Verteidigung

Diplomarbeit

Thema: Evaluation des Projekts „Quality Objects“

Sven Harazim
sh17@inf.tu-dresden.de

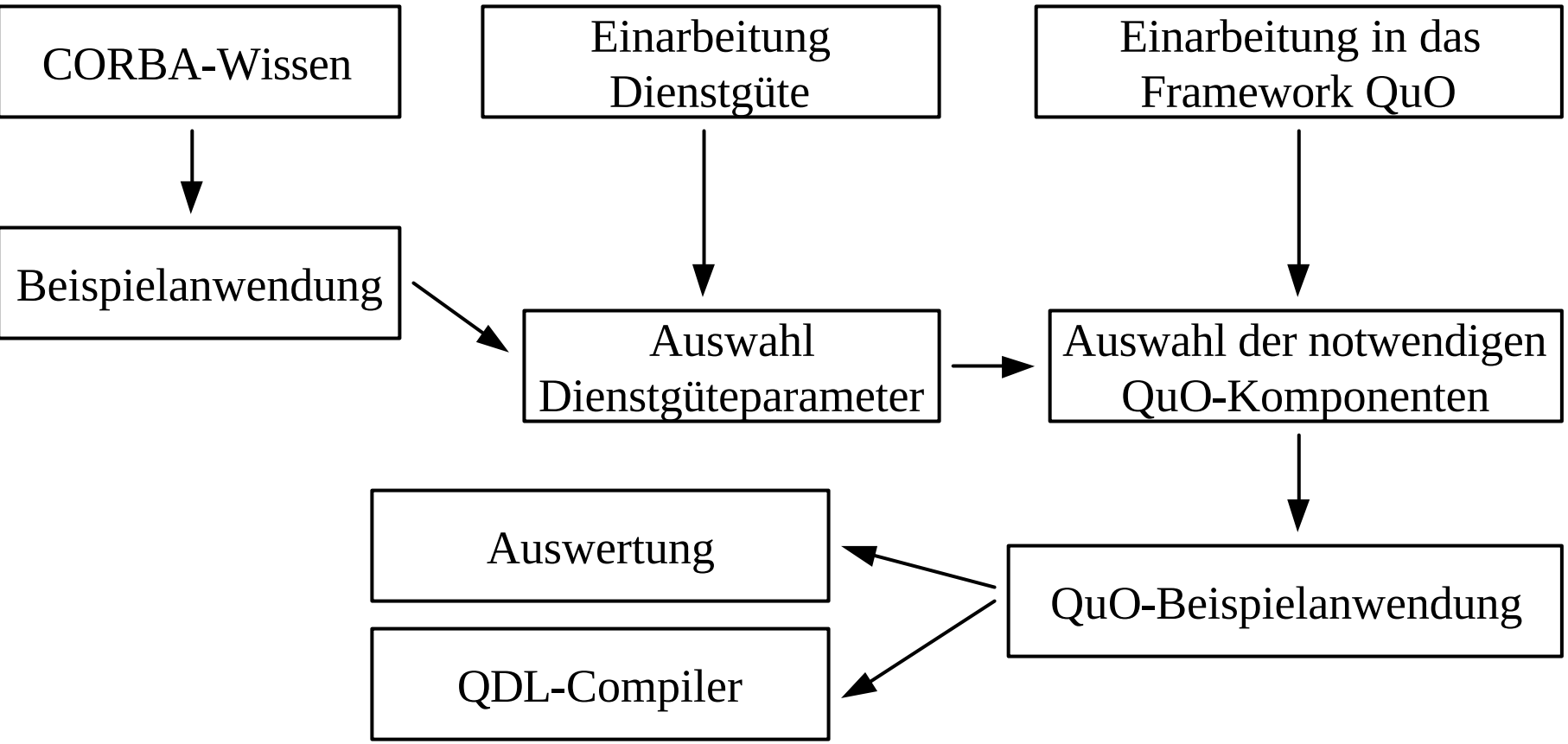
Gliederung

- Aufgabenstellung
- Vorgehensweise
- Aufbau und Anwendung des Frameworks Quality Objects (QuO)
- Ergebnisse
- Zusammenfassung

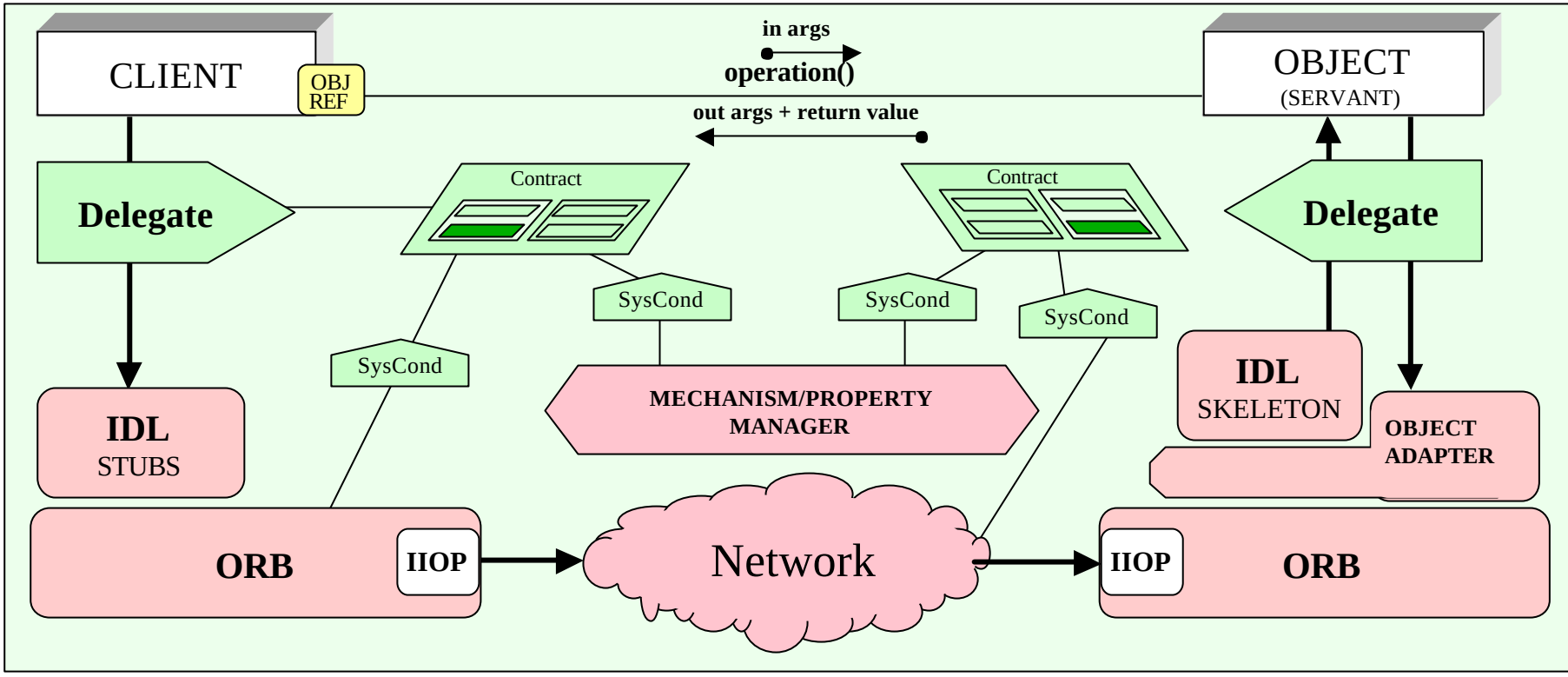
Aufgabenstellung

- Aufführung von Qualitätsanforderungen und Adaptionsmöglichkeiten bei deren Nichteinhaltung
- Umsetzung ausgewählter Anforderungen mittels des Frameworks Quality Objects in einer Beispiel-anwendung
- Bewertung der Durchsetzung der Anforderungen und der Anwendungsmöglichkeiten der QDL
- Analyse des vom QDL-Compiler generierten Quelltextes

Vorgehensweise bei der Lösung der Aufgaben



Zusammenspiel von Anwendung und QuO



Weitere Bestandteile von QuO

- Quality Description Languages – QDL
  - Contract Description Language - CDL
  - Aspect Structure Language - ASL
- QDL-Compiler
- QuO-Laufzeit-Kernel
- Qoskets
- Ressource Status Service - RSS

Contract Description Language CDL

```
contract MyContract  (
    callback ExampleQosket::MyCallback myCallback
) {
 syscond  quo::ValueSC quo_sc::ValueSCImpl myValue

 region Normal ((long)myValue <= 50)  {}
 region High ( (long)myValue > 50) {}

 transition Normal -> High  {
    synchronous {
         myCallback.valueIsHigh();
}}};
```

Aspect Structure Language ASL

```
behavior ExampleQosket() {
  qosket bqosket::ExampleQosket::MyDelegateQosket myQosket;
  remote_object Remote::Client remote;

  void Remote::Client::newValue (in long val) {

    before METHODENTRY {
      region Normal {
        myQosket.incSysCondVal(1);
      }
    }

    inplaceof METHODCALL {
      region Normal {
        remote.newValue(myQosket.getSysCondVal() + val);
      }
      region High {
        remote.newValue(myQosket.getSysCondVal() - val);
      }
}}};
```

ASL-Schablonen

```
template behavior<method METH> CounterTemplate () {

 qosket bqosket::ExampleQosket::MyDelegateQosket myQosket;

  METH.signature {

    before METHODENTRY {

      region Normal {
        myQosket.incSysCondVal(1);
      }

    }
  }
};
```

Entwicklerrollen in QuO

- Anwendungs-Entwickler
- Qosket-Entwickler
- Entwickler von adaptiven Objekten
- Mechanismus-Entwickler
- Installateur
- System-Administrator

Beispielanwendung: Börsenticker

- Client abonniert bei einem Server einen oder mehrere Aktienkurse
- Verschiedene Dienstgüteanforderungen an die Anwendung:
  - Aktualisierungsrate
  - Priorität der Aktualisierung
  - Maximale Auslieferungsverzögerung
  - Übertragungskapazität
  - Sicherheit

Verwendete Qoskets

- BW-Select-Qosket
  - Wählt einen Server in Abhängigkeit der Auslastung seiner Netzanbindung aus
- RSS-Qosket
  - Überwacht Priorität und Auslieferungs-verzögerung einer Kursaktualisierung
- Updaterate-Qosket
  - Überwacht geforderte Aktualisierungsrate eines Aktienkurses

Bewertung BW-Select-Qosket

- Erfüllt die Anforderung der Auswahl des geeignetsten Servers
- Alle Server müssen zum Initialisierungs-zeitpunkt des Qoskets verfügbar sein
- Keine dynamische Umkonfiguration des Qoskets möglich
- Um Bandbreitenmessungen durchführen zu können, muss die Netzwerktopologie bekannt sein.

Bewertung RSS-Qosket

- Führt Messung von Methodenaufrufzeiten durch
- Bietet Informationen über erwartete Aufrufverzögerung
- Weitreichende Eingriffe in den Code der Geschäftsanwendung notwendig
- Zusätzliche Verwaltung von Objektreferenzen in der Anwendung
- Transparenz leidet
- Auch hier steigender administrativer Aufwand bei steigendem Verteilungsgrad

Bewertung Updaterate-Qosket

- Prinzipiell erfolgt die Durchsetzung der geforderten Aktualisierungsrate
- Ab zwei Instanzen dieses Qoskets in einer Anwendung tritt ein Fehler im Framework auf
- Aktive Regionen des Contracts werden dann nicht mehr korrekt bestimmt
- Grund: Fehlerhaft generierte Contract-Klasse

Allgemeine Bewertung

- Hohe Anzahl von zusätzlichen CORBA-Objekten in QuO
- Durch die mögliche Verteilung ist steigende Anzahl von entfernten Methodenaufrufen zu verzeichnen.
- Beschreibungssprachen sind einfach zu erlernen und decken die meisten möglichen Anwendungsfälle ab.

QDL-Compiler

- Klassischer Zwei-Phasen-Compiler
- Frondends sind aus Syntaxbeschreibungen automatisch generierte Parser und Scanner
- Backends erzeugen gewünschten Zielcode
- Generierte Klassen erweitern vorhandene Klassen des Frameworks
- Es erfolgt keine Optimierung

Zusammenfassung

- Durchsetzung von quantifizierbaren Dienstgüteanforderungen möglich
- Sprachen bieten ausreichende Beschreibungsmöglichkeiten für die meisten Anwendungsfälle
- Nur für verteilte Anwendungen mit statischen Systemstrukturen geeignet
- QDL-Compiler folgt den Prinzipien des allgemeinen Compilerbaus

Ausblick

- Verwendung von RSVP, dem Protokoll zur Reservierung von Ressourcen auf der Übertragungsstrecke
- Erweiterung von QuO-Anwendung um Fehlertoleranz mittels AQuA (Adaptive Quality of Service for Availability)
- Alternativen zum RSS