Describing Software Components with Parametric Contracts

Ralf Reussner (reussner@ipd.uka.de)
Chair Software Design and Quality
Institute for Program Structures and Data Organization (IPD)
Fakultät für Informatik, Universität Karlsruhe (TH),
Karlsruhe Institute of Technology
- Design-by-Contract for Software-Components
- Specification of the relation between provides and requires-interface(s) (and the usage profile and execution environment)
Overview

- Contracts and Parametric Contracts
- Protocol Adaptation with Parametric Contracts
- Prediction of Non-Functional Properties with Parametric Contracts
  - Reliability
  - Performance
- Conclusions
What is a component?

▪ “A component is a contractually specified building unit of software which can be readily composed or deployed.”
  – “readily composed or deployed”:
    • without having to understand the interna as a human
    • these are the two main things to be done with components
  – not necessarily “black-box”: Information on interna can be available to tools.

▪ “Components are for composition, much beyond is unclear…” (Clemens Szyperski)
Different Abstraction of Components

<table>
<thead>
<tr>
<th>Type</th>
<th>Implementation</th>
<th>Deployment</th>
<th>Runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>CT Type</td>
<td>QoS (i, es, up, d)</td>
<td>QoS (es, up, d)</td>
<td>QoS (up)</td>
</tr>
<tr>
<td>IID Implementation Instance Description</td>
<td>FP (es)</td>
<td>FP (es)</td>
<td>FP ()</td>
</tr>
<tr>
<td>DID Deployment Instance Description</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Not considered within the Palladio ComponentModel

Contracts – Protocols - Quality - Conclusions
Contractual Use of Methods

- **Design-by-contract (B. Meyer, 1992)**
  
  “Supplier guarantees post-condition if pre-condition is fulfilled by client.”

  [Bertrand Meyer, Object Oriented Software Construction, Prentice-Hall, 1997 2nd ed.]

```
CustomerRecord getCustomerRecord (String ID)
```

Post-condition                        Pre-condition
## Design-by-Contract (Bertrand Meyer, 1992)

<table>
<thead>
<tr>
<th>Client</th>
<th>Obligations</th>
<th>Benefits</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Satisfy precondition: (attached database)</td>
<td>From postcondition: Get customer returned or null</td>
</tr>
<tr>
<td>Supplier</td>
<td>Satisfy postcondition: Get right customer or return null.</td>
<td>From Precondition: Handle queries only if database is attached.</td>
</tr>
</tbody>
</table>
Benefits of Contracts

- Developing less error-prone software through:
  - Better documentation
  - Reduced error-handling code
  - Clear responsibilities
  - Results of run-time checking as a debugging aid
Contractual Use of Components

Component

Required Interface

Component'

Provided Interface

Contracts - Protocols - Quality - Conclusions
Use of Components

There are two different times of component usage:

1. At system run-time (== component run-time): call of component services
2. At system design-time or system configuration time: plugging in of components
Contractual Use of Components

- Design-by-contract (B. Meyer, 1992)
  - “Supplier guarantees post-condition if pre-condition is fulfilled by client.”

- Contractual Use of Components
  - (at system design-/configuration-time)
  - “Component guarantees provided services (as described in the provides-interface) if the environment offers all required services (as described in the requires-interface).”
Contractual Use of Components

- **Pre-condition**
  - Required services
  - Information concerning required services (sequences, QoS)

- **Post-condition**
  - Offered services
  - Information concerning offered services (sequences, QoS)

- **Invariant**
  - Relation between provided and required services (as this is independent from the usage context)
Contractual use of Components

- Design-by-contract (B. Meyer, 1992)
  “The service supplier guarantees the post-condition, if the client guarantees the precondition of the service.”

- Contractual Use of Components
  (at system-(re-) configuration time)
  “The component offers the provided services (as specified in the provides interfaces), if the environment guarantees the required services (as specified in the requires interfaces).”
Contractual Use of Components

- Contractual use corresponds to interoperability checks: $R_C \subseteq P_C$.
- Static checks of practical relevance

![Diagram](image)

**Contracts** – Protocols - Quality - Conclusions

Ralf Reussner, FLACOS, Oslo 2007
Consequences (1)

- Interoperability checks during design- or reconfiguration time are possible by checking the pre-condition against the post-condition of the environment
Problem:
Three external Influences on Quality (Performance / Reliability)

Usage Model

Performance / Reliability?

... if (a>b) c = a; ...

External Services

Execution Environment

Contracts – Protocols - Quality - Conclusions
Consequences (2)

- But:
  - Fixed pre- and post-condition hinder adaptation
  - Pre- and post-condition are hard to fix at design-time
  - In particular, if non-functional properties are to be specified in interfaces.

Components and their contracts need to be adapted to context at deployment, which is after development!
Need for Parametric Contracts

- One single pair of pre- and postcondition insufficient:
  - restriction of reuse
  - need for transparent component adaptation
  - QoS-properties depend on component context and are not fixed.
Parametric Contracts for Components

Parametric Contract

Contracts – Protocols - Quality - Conclusions
Ralf Reussner, FLACOS, Oslo 2007
Parametric Contracts

- Link provides- and requires interface of the same component by bijective function $p: \{P\} \rightarrow \{R\}$
- Pre-condition computed out of post-condition (and vice versa)
- Mechanism of automatic component adaptation (besides adaptor generation)
Parametric Contracts for Components

- Computation of environment-dependent provides-interface
- Computation of use-dependent requires-interface (‘wp-calculus’)
- Fine-grain adaptation of large-grain components
Computation of Parametric Contracts

- Computation of context-dependent provides-interface:
  1. \( R_A = p_A(P_A) \)
  2. \( R_{A'} := R_A \cap P_B \)
  3. \( P_{A'} = p_A^{-1}(R_{A'}) \)

- Computation of context-dependent requires-interface:
  1. \( P_{B'} := R_A \cap P_B \)
  2. \( R_{B'} = p_B(P_{B'}) \)

Contracts – Protocols – Quality – Conclusions
Lifting the Design-by-Contract Principle to Software Components

Linking the provided and required services of the same component
Hierarchie of Interface Models

Contracts – Protocols – Quality – Conclusions

Ralf Reussner, FLACOS, Oslo 2007
Service Effect Specification: List of required services for each provided service

Service-Effect-Specifications

void foo () requires externalMethod1 externalMethod2 ...

{ }

...
A method $n$ is then and only then required, if there exists (at least) one method $m$ with $n \in \text{se}(m)$.

\[ + \text{Service-Effect-Specification} = \]

\[ \text{Provides-Interface} \]

\[ = \text{Service-Effect-Specification} + \]

\[ \text{Requires-Interface} \]

A method $n$ is then and only then provided, if all its required methods $\text{se}(n)$ are available.
Service Effect Specification: List of required call sequences of services for each provided service

provides-interface {
    void foo();
    int bar();
    ...
}

Service-Effect-Specifications

Component - P - R
Protocol-Adaptation with Parametric Contracts

Contracts – Protocols – Quality – Conclusions

Ralf Reussner, FLACOS, Oslo 2007
A call sequence \( s \) is then and only then required, if there exists (at least) one method \( m \) with \( s \in se(m) \).

A method \( n \) is then and only then provided, if all its required call sequences \( se(n) \) are available.

Contracts – Protocols - Quality - Conclusions
Service Effect Specification

<<Interface>>

interface1

service1(X : Integer,
Y : Collection) : Collection

<<provides>>

<<BasicComponent>>

Component1

<<VariableUsage>>

Z.BYTESIZE = 2000

Contracts – Protocols - Quality - Conclusions
public void handle(ExternalCallAction call) {
    Signature serviceToBeCalled = call.getCalledService_ExternalService();
    Interface requiredInterface = (Interface) serviceToBeCalled.eContainer();
    EList<VariableUsage> parametricParameterUsages = call.getParameterUsage_ExternalCallAction();
    AssemblyConnector foundAssemblyConnector = findAssemblyConnector(requiredInterface);
    if (foundAssemblyConnector == null) {
        logger.info("Found System External Call");
        String timeSpecification = getTimeSpecification(serviceToBeCalled);
        createInternalAction(timeSpecification, call);
    } else {
        logger.info("Found Assembly Connector");
        SeffVisitor nextVisitor = visitNextSeff(serviceToBeCalled, parametricParameterUsages, foundAssemblyConnector);
        storeOutputParametersToUsageContext(call, nextVisitor);
    }
}

private void createInternalAction(String timeSpecification, ExternalCallAction call) {
    ParametricResourceDemand demand = seffFactory.createParametricResourceDemand();
    demand.setSpecification(timeSpecification);
    demand.setUnit(
    demand.setRequiredResource_ParametricResourceDemand(getProcessingResourceType());
    InternalAction action = seffFactory.createInternalAction();
    action.getResourceDemand_Action().add(demand);
    // Add new internal action into control flow after external action
    action.setSuccessor_AbstractAction(call.getSuccessor_AbstractAction());
    action.setPredecessor_AbstractAction(call);
    ResourceDemandingBehaviour rdb = (ResourceDemandingBehaviour)call.eContainer();
    rdb.getSteps_Behaviour().add(action);
}
Service Effect Specification

<<BasicComponent>>
Component1

<<VariableUsage>>
Z.BYTESIZE = 2000

<<Interface>>
interface1
service1(X : Integer, Y : Collection) : Collection

<<provides>>
service1

<<InternalAction>>

<<GuardedBranchTransition>>
specification = "X.VALUE<0"

<<GuardedBranchTransition>>
specification = "X.VALUE>=0"

<<InternalAction>>

<<BranchAction>>

<<ParametricResourceDemand>>
name = "CPU"
specification = "Z.BYTESIZE * 10"

<<ParametricResourceDemand>>
name = "CPU"
specification = "Z.BYTESIZE * 2"

Contracts – Protocols - Quality - Conclusions
Service Effect Specification

Contracts – Protocols - Quality - Conclusions

Ralf Reussner, FLACOS, Oslo 2007

14/10/2007 33
Service Effect Specification

<<Interface>>
interface1
service1(X : Integer, Y : Collection) : Collection
<<provides>>

<<BasicComponent>>
Component1

<<VariableUsage>>
Z.BYTESIZE = 2000

<<SetVariableAction>>
service1.RETURN.NUMBER_OF_ELEMENTS = Y.NUMBER_OF_ELEMENTS / 3

<<GuardedBranchTransition>>
specification = "X.VALUE<0"
<<ProcessingResource>>
name = "CPU"

<<GuardedBranchTransition>>
specification = "X.VALUE>=0"
<<ProcessingResource>>
name = "CPU"

<<ParametricResourceDemand>>
specification = "Z.BYTESIZE * 10"
<<ProcessingResource>>
name = "CPU"

<<ParametricResourceDemand>>
specification = "Z.BYTESIZE * 2"
<<ProcessingResource>>
name = "CPU"

<<CollectionIteratorAction>>

<<ExternalCallAction>>
extCallPar.VALUE = Y.INNER.VALUE

<<VariableUsage>>

<<BranchAction>>
parameterName = "Y"

<<InternalAction>>

Contracts – Protocols - Quality - Conclusions

Ralf Reussner, FLACOS, Oslo 2007
Service Effect Specification

<<Interface>>
interface1
service1(X : Integer,
Y : Collection) : Collection

<<UsageContext>>
P(X.VALUE = -5) = 0.8

Contracts – Protocols - Quality - Conclusions
Service Effect Specification

<<Interface>>
interface1

service1(X : Integer,
Y : Collection) : Collection

<<UsageContext>>
P(X.VALUE = -5) = 0.8

<<ResourceDemandingSEFF>>
service1

<<BranchAction>>

<<InternalAction>>

<<GuardedBranchTransition>>
specification = 0.8

<<ParametricResourceDemand>>
specification = \(Z.BYTESIZE \times 10\)
nname = "CPU"

<<GuardedBranchTransition>>
specification = \(X.VALUE \geq 0\)

<<Parameter>>
parameterName = "Y"

<<CollectionIteratorAction>>

<<ExternalCallAction>>
extCallPar.VALUE = Y.INNER.VALUE

<<SetVariableAction>>

<<Parameter>>
parameterName = "Y"

<<BranchAction>>

<<InternalAction>>

<<GuardedBranchTransition>>
specification = \(X.VALUE = 0\)

<<ParametricResourceDemand>>
specification = \(Z.BYTESIZE \times 2\)
nname = "CPU"

<<VariableUsage>>
service1.RETURN.NUMBER_OF_ELEMENTS = Y.NUMBER_OF_ELEMENTS / 3

Contracts – Protocols - Quality - Conclusions
Service Effect Specification

interface1

\[ \text{service1}(X: \text{Integer}, \ Y: \text{Collection}) : \text{Collection} \]

\[ P(X.\text{VALUE} = -5) = 0.8 \]
\[ P(X.\text{VALUE} = 7) = 0.2 \]

Contracts – Protocols - Quality - Conclusions
Service Effect Specification

<<Interface>>
interface1

service1(X : Integer, Y : Collection) : Collection

<<ResourceDemandingSEFF>>

service1

<<InternalAction>>

<<GuardedBranchTransition>>
specification = 0.8
<<ProcessingResource>>
name = "CPU"

<<GuardedBranchTransition>>
specification = 0.2
<<ProcessingResource>>
name = "CPU"

<<CollectionIteratorAction>>

<<ExternalCallAction>>
extCallPar.VALUE = Y.INNER.VALUE

<<SetVariableAction>>

service1.RETURN.NUMBER_OF_ELEMENTS = Y.NUMBER_OF_ELEMENTS / 3

Contracts – Protocols - Quality - Conclusions
Service Effect Specification

<<ResourceDemandingSEFF>>

service1

<<BranchAction>>

<<GuardedBranchTransition>>
specification = 0.8

<<GuardedBranchTransition>>
specification = 0.2

<<ParametricResourceDemand>>
name = "CPU"

<<ParametricResourceDemand>>
specification = _Z.BYTESIZE * 10*

<<ParametricResourceDemand>>
specification = _Z.BYTESIZE * 2*

<<ProcessingResource>>
name = "CPU"

<<CollectionIteratorAction>>

<<ExternalCallAction>>

<<SetVariableAction>>

<<VariableUsage>>

contracts – protocols - quality - conclusions

Ralf Reussner, FLACOS, Oslo 2007

<<UsageContext>>
P(X.VALUE = -5) = 0.8
Service Effect Specification

Service Effect Specification

<<Interface>>

interface1

service1(X : Integer,
Y : Collection) : Collection

<<UsageContext>>

P(X.VALUE = -5) = 0.8
P(X.VALUE = 7) = 0.2

P(Z.BYTESIZE = 2000) = 1.0

<<ResourceDemandingSEFF>>

service1

<<BranchAction>>

<<CollectionIteratorAction>>

<<Parameter>>

parameterName = "Y"

<<SetVariableAction>>

service1.RETURN.NUMBER_OF_ELEMENTS = Y.NUMBER_OF_ELEMENTS / 3

Contracts – Protocols - Quality - Conclusions
<<ResourceDemandingSEFF>>

service1

<<InternalAction>>

<<GuardedBranchTransition>>

specification = 0.8
<<ProcessingResource>>
name = "CPU"

<<InternalAction>>

<<GuardedBranchTransition>>

specification = 0.2
<<ProcessingResource>>
name = "CPU"

<<ParametricResourceDemand>>

specification = 20000
<<ProcessingResource>>
name = "CPU"

<<ParametricResourceDemand>>

specification = 4000
<<ProcessingResource>>
name = "CPU"

<<CollectionIteratorAction>>

<<ExternalCallAction>>

extCallPar.VALUE = Y.INNER.VALUE

<<SetVariableAction>>

service1.RETURN.NUMBER_OF_ELEMENTS = Y.NUMBER_OF_ELEMENTS / 3

<<Interface>>

interface1

service1(X : Integer,
Y : Collection) : Collection

Contracts – Protocols - Quality - Conclusions

P(X.VALUE = -5) = 0.8
Service Effect Specification

<<Interface>>
interface1
service1(X : Integer, Y : Collection) : Collection

<<ResourceDemandingSEFF>>
service1

<<ParametricResourceDemand>>
name = "CPU"
specification = 20000

<<ParametricResourceDemand>>
name = "CPU"
specification = 4000

Contracts – Protocols - Quality - Conclusions

<<UsageContext>>
P(X.VALUE = -5) = 0.8
Service Effect Specification

<<Interface>>
interface1

service1(X : Integer,
        Y : Collection) : Collection

<<UsageContext>>
P(X.VALUE = -5) = 0.8
P(X.VALUE = 7) = 0.2
P(Y.INNER.VALUE = "foo") = 1.0
P(Z.BYTESIZE = 2000) = 1.0

<<ResourceDemandingSEFF>>

service1

<<BranchAction>>

<<GuardedBranchTransition>>
specification = 0.8
<<ParametricResourceDemand>>
specification = 20000
<<ProcessingResource>>
name = "CPU"

<<GuardedBranchTransition>>
specification = 0.2
<<ParametricResourceDemand>>
specification = 4000
<<ProcessingResource>>
name = "CPU"

<<CollectionIteratorAction>>

<<ExternalCallAction>>
extCallPar.VALUE = "foo"

<<SetVariableAction>>

<<Parameter>>
parameterName = "Y"

<<VariableUsage>>
service1.RETURN.NUMBER_OF_ELEMENTS = Y.NUMBER_OF_ELEMENTS / 3

<<Interface>>
interface1

service1(X : Integer,
        Y : Collection) : Collection

<<UsageContext>>
P(X.VALUE = -5) = 0.8
P(X.VALUE = 7) = 0.2
P(Y.INNER.VALUE = "foo") = 1.0
P(Z.BYTESIZE = 2000) = 1.0

Contracts – Protocols - Quality - Conclusions

Ralf Reussner, FLACOS, Oslo 2007
### Service Effect Specification

#### Interface

**interface1**

service1(X : Integer,
         Y : Collection) : Collection

#### Resource Demanding SEFF (service1)

- **<<BranchAction>>**
- **<<CollectionIteratorAction>>**
- **<<InternalAction>>**
- **<<ExternalCallAction>>**
- **<<VariableUsage>>**
- **<<SetVariableAction>>**

#### Resource Demanding Parameters

- **<<ParametricResourceDemand>>**
  - specification = 20000
  - ProcessingResource
  - name = "CPU"

- **<<ParametricResourceDemand>>**
  - specification = 4000
  - ProcessingResource
  - name = "CPU"

- **<<VariableUsage>>**
  - extCallPar.VALUE = "foo"

- **<<SetVariableAction>>**
  - service1.RETURN.NUMBER_OF_ELEMENTS = Y.NUMBER_OF_ELEMENTS / 3

#### Guarded Branch Transitions

- **<<GuardedBranchTransition>>**
  - specification = 0.8

- **<<GuardedBranchTransition>>**
  - specification = 0.2

#### Probability

- **P(X.VALUE = -5) = 0.8**

---

Ralf Reussner, FLACOS, Oslo 2007

<<UsageContext>>

Contracts – Protocols - **Quality** - Conclusions
Service Effect Specification

<<Interface>>
interface1
  service1(X : Integer, Y : Collection) : Collection

<<UsageContext>>
P(X.VALUE = -5) = 0.8
P(X.VALUE = 7) = 0.2
P(Y.NoE = 30) = 0.1
P(Y.NoE = 60) = 0.9
P(Y.INNER.VALUE = "foo") = 1.0
P(Z.BYTESIZE = 2000) = 1.0

<<BranchAction>>

<<CollectionIteratorAction>>

<<Parameter>>
  parameterName = "Y"

<<ExternalCallAction>>
  extCallPar.VALUE = "foo"

<<SetVariableAction>>
  service1.RETURN.NUMBER_OF_ELEMENTS = INT_PMF[(10; 0.1)(20;0.9)]

<<GuardedBranchTransition>>
  specification = 0.8

<<GuardedBranchTransition>>
  specification = 0.2

<<ParametricResourceDemand>>
  specification = 20000
  name = "CPU"

<<ParametricResourceDemand>>
  specification = 4000
  name = "CPU"

Contracts – Protocols - Quality - Conclusions

Ralf Reussner, FLACOS, Oslo 2007
Service Effect Specification

<<Interface>>
interface1

service1(X : Integer, Y : Collection) : Collection

<<GuardedBranchTransition>>
specification = 0.8

<<GuardedBranchTransition>>
specification = 0.2

<<ParametricResourceDemand>>
specification = 20000
<<ProcessingResource>>
name = "CPU"

<<InternalAction>>

<<ParametricResourceDemand>>
specification = 4000
<<ProcessingResource>>
name = "CPU"

<<CollectionIteratorAction>>

<<ExternalCallAction>>

<<SetVariableAction>>

service1.RETURN.NUMBER_OF_ELEMENTS = INT_PMF[(10; 0.1)(20;0.9)]

P(X.VALUE = -5) = 0.8
P(X.VALUE = 7) = 0.2
P(Y.NoE = 30) = 0.1
P(Y.NoE = 60) = 0.9
P(Y.INNER.VALUE = "foo") = 1.0
P(Z.BYTESIZE = 2000) = 1.0

P(X.VALUE = -2) = 0.3
P(X.VALUE = 9) = 0.7
P(Y.NoE = 15) = 0.2
P(Y.NoE = 18) = 0.8
P(Y.INNER.VALUE = "bar") = 1.0
P(Z.BYTESIZE = 300) = 1.0

Contracts – Protocols - Quality - Conclusions
Service Effect Specification

<<Interface>>
interface1

service1(X : Integer,
Y : Collection) : Collection

<<UsageContext>>
P(X.VALUE = -5) = 0.8
P(X.VALUE = 7) = 0.2
P(Y.NoE = 30) = 0.1
P(Y.NoE = 60) = 0.9
P(Y.INNER.VALUE = "foo") = 1.0
P(Z.BYTESIZE = 2000) = 1.0

<<UsageContext>>
P(X.VALUE = -2) = 0.3
P(X.VALUE = 9) = 0.7
P(Y.NoE = 15) = 0.2
P(Y.NoE = 18) = 0.8
P(Y.INNER.VALUE = "bar") = 1.0
P(Z.BYTESIZE = 300) = 1.0

<<ResourceDemandingSEFF>>

service1

<<GuardedBranchTransition>>
specification = 0.3
<<ProcessingResource>>
name = "CPU"

<<GuardedBranchTransition>>
specification = 0.7
<<ProcessingResource>>
name = "CPU"

<<ParametricResourceDemand>>
specification = 3000
<<ProcessingResource>>
name = "CPU"

<<ParametricResourceDemand>>
specification = 600
<<ProcessingResource>>
name = "CPU"

<<CollectionIteratorAction>>

<<ExternalCallAction>>
extCallPar.VALUE = "bar"

<<SetVariableAction>>

<<BranchAction>>

<<Parameter>>
parameterName = "Y"

<<VariableUsage>>

Contracts – Protocols - Quality - Conclusions
Palladio Component Model

Contracts – Protocols - Quality - Conclusions

Ralf Reussner, FLACOS, Oslo 2007
Stochastic Regular Expression

\[ P ::= a \mid P \cdot Q \mid P +_\pi Q \mid P^{*(l)} \]

- **Sequence**
- **Loop**
  - Iterations \((l)\)
- **Symbol**
- **Alternative**
  - Probability \(\pi, 1-\pi\)
<<ExternalCallAction>>
checkEnvironment()
Palladio Component Model

Contracts – Protocols – Quality – Conclusions

Ralf Reussner, FLACOS, Oslo 2007
Model Solution

$$f_P \cdot Q(t) = (f_P \ast f_q)(t)$$
Model Solution

\[ f_{P+\pi Q}(t) = \pi f_P(t) + (1 - \pi) f_Q(t) \]

Contracts – Protocols - **Quality** - Conclusions
Model Solution

\[ f_{P*}(t) = \sum_{i=0}^{N} p_l(i) \left( \bigotimes_{j=1}^{i} f_P \right)(t) \]

Contracts – Protocols - Quality - Conclusions

Ralf Reussner, FLACOS, Oslo 2007
MediaStore - Architecture

<<ResourceContainer>>

Contradict - Protocols - Quality - Conclusions

Ralf Reussner, FLACOS, Oslo 2007
Download - Use Case

Contracts – Protocols - Quality - Conclusions

Ralf Reussner, FLACOS, Oslo 2007
Case Study: Usage Profiles

Parameter Characterisation

Contracts - Protocols - Quality - Conclusions
Results

Contracts – Protocols - Quality - Conclusions

Ralf Reussner, FLACOS, Oslo 2007
Results

Cumulative Probability

Usage Profile 1
(Prediction)

Usage Profile 1
(Measurement)

Response Time (Seconds)

Contracts – Protocols - Quality - Conclusions

Ralf Reussner, FLACOS, Oslo 2007
Model-based Prediction of Quality

Executable Software

Software Design Model

Annotated Software Design Model

Analysis Results

Analysis Modell

UML, ADL, ...

Transformation (MDD)

Estimation Measurement

UML Performance Profile, QML, ...

Analysis / Simulation

Queueing models
Stochastic Petri-Nets,
Stochastic Process Algebra,
...

Response time
Throughput, Utilisation,
...

Automated by Tools

Contracts – Protocols - Quality - Conclusions

Ralf Reussner, FLACOS, Oslo 2007
Scientific Approach

Measured Quality

Predicted Quality

Comparison

Interpretation

Acceptance / rejection of abstract model

Abstraction

Software

Modell of Software (mit Annotationen)

Prediction

Improvement / Extension

Measurement

SDQ

Engineering? - Components - Contracts - Protocols - Quality - Conclusions
Summary

- **Contractual Use of Components**
  - Requires-interfaces as pre-conditions
  - Provides-interfaces as post-conditions
  - Enabling (static) interoperability checks

- **Parametric Contracts**
  - Compute pre-condition in dependency of post-condition and vice versa
  - Prediction of properties in non-cyclic architectures by propagating interface information
  - Formal models of parametric contracts for predicting quality properties are created in a process oriented toward natural sciences.