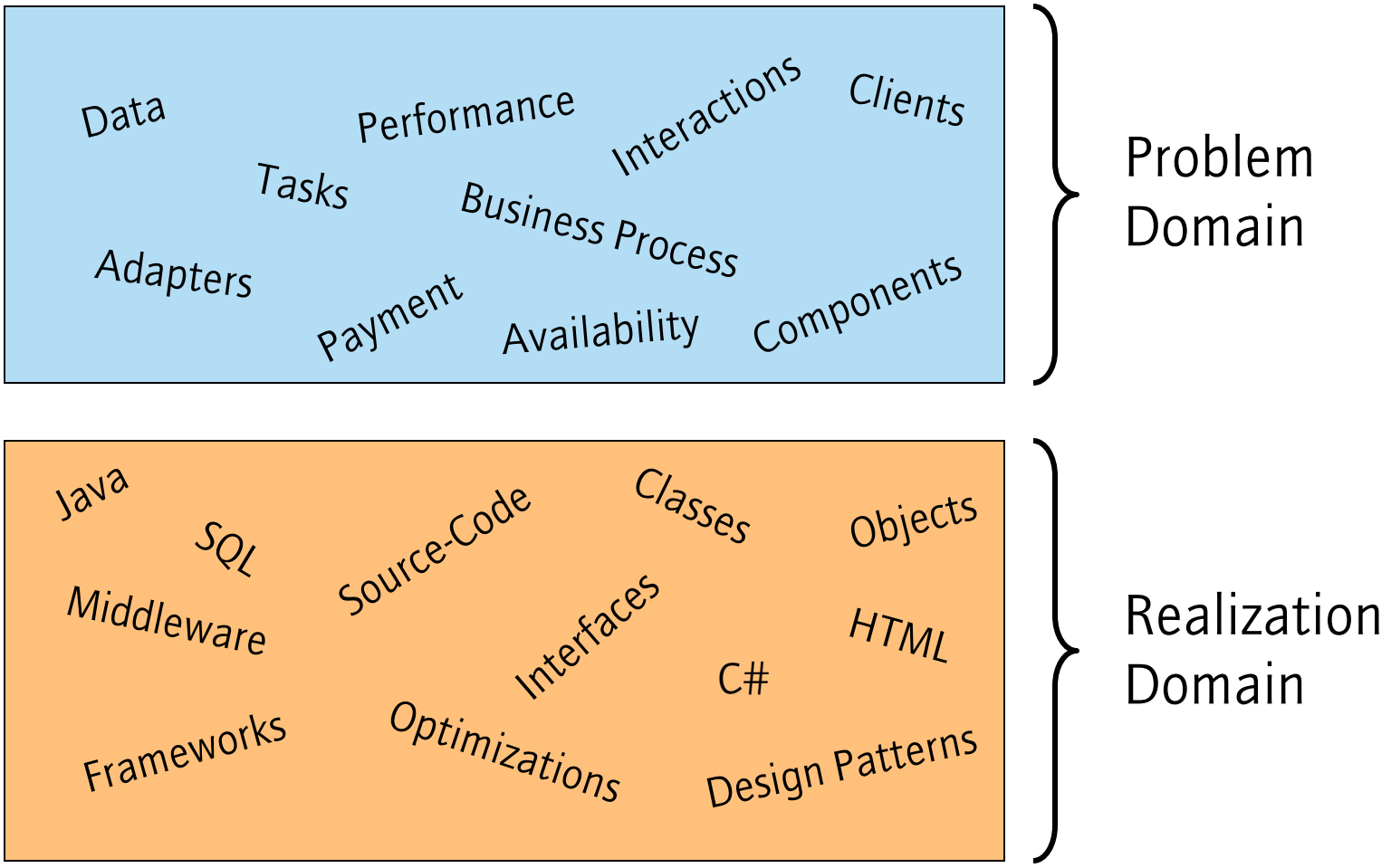
© Torben Weis, Andreas Ulbrich, and Kurt Geihs

Problem and Realization Domain (1/2)



© Torben Weis, Andreas Ulbrich, and Kurt Geihs

Problem and Realization Domain (2/2)



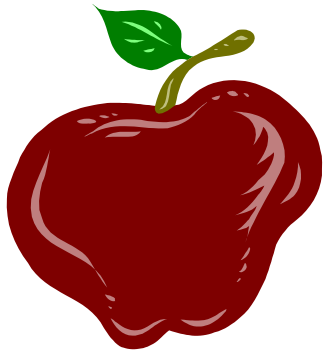
© Torben Weis, Andreas Ulbrich, and Kurt Geihs

UML Stereotypes

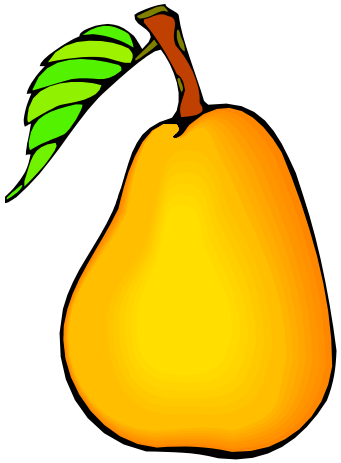
>Stereotypes are the built-in extension mechanisms of the UML

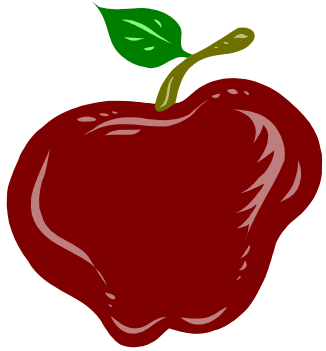
>Stereotypes can be …

- > used for refinements
- > abused for reinterpretations



<<Cox Orange>>

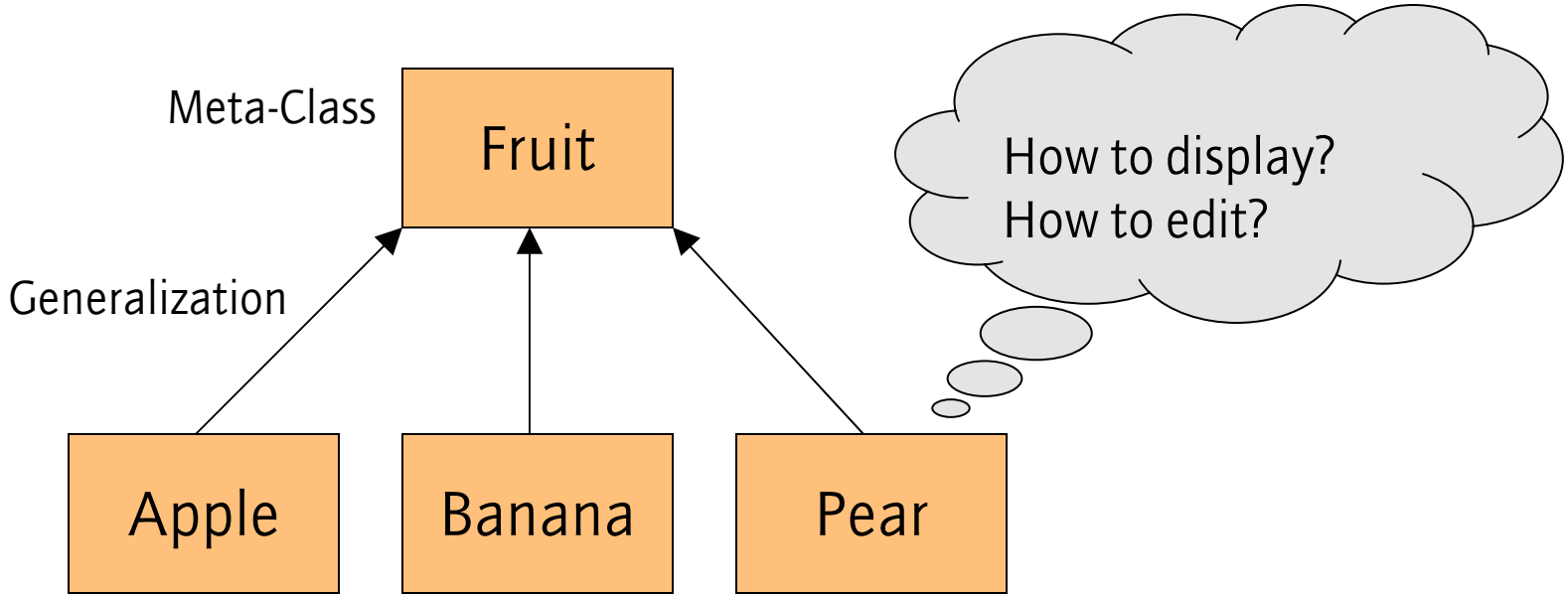




<<Pear>>

© Torben Weis, Andreas Ulbrich, and Kurt Geihs

UML Meta-Model Extensions



© Torben Weis, Andreas Ulbrich, and Kurt Geihs

Modeling QoS-aware Applications - Overview

- Motivation
- Model-driven Development
- QoS-support in platform independent models (PIM)
  - Components
  - QoS-contracts
  - Binding QoS-contracts to interfaces and tasks
  - Contract dependencies
- Model Transformation
- QoS-support in platform specific models (PSM)

© Torben Weis, Andreas Ulbrich, and Kurt Geihs

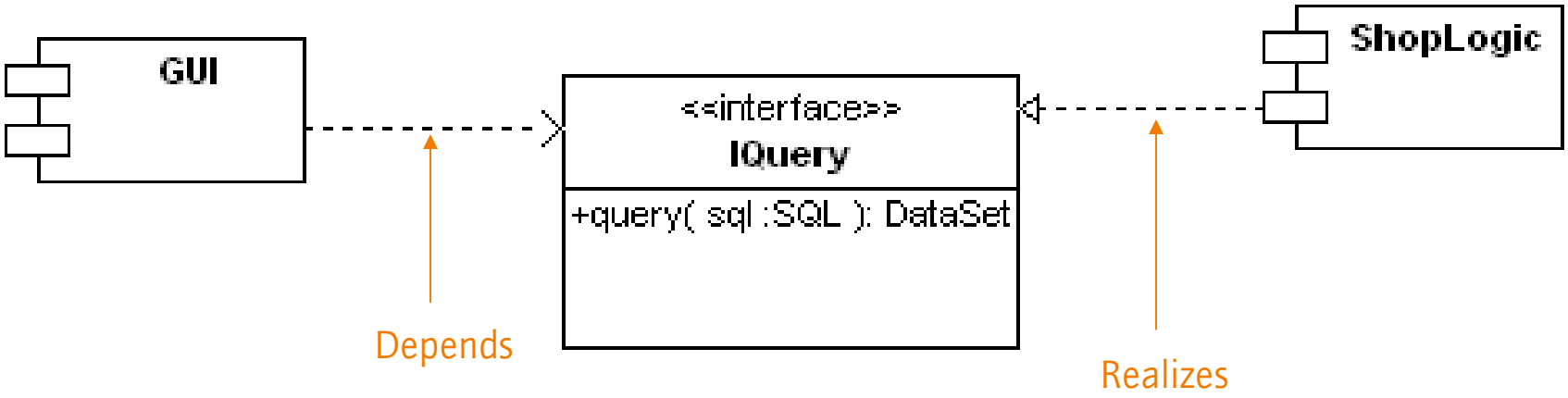


UML Components

- UML 1.4 components
  - Conceptual mistakes
  - Not very useful for QoS-contracts
- EDOC – CCA components
  - OMG Standard
  - Did not get lots of attention
  - Could be extended with QoS-contracts
- UML 2.0 draft by Rational, et al.
  - Supports Ports and Connectors
  - Conceptually close to EDOC-CCA components
  - QoS-contracts can be attached to ports or connectors

© Torben Weis, Andreas Ulbrich, and Kurt Geihs

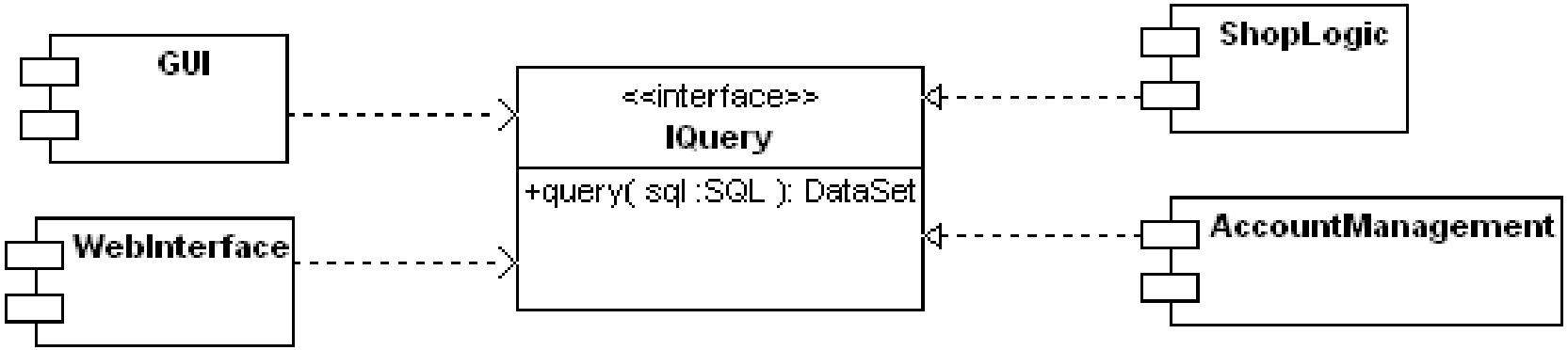
UML 1.x Components (1/3)



UML 1.4 components have no "port" concept

© Torben Weis, Andreas Ulbrich, and Kurt Geihs

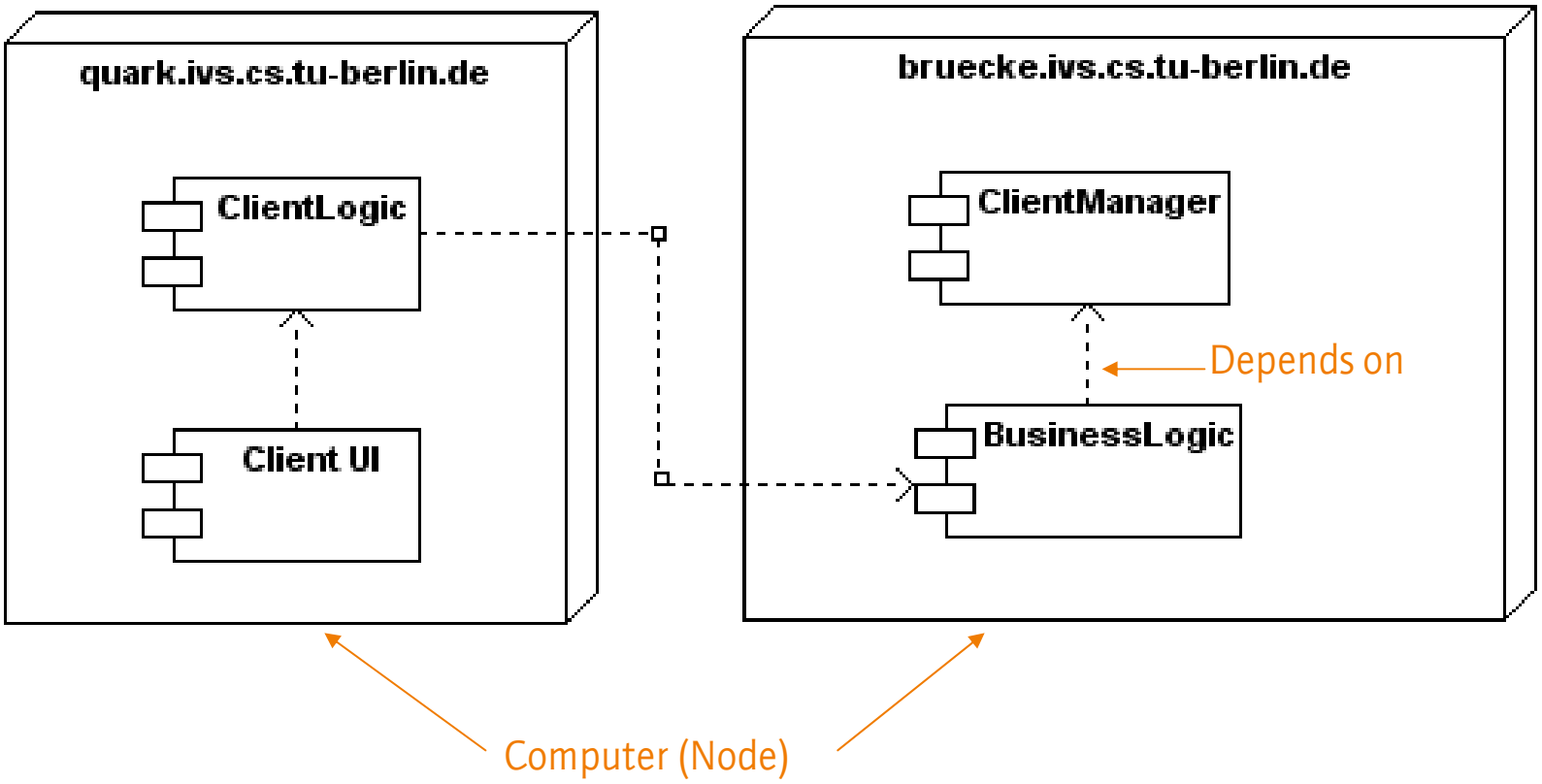
UML 1.x Components (2/3)



© Torben Weis, Andreas Ulbrich, and Kurt Geihs

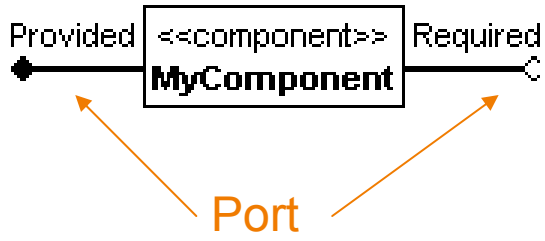
UML 1.x Components (3/3)

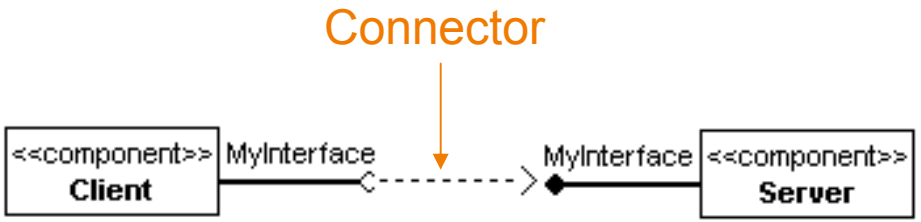
Components in Deployment Diagrams

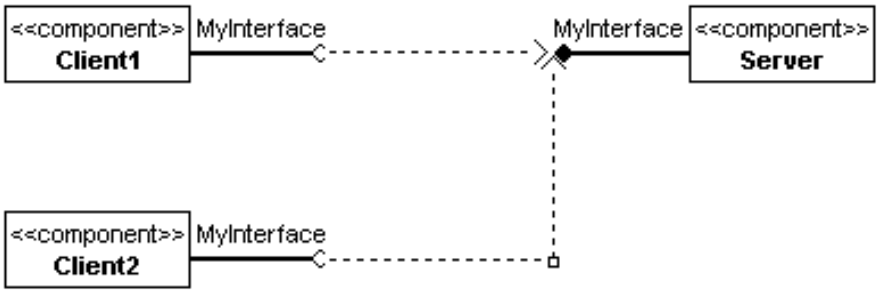


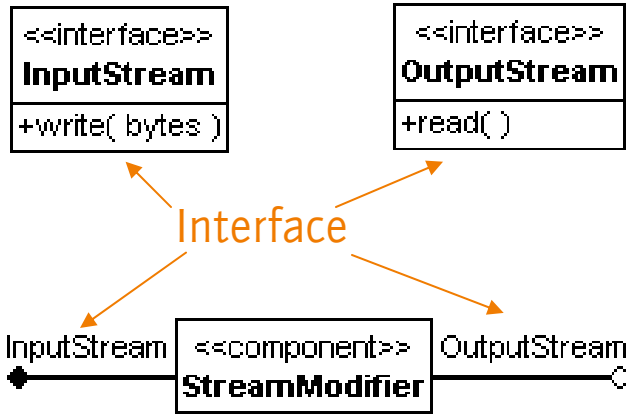
© Torben Weis, Andreas Ulbrich, and Kurt Geihs

UML 2.0 Components (1/2)



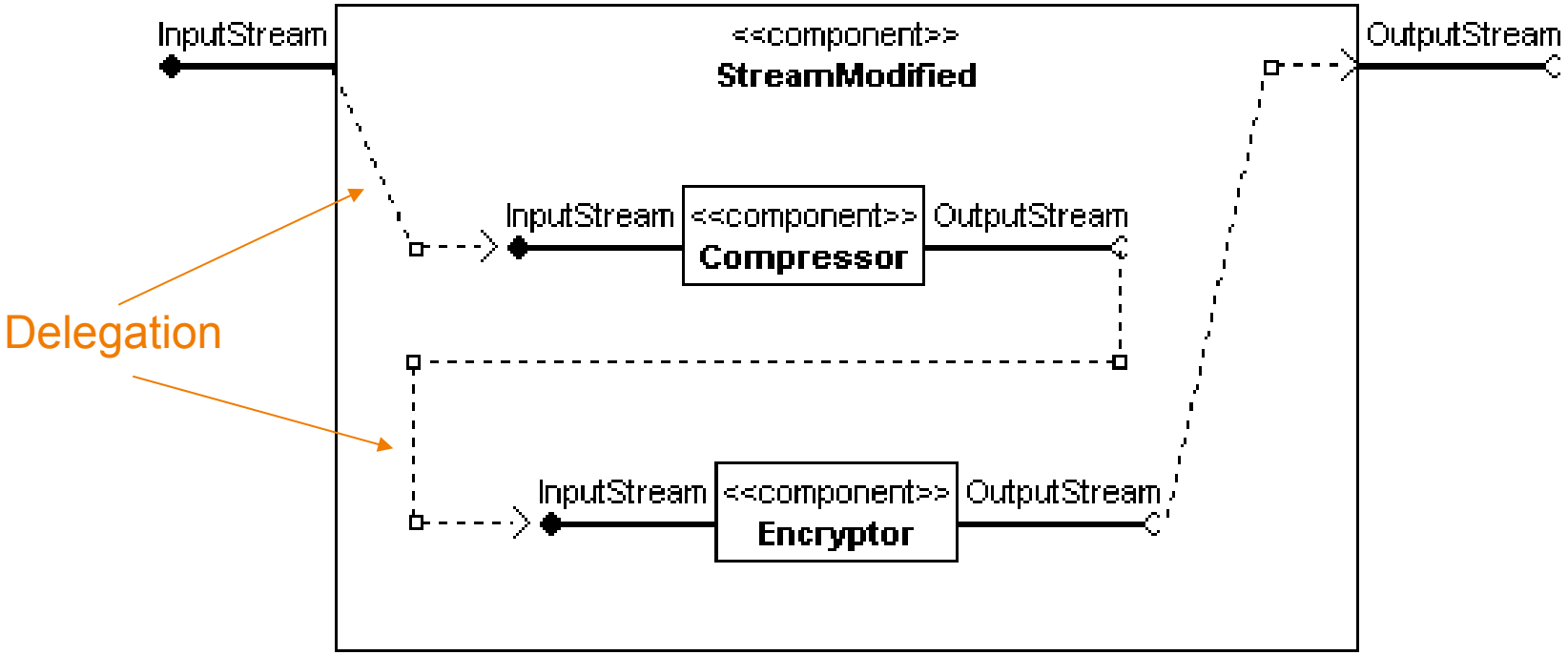






© Torben Weis, Andreas Ulbrich, and Kurt Geihs

UML 2.0 Components (2/2)



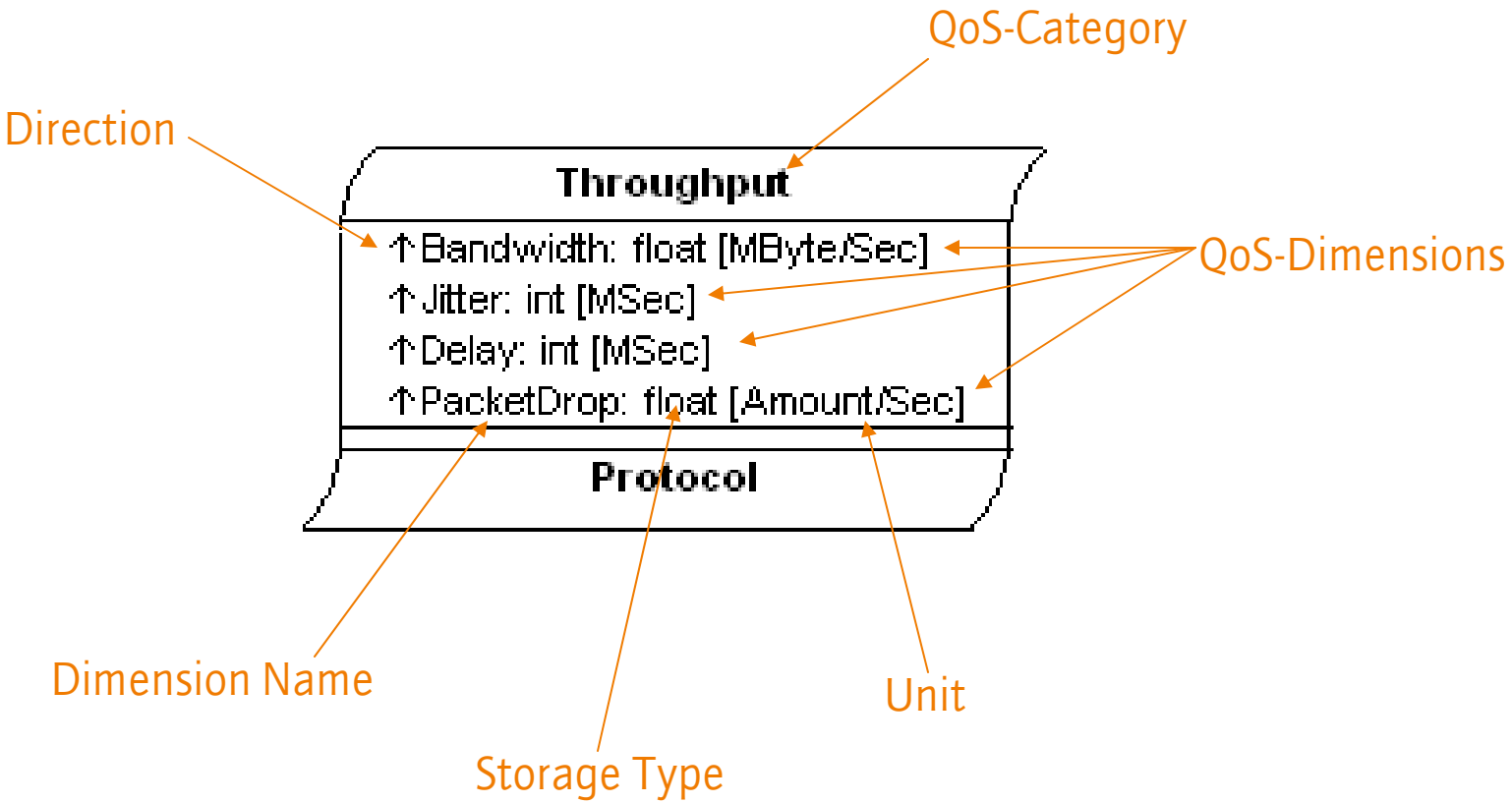
© Torben Weis, Andreas Ulbrich, and Kurt Geihs

Modeling QoS-aware Applications - Overview

- Motivation
- Model-driven Development
- QoS-support in platform independent models (PIM)
  - Components
  - QoS-contracts
  - Binding QoS-contracts to interfaces and tasks
  - Contract dependencies
- Model Transformation
- QoS-support in platform specific models (PSM)

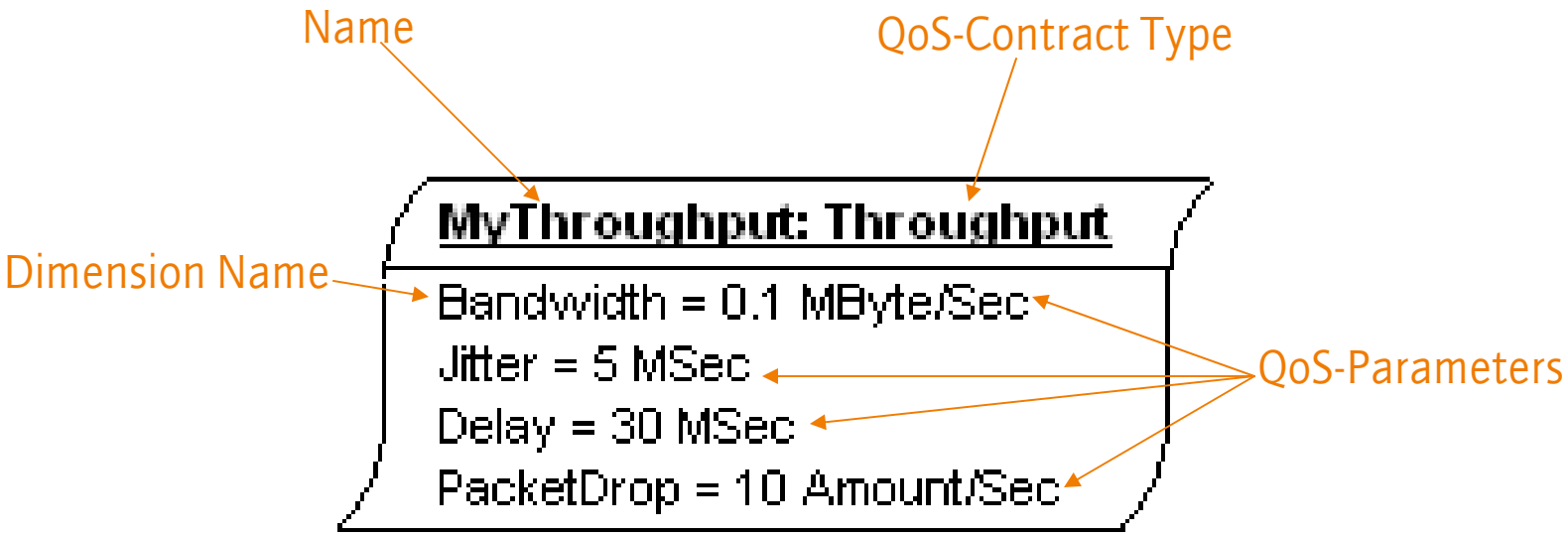
© Torben Weis, Andreas Ulbrich, and Kurt Geihs

Notation (1/2)



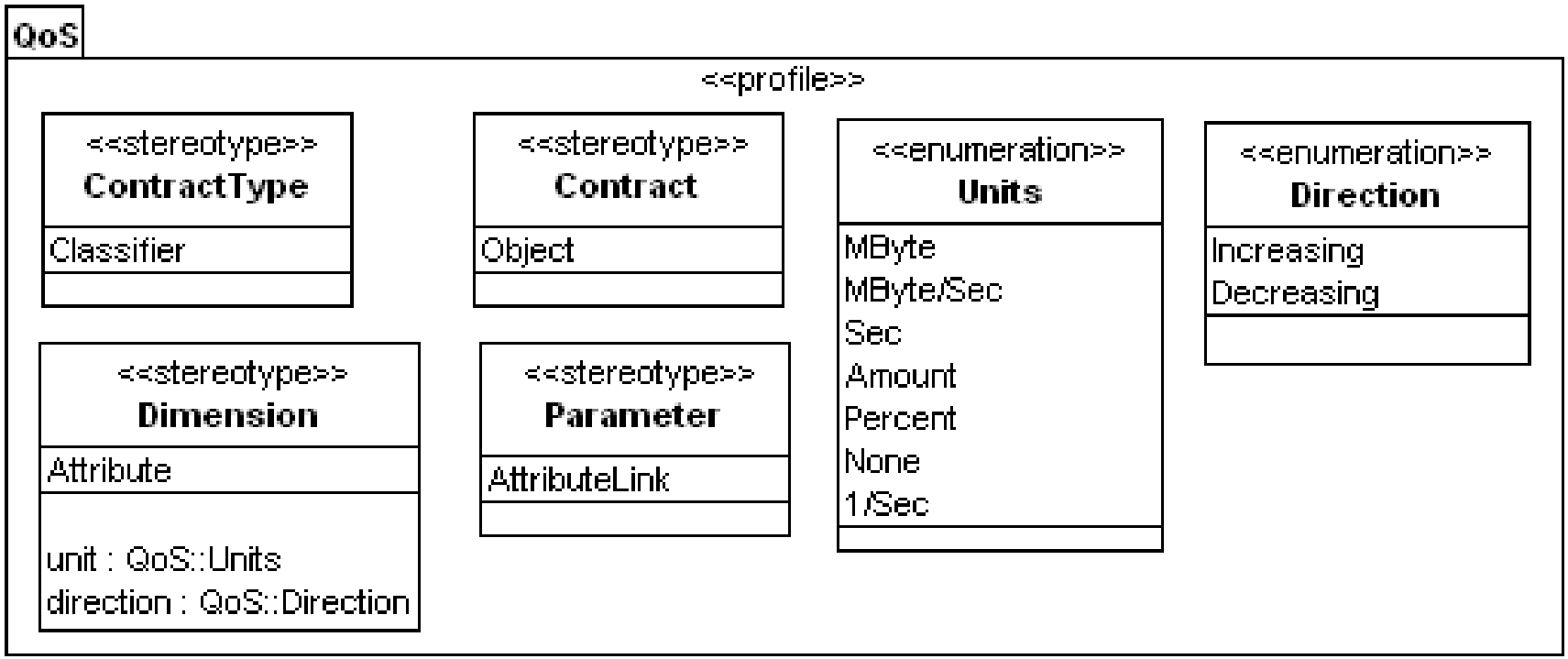
© Torben Weis, Andreas Ulbrich, and Kurt Geihs

Notation (2/2)



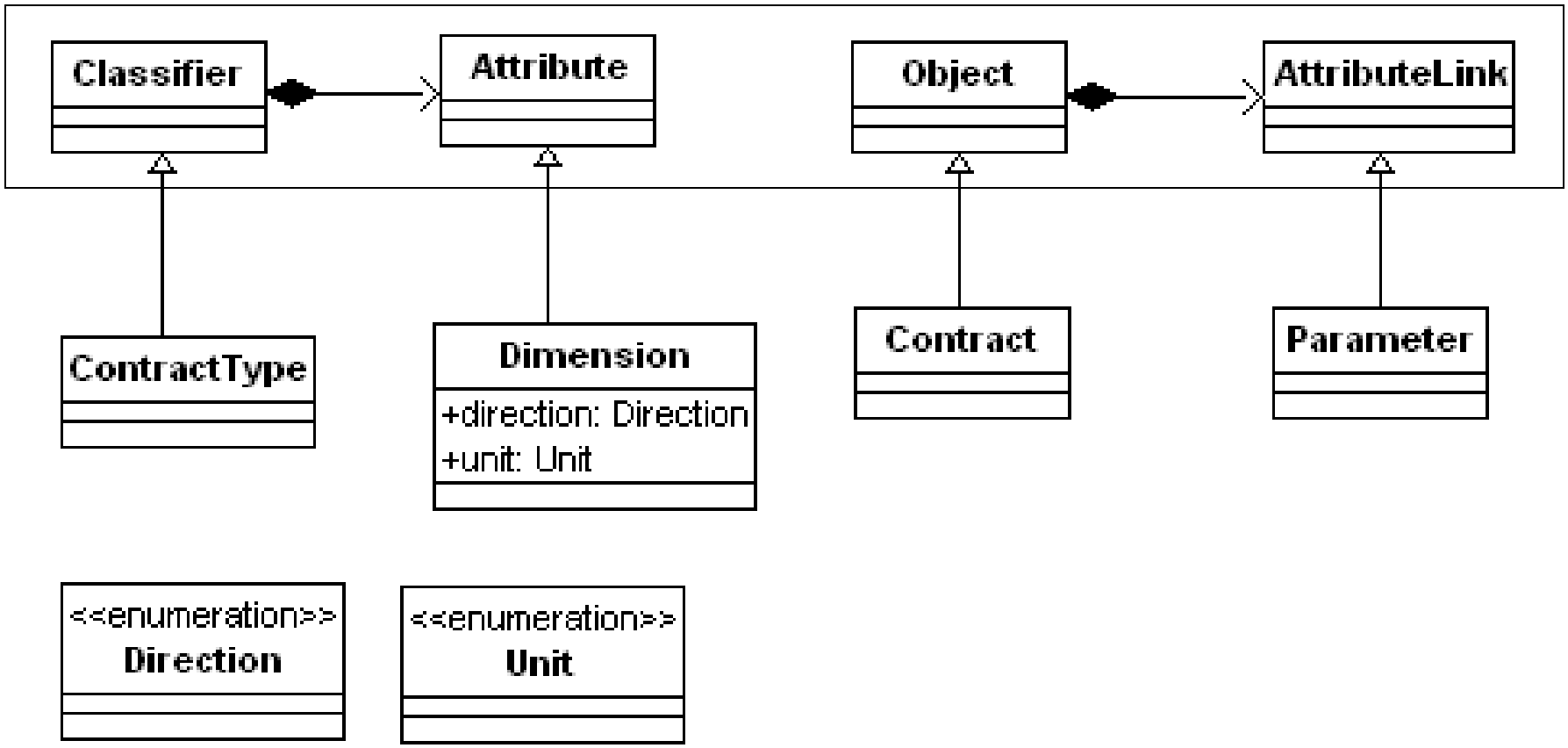
© Torben Weis, Andreas Ulbrich, and Kurt Geihs

QoS Profile (partial)



© Torben Weis, Andreas Ulbrich, and Kurt Geihs

QoS Meta Model (partial)



© Torben Weis, Andreas Ulbrich, and Kurt Geihs

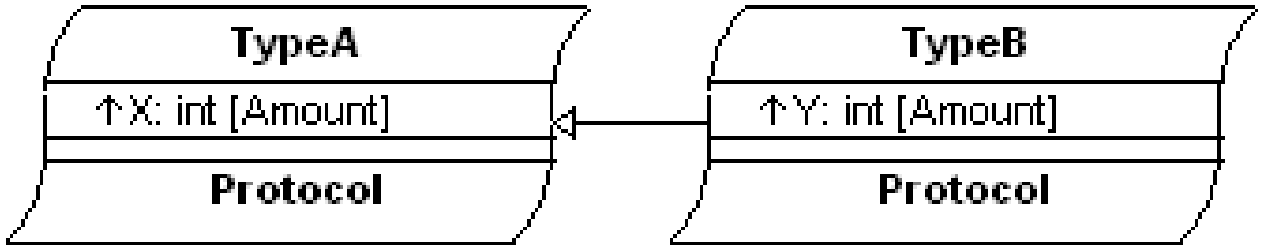
Contract Specialization

> A(X) := Constraint on provided QoS

> B(Y) := Additional constraint on provided QoS

> G(Z) := { true : A(Z->X) and B(Z->Y)

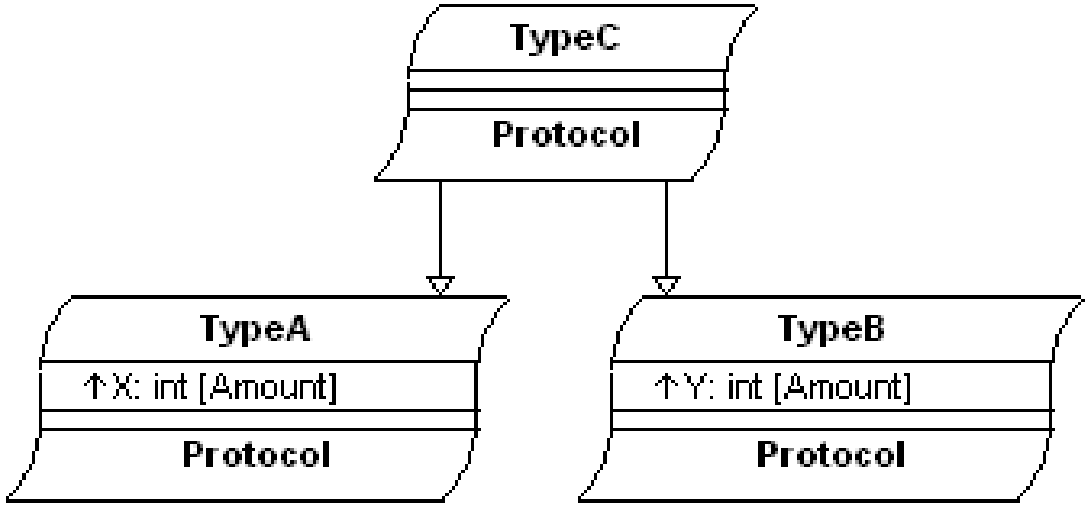
> { false: otherwise



© Torben Weis, Andreas Ulbrich, and Kurt Geihs

Polymorphism

- C(X,Y) := true : A(X) and B(Y)
- false: otherwise
- A contract negotiation of `TypeC` is an atomic negotiation of `TypeA` and `TypeB`



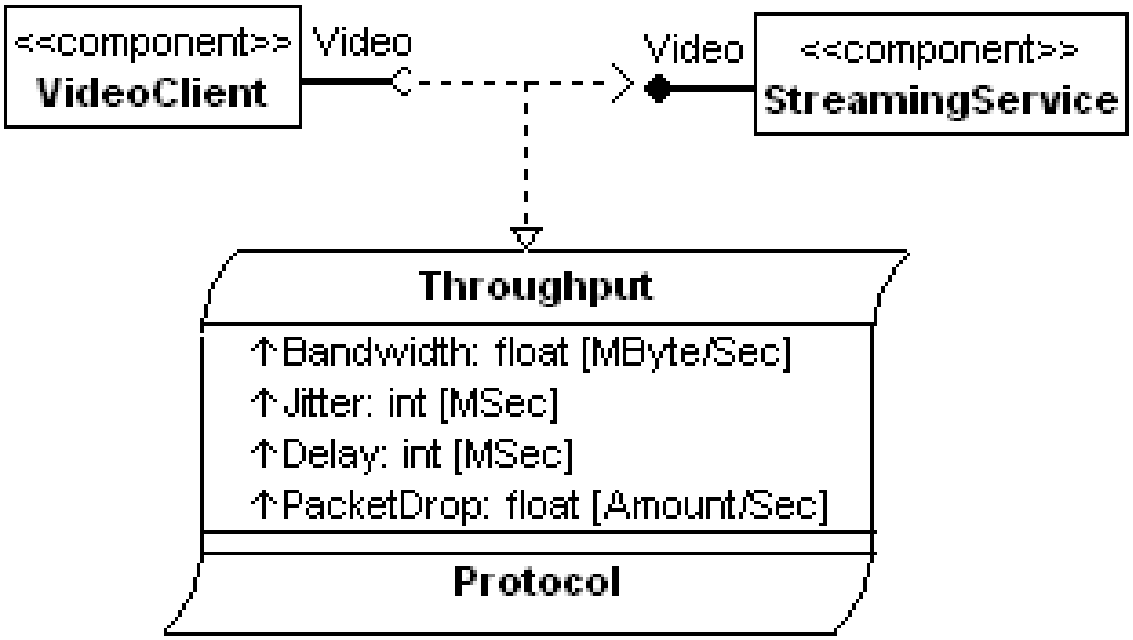
© Torben Weis, Andreas Ulbrich, and Kurt Geihs

Modeling QoS-aware Applications - Overview

- Motivation
- Model-driven Development
- QoS-support in platform independent models (PIM)
  - Components
  - QoS-contracts
  - Binding QoS-contracts to interfaces and tasks
  - Contract dependencies
- Model Transformation
- QoS-support in platform specific models (PSM)

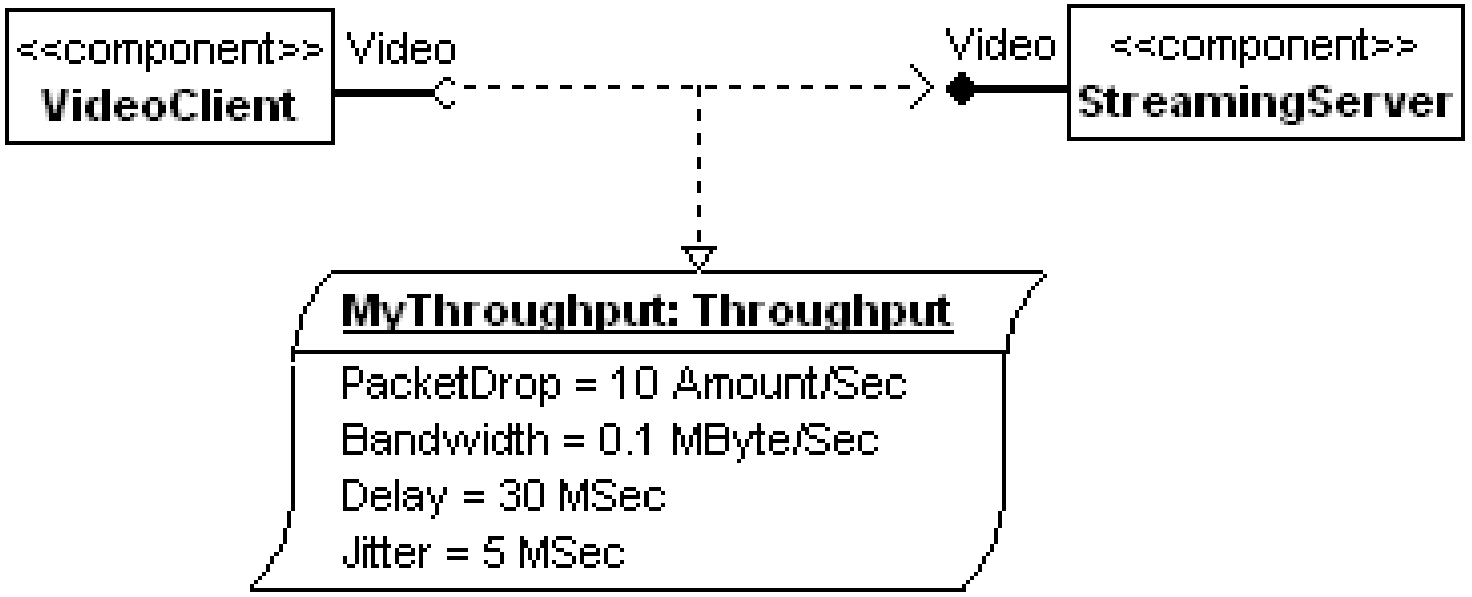
© Torben Weis, Andreas Ulbrich, and Kurt Geihs

Binding QoS-Contract Types to Connectors



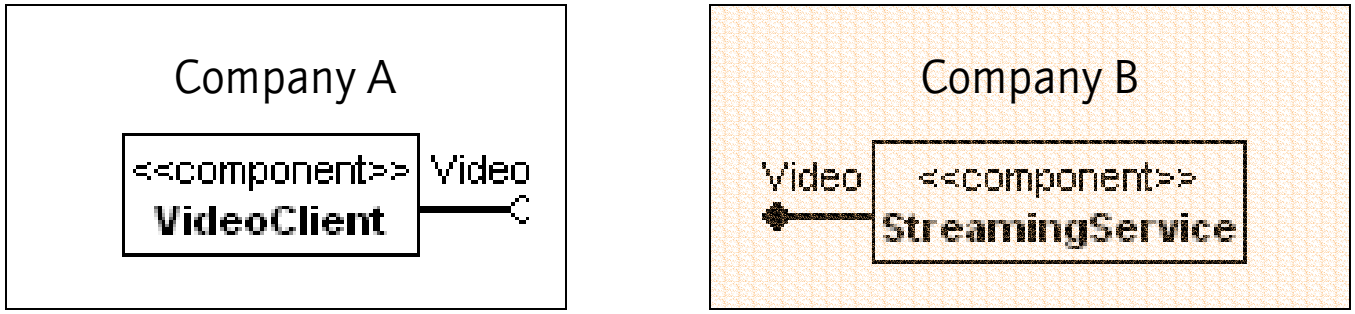
© Torben Weis, Andreas Ulbrich, and Kurt Geihs

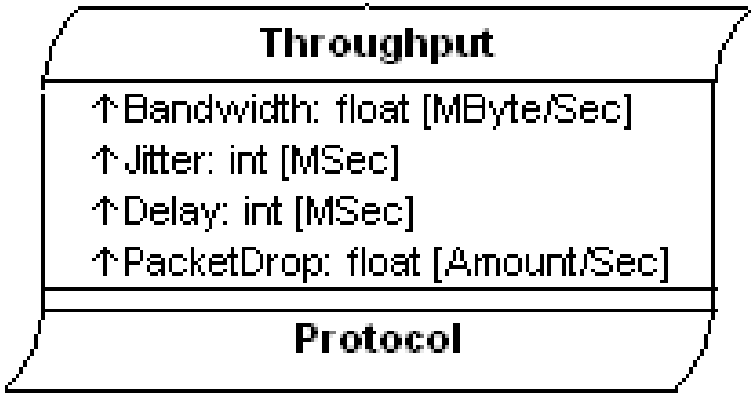
Binding QoS-Contracts to Connectors



© Torben Weis, Andreas Ulbrich, and Kurt Geihs

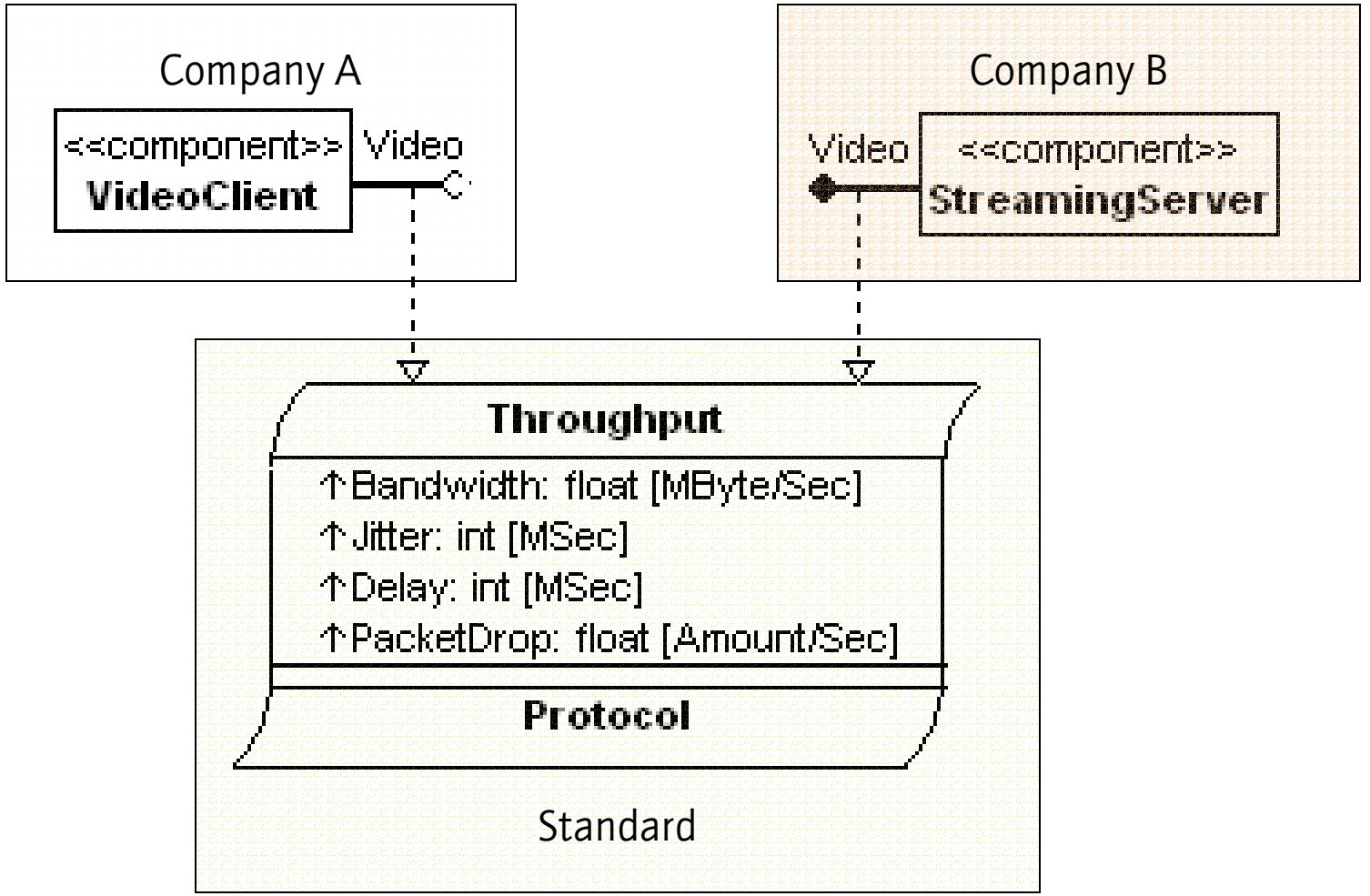
Bottom-Up Development (1/2)





© Torben Weis, Andreas Ulbrich, and Kurt Geihs

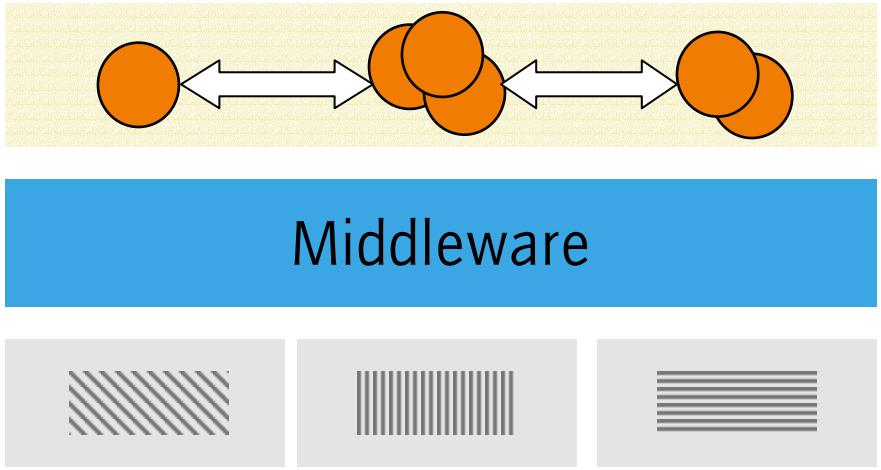
Bottom-Up Development (2/2)



© Torben Weis, Andreas Ulbrich, and Kurt Geihs

Middleware

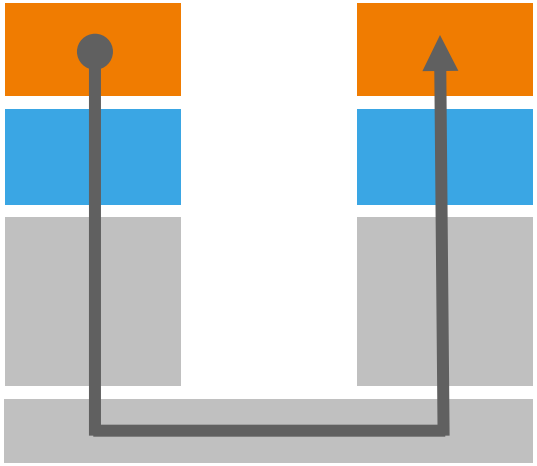
- Platform support for distributed applications
  - interaction, integration, interoperability, coordination
  - distribution transparency
- Examples: CORBA, J2EE/EJB, .NET, …



© Torben Weis, Andreas Ulbrich, and Kurt Geihs

QoS Properties

- End-to-End
  - QoS management involves all layers along the message path
- Cross Cutting
  - QoS management cannot be encapsulated
- Phases
  - specification — category and parameters
  - implementation — map to mechanisms and integration
  - runtime — 1. negotiation; 2. monitoring, adaptation, accounting



© Torben Weis, Andreas Ulbrich, and Kurt Geihs

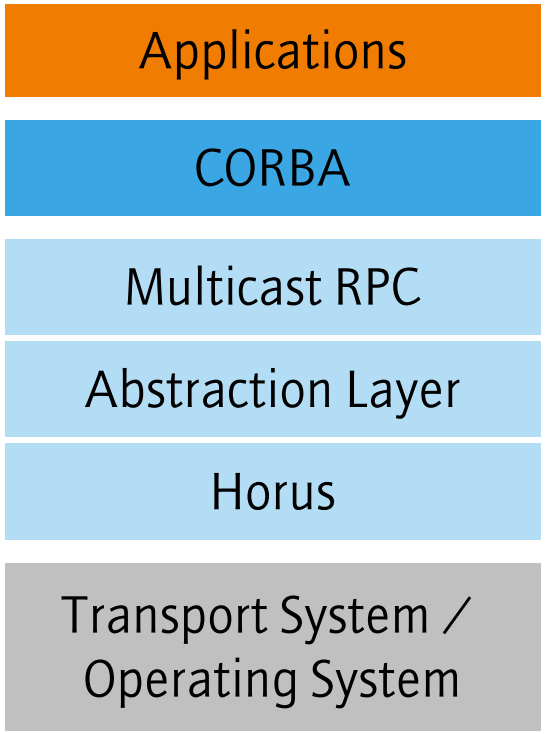
QoS in Middleware

- Single category platforms
  - real-time (TAO / D. Schmidt et.al.)
  - availability thru replication (Electra/S. Maffeis, Eternal/Moser)
- Frameworks for multiple QoS categories
  - QuO (BBN)
  - MAQS (University of Frankfurt/Main, TU Berlin)
  - Aspectix (Uni Erlangen)
  - DotQoS (TU Berlin)

© Torben Weis, Andreas Ulbrich, and Kurt Geihs

Example: Electra (S. Maffeis, 1995)

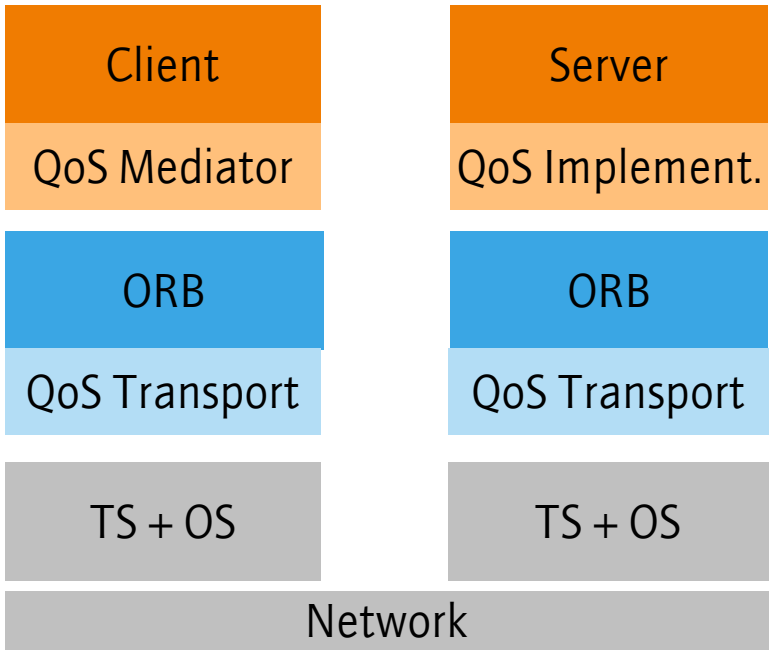
- Availability thru replication
- extensions to CORBA
- service is group of objects
- uses atomic broadcast (e.g. Horus toolkit)
- pre-configured platform with specific ORB objects and interfaces



© Torben Weis, Andreas Ulbrich, and Kurt Geihs

Example: MAQS (Becker/Geihs, 1997 ff.)

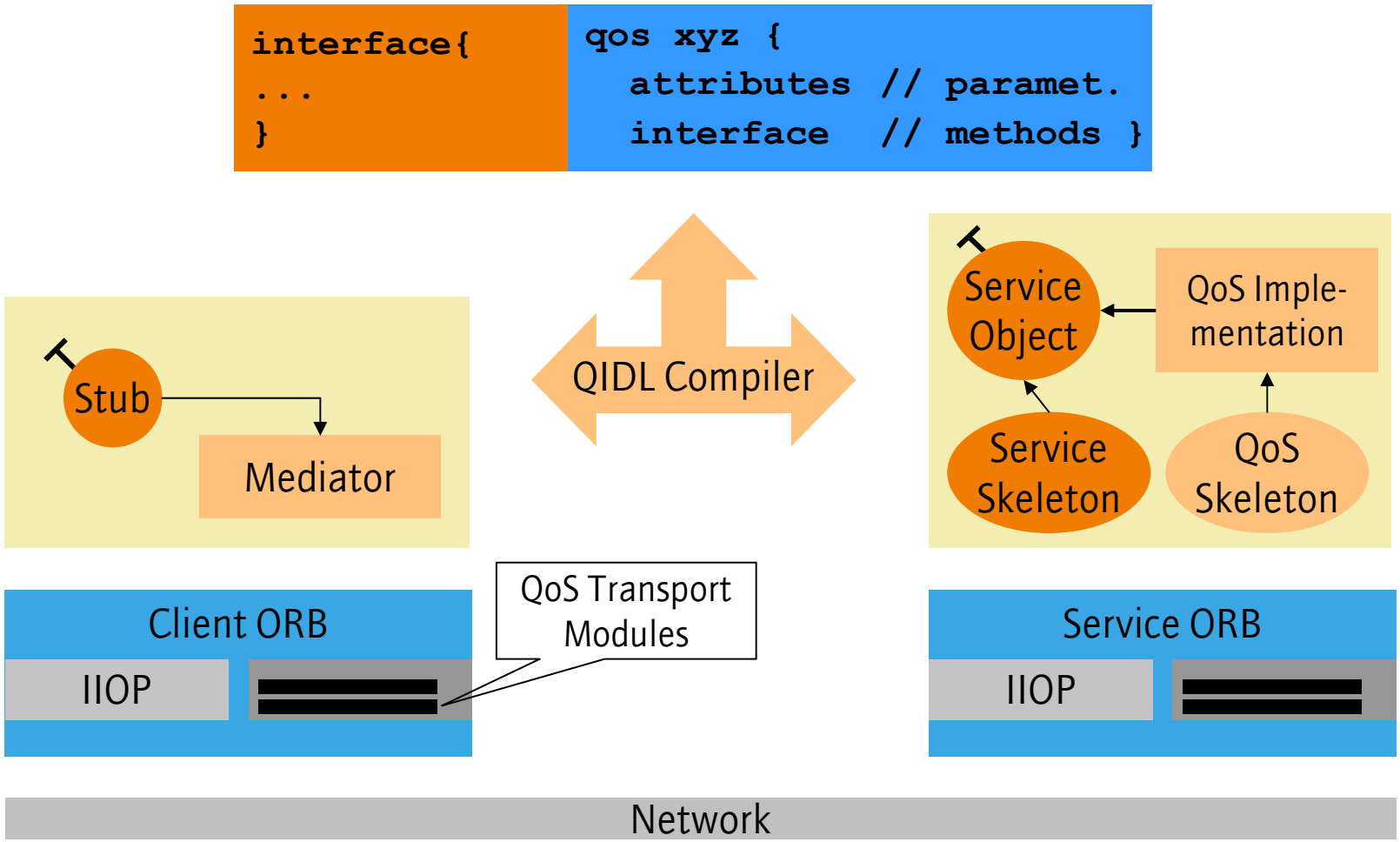
- Generic QoS framework
- based on CORBA (MICO)
- separation of concerns
  - application funct. → QoS functionality
  - QoS specification → QoS implement.
- guideline:
  „QoS is aspect" (→ AOP)
- extensions to CORBA
  - enhanced IDL → QIDL
  - negotiation protocol for QoS contracts
  - integration of QoS mechanisms



(Becker 2000)

© Torben Weis, Andreas Ulbrich, and Kurt Geihs

MAQS: System architecture



© Torben Weis, Andreas Ulbrich, and Kurt Geihs



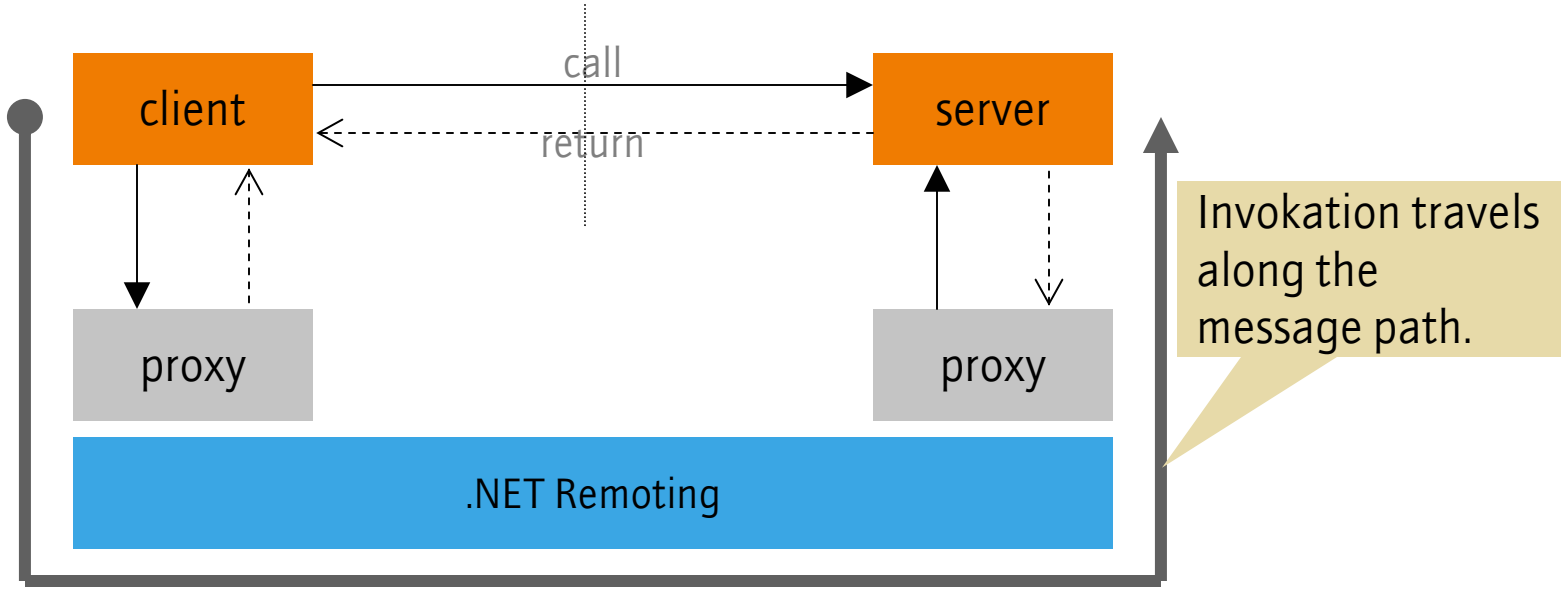
Overview

- QoS Explained
- .NET Remoting
- DotQoS – QoS in .NET

© Torben Weis, Andreas Ulbrich, and Kurt Geihs

Using Remote Objects with .NET Remoting

- .NET Remoting
  - Distribution transparency
  - Abstracts from message creation, transport, and dispatching
  - Comparable to OMG CORBA or Java RMI



© Torben Weis, Andreas Ulbrich, and Kurt Geihs

Example: Remote Object Implementation

```
public class HelloWorld : MarshalByRefObject
{
    public HelloWorld() {}

    public string SayHello()
    {
        return „Hello World!“
    }
}
```


© Torben Weis, Andreas Ulbrich, and Kurt Geihs



Example: Client Mainline

```
public class Client
{
    public static void Main(string args[])
    {
        RemotingConfiguration.Configure(„client.xml“);

        HelloWorld hello = new HelloWorld();
        Console.WriteLine(hello.SayHello());
    }
}
```


© Torben Weis, Andreas Ulbrich, and Kurt Geihs



Example: Server Mainline

```
public class Server
{
    public static void Main(string args[])
    {
        RemotingConfiguration.Configure(„server.xml“);
        Console.ReadLine();
    }
}
```


© Torben Weis, Andreas Ulbrich, and Kurt Geihs



Example: Client Configuration

```
<configuration>
    <system.runtime.remoting>
        <application>
            <client url="tcp://localhost:4711/HelloWorld">
                <activated type="HelloWorld, server"/>
            </client>
        </application>
    </system.runtime.remoting>
</configuration>
```


© Torben Weis, Andreas Ulbrich, and Kurt Geihs

Example: Server Configuration

```
<configuration>
    <system.runtime.remoting>
        <application>
            <channels>
                <channel ref="tcp" port="4711">
            </channels>
            <service>
                <activated type="HelloWorld, server"/>
            </service>
        </application>
    </system.runtime.remoting>
</configuration>
```


© Torben Weis, Andreas Ulbrich, and Kurt Geihs

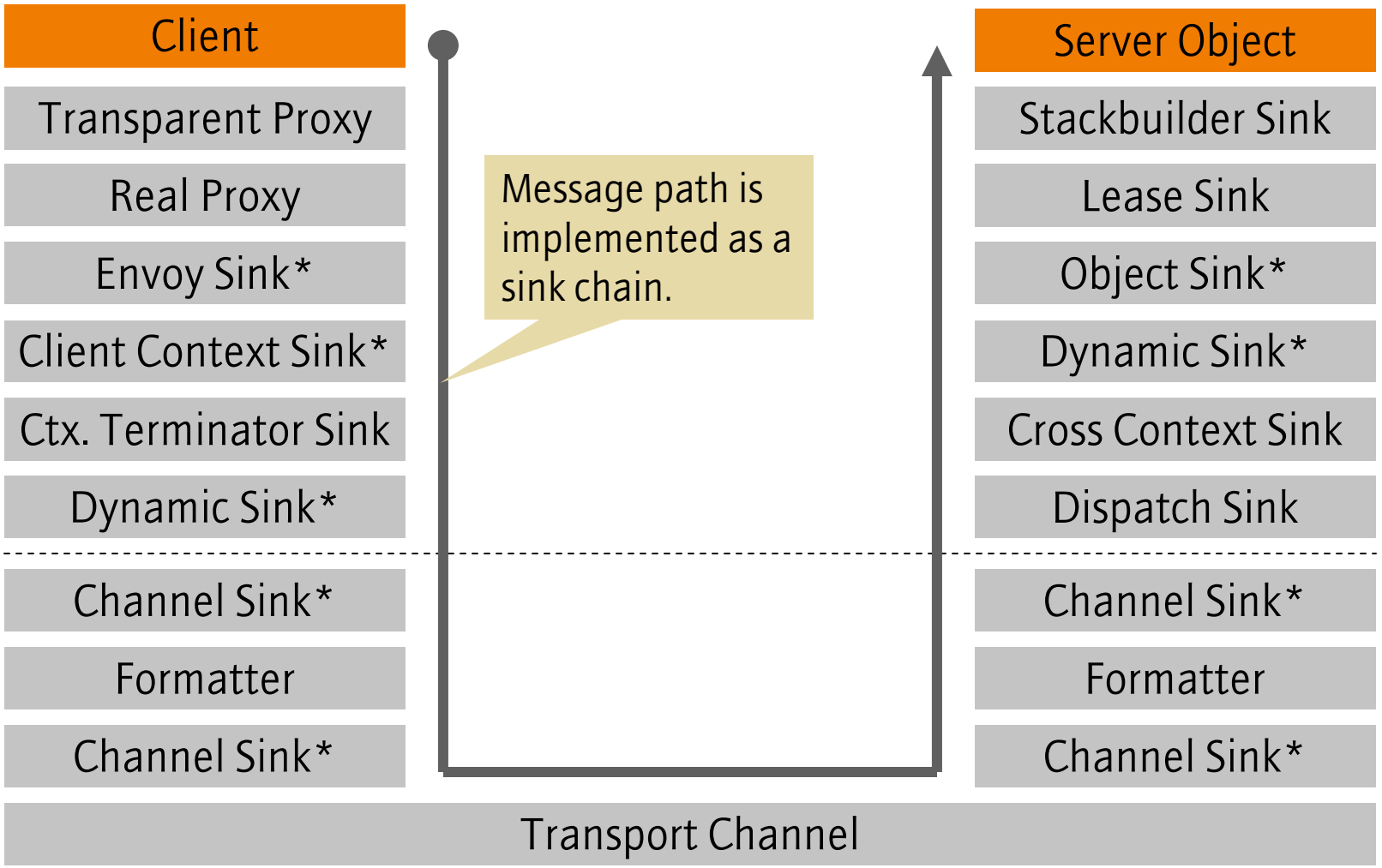


How it works

- Client side proxy created automatically
  - No IDL → No rmic, no idl-compiler, …
  - Uses reflection and reflection emit at runtime
- Generic server side proxy
  - Stackbuilder dispatches method invokations
  - Uses reflection
- Almost everything configured using XML-Files
- Message Path is configurable

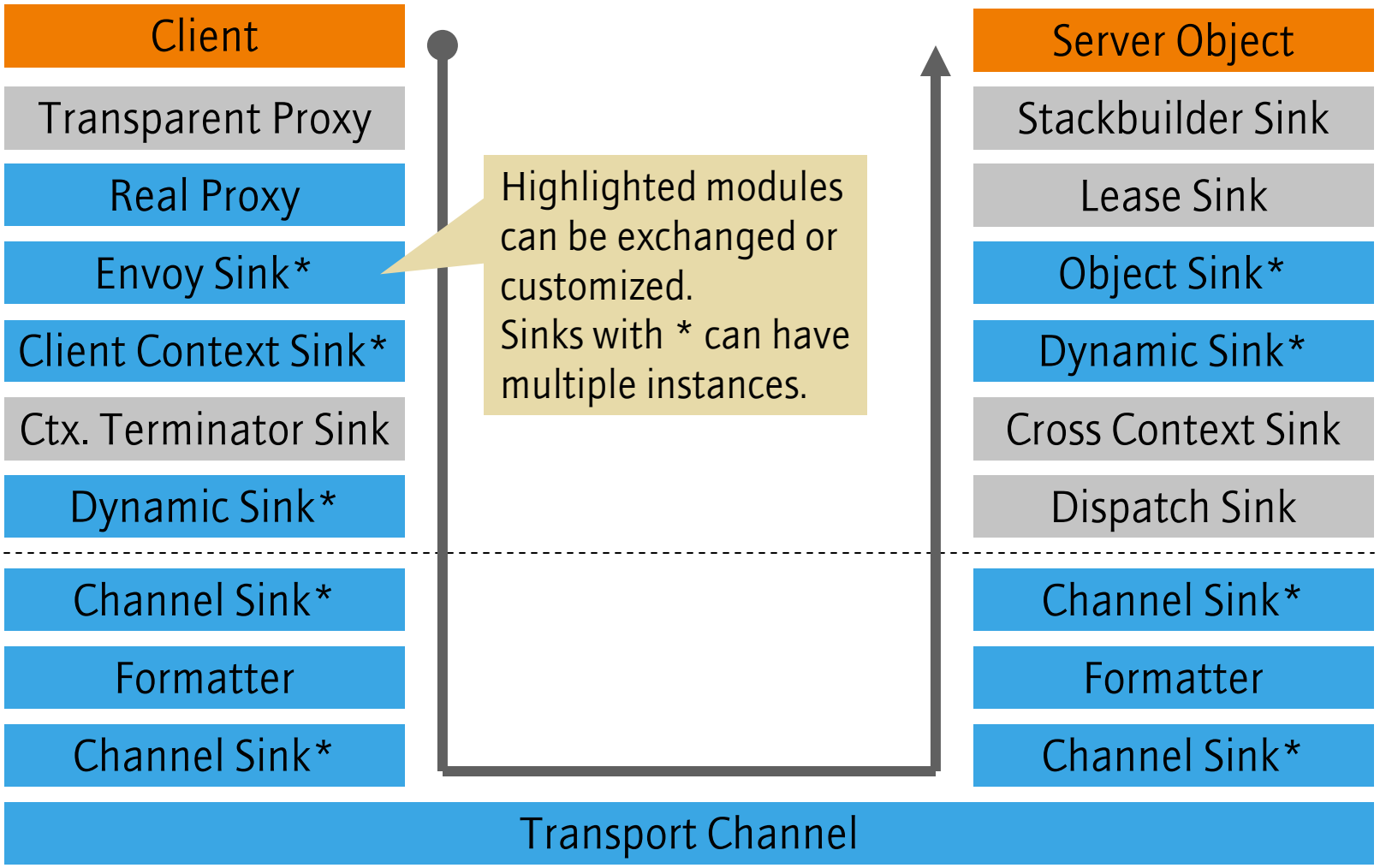
© Torben Weis, Andreas Ulbrich, and Kurt Geihs

Remoting Message Path



© Torben Weis, Andreas Ulbrich, and Kurt Geihs

Configurable Sink Chain



© Torben Weis, Andreas Ulbrich, and Kurt Geihs

Example: Sink Implementation

```
public class PriorityChangerSink : IServerChannelSink, IChannelSinkBase
{
    public PriorityChangerSink(IServerChannelSink next) { this.next = next; }

    public ServerProcessing ProcessMessage(
        sinkStack, reqMsg, reqHdrs, reqStrm,
        out resMsg, out resHdrs, out resStrm)
    {
        ThreadPriority oldprio = Thread.CurrentThread.Priority;
        Thread.CurrentThread.Priority = ThreadPriority.Highest;

        ServerProcessing s = this.next.ProcessMessage(sinkStack, reqMsg,
            reqHdrs, out reqStrm, out resHdrs, out resStrm);

        Thread.CurrentThread.Priority = oldprio;
        return s;
    }
}
```

(Rammer 2002)

© Torben Weis, Andreas Ulbrich, and Kurt Geihs

Example: Sink Configuration

```
<configuration>
    <system.runtime.remoting>
        <application>
            <channels>
                <channel ref="http" port="5180">
                    <serverProviders>
                        <formatter ref="soap"/>
                        <provider
                            type="PriorityChangerSinkProvider, priority"/>
                    </serverProviders>
                </channel>
            </channels>
        </application>
    </system.runtime.remoting>
</configuration>
```


© Torben Weis, Andreas Ulbrich, and Kurt Geihs



Overview

- QoS Explained
- .NET Remoting
- DotQoS – QoS in .NET

© Torben Weis, Andreas Ulbrich, and Kurt Geihs

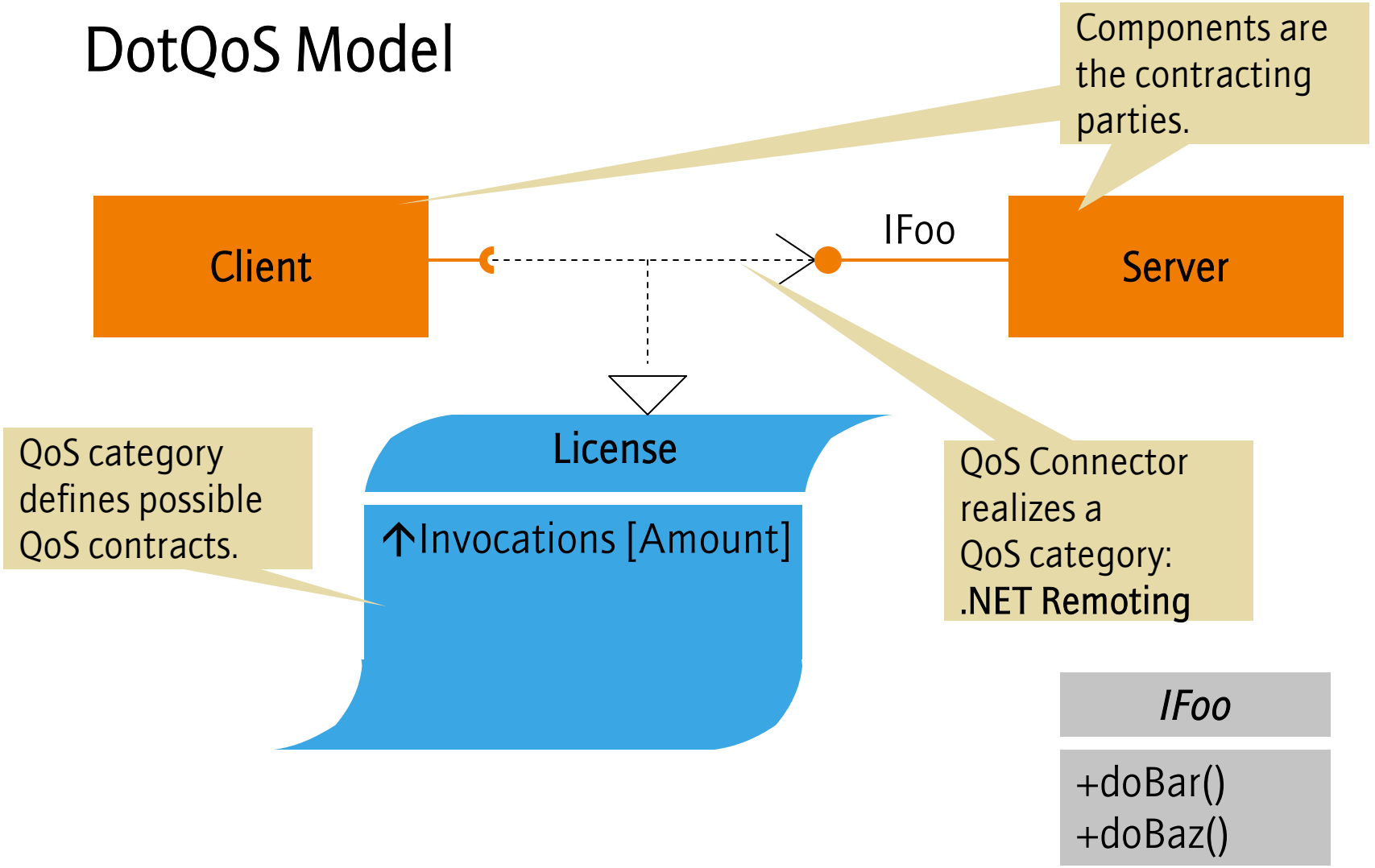


DotQoS Features

- Specification of non-functional properties
  - QoS categories
  - Parameters of QoS categories
- Negotiation
  - Reaching agreement on the delivered QoS
- QoS runtime provisioning
  - Dynamic handling of QoS mechanism
  - Support for multiple QoS categories
  - Extending .NET Remoting

© Torben Weis, Andreas Ulbrich, and Kurt Geihs

DotQoS Model



enhanced UML 2.0 notation

© Torben Weis, Andreas Ulbrich, and Kurt Geihs

QoS Specification in C# (1/2)

```
public class License : QoSCategorySchemeBase
{
    public License() {
        this.Invocations = 0;
    }

    [QoSDimension(QoSUnit.Amount, QoSDirection.Ascending)]
    public int Invocations {
        get { return this.Parameters["#inv"]; }
        set { this.Parameters["#inv"] = value; }
    }
}
```

Base class for all category specifications.

Dimension define QoS parameters.

Unit of measurement, e.g. 1, s^{-1}, s, byte s^{-1} …

Sort-order, e.g. higher value → better quality

© Torben Weis, Andreas Ulbrich, and Kurt Geihs

QoS Specification in C# (2/2)

Defines a class of QoS contract that the component supports

The QoS category that is supported

The port which support the License Category.

```
[QoSContractClass(typeof(License), typeof(IFoo))]
public class ServerImpl : DotQoS.RemoteObject, IFoo
{
    public ServerImpl() {}

    public void doBar() { … }
    public void doBaz() { … }
}
```

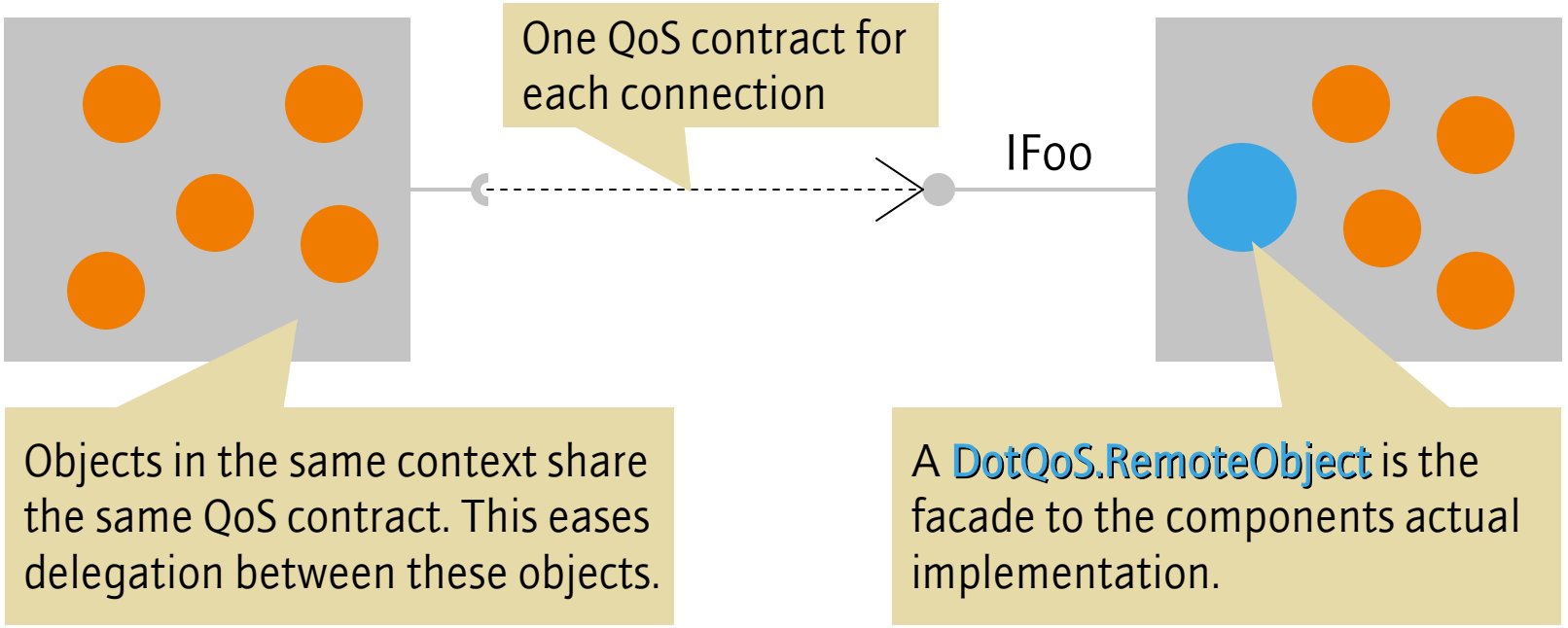
Inherits from MarshalByRefObject

More than one contract class can be specified.

© Torben Weis, Andreas Ulbrich, and Kurt Geihs

QoS Negotiation (1/3)

- Components as Contracting parties
  - Objects too fine-grained (especially on client side)
  - Group of objects which share a set of properties → QoS Context
  - .NET has great support for object contexts, plays well with remoting



© Torben Weis, Andreas Ulbrich, and Kurt Geihs

QoS Negotiation (2/3)

- Negotiation is application and environment dependent
  - Must be customizable
- Negotiation is orthogonal to application
  - Separate concerns
  - Several different negotiation strategies for the same component
- Special class for negotiation: **FrameContract**
  - Means to specify desired QoS Parameters
  - Responsible for contract activation
  - Default: accept all QoS requirements, fail on activation

© Torben Weis, Andreas Ulbrich, and Kurt Geihs

QoS Negotiation (3/3)

```
Server svr = ServerHome.Create();

#region qos specific
FrameContract fc = svr.GetFrameContract(observer);
License license = new License();
license.Invocations = 100;
fc.SetQoSForPort(typeof(IFoo), license);

Fc.Activate();
#endregion

svr.doBar();
svr.doBaz();
```

QoS Monitor

Specify QoS requirements.

Triggers configuration of .NET Remoting.

Implementation uses custom attributes of ServerImpl and License.

QoS negotiation can be driven by XML-config file. This reduces code complexity.

© Torben Weis, Andreas Ulbrich, and Kurt Geihs

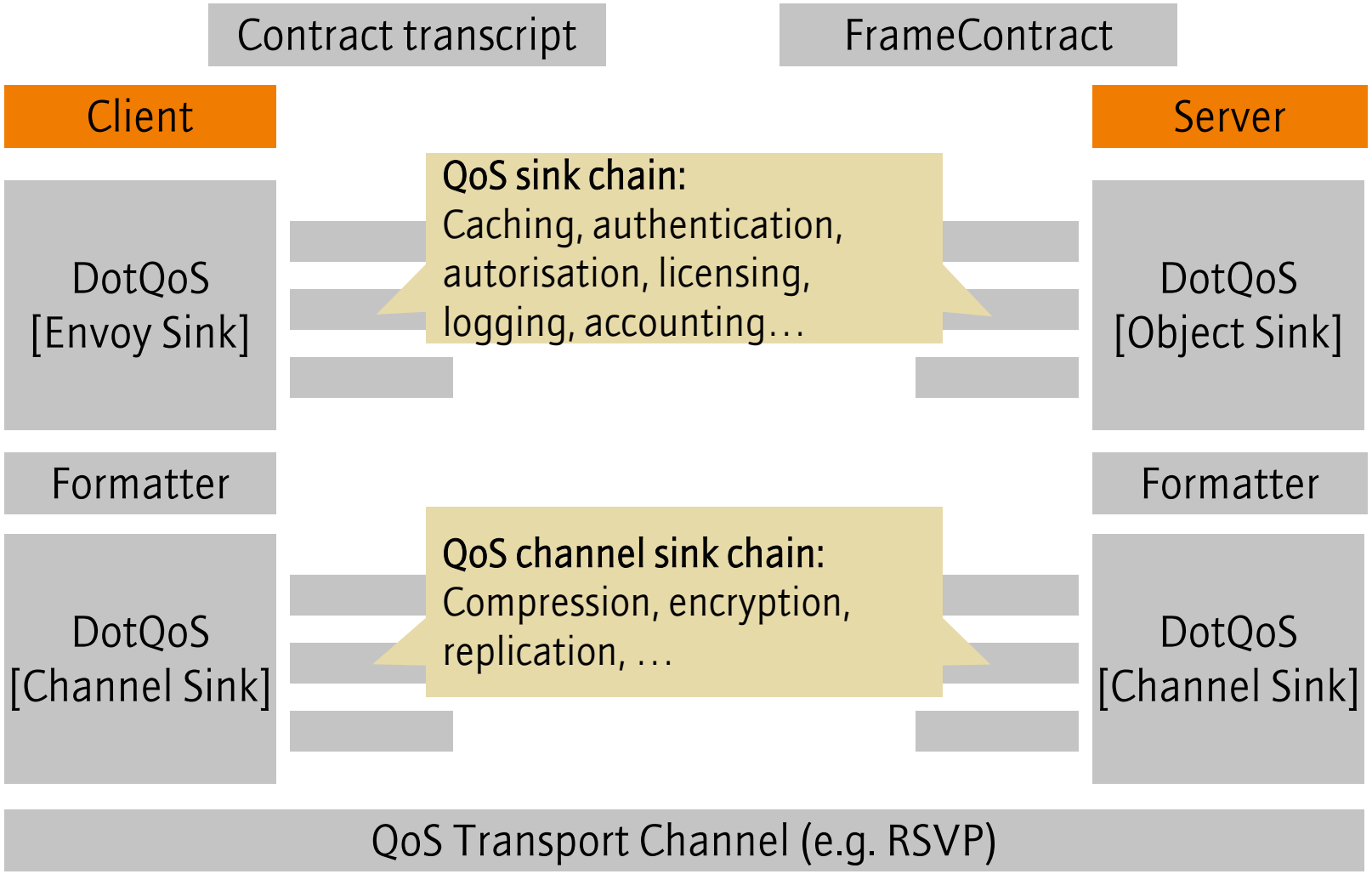


QoS Mechanisms (1/3)

- Everything in the message path can be a QoS mechanism
  - Sinks
  - Formatters
  - Channels
- Problem: Message path is rather static
  - But need flexible, dynamic integration of QoS mechanisms
  - Depends on QoS parameters, …
  - Client component, …
  - Resource availability
- Solution: proxy sinks determine correct sinks at runtime

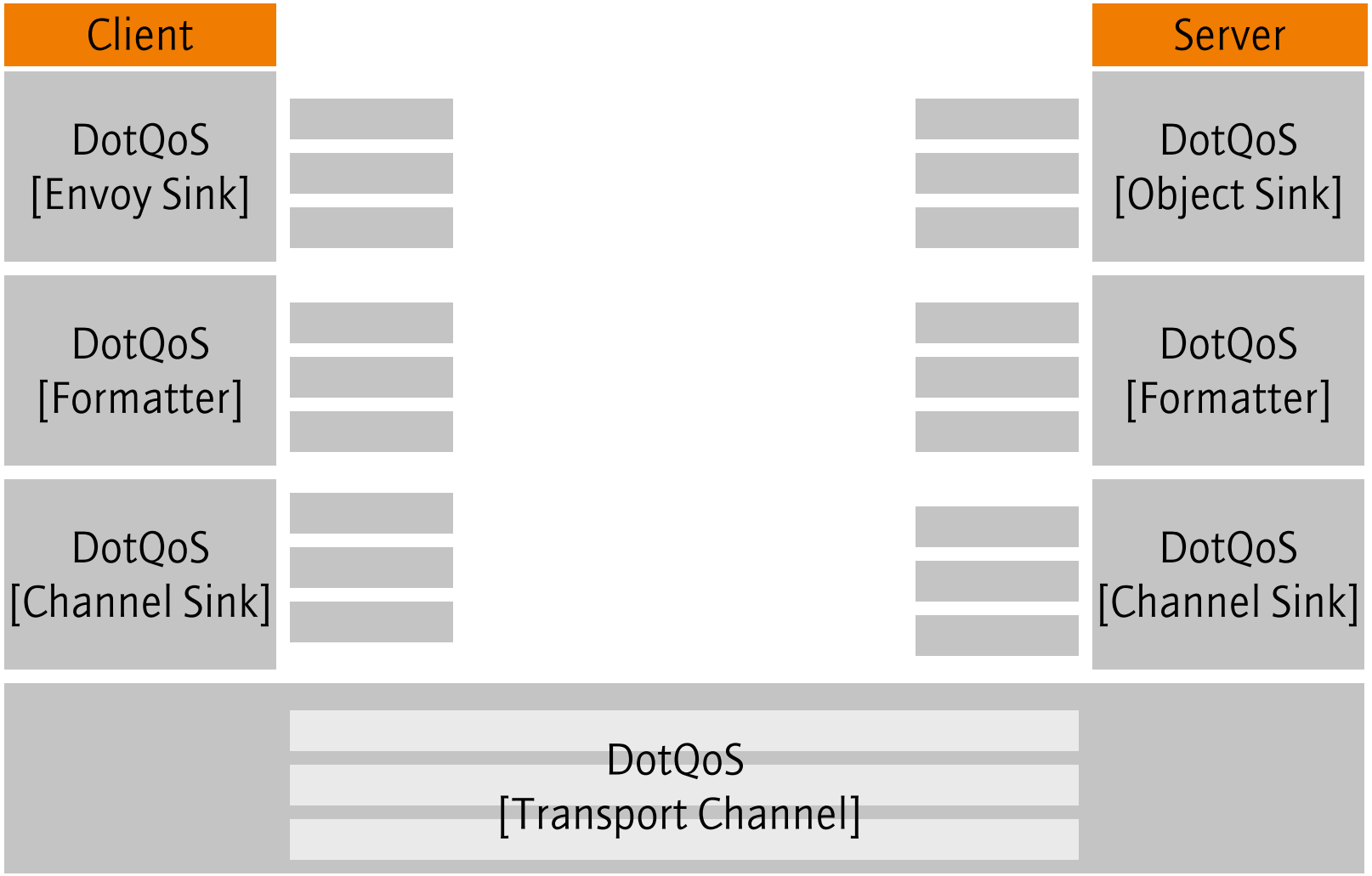
© Torben Weis, Andreas Ulbrich, and Kurt Geihs

QoS Mechanisms (2/3)



© Torben Weis, Andreas Ulbrich, and Kurt Geihs

Future Versions of DotQoS?



© Torben Weis, Andreas Ulbrich, and Kurt Geihs



QoS Mechanisms (3/3)

- Which QoS mechanism?
  - Depends on: QoS Requirements, available mechanisms, applications, client components, ...
  - DotQoS consults FrameContract:

    FrameContract.GetClientResources(...)

    FrameContract.GetServerResources(...)
- In which order?
  - Depends on mechanisms
  - Not every combination possible
  - Constraints can be provided using custom attributes

© Torben Weis, Andreas Ulbrich, and Kurt Geihs



QoS mechanism order

```
public class LicenseInterceptor : QoSInterceptorAttribute {}

[LicenseInterceptor(
    Position=QoSPosition.Head,
    After=AuthenticationInterceptor)]
public class LicenseServerSinkProvider : DotQoS.IQoSSinkProvider
{
    // factory class
}

public class LicenseServerSink : IMessageSink
{
    // sink implementation
}
```


© Torben Weis, Andreas Ulbrich, and Kurt Geihs



Summary

- QoS and non-functional Properties
  - Frequent requirement
  - Requires substantial middleware support → but hard to integrate
- DotQoS
  - Provides QoS handling for .NET
  - Specification, negotiation, and mechanisms for multiple categories
- DotQoS is non-intrusive
  - No changes to the middleware → DotQoS is an add-on
  - Relies entirely on platform features
  - Eases distribution and maintainance
- Tool support is helpful: Kase

© Torben Weis, Andreas Ulbrich, and Kurt Geihs



Experiences

- QoS integration fostered by .NET features
  - Easy compared to CORBA
  - Open and modular middleware
  - Custom attributes and reflection
  - Context-bound objects
- Downsides of .NET
  - Hard-wired dispatcher (CORBA has portable object adapters POA)
  - Advanced stuff fairly is undocumented. (Dig through Rotor sources if you really want to know how .NET Remoting works.)
  - No hard real-time (.NET just wasn't designed for this.)

© Torben Weis, Andreas Ulbrich, and Kurt Geihs