Diplomarbeit

Entwicklung einer Laufzeitumgebung für Komponenten mit Ressourcenanforderungen

bearbeitet von

Brit Engel

geboren am 15. Dezember 1976 in Perleberg

Technischen Universität Dresden

Fakultät Informatik
Institut für Software- und Multimediatechnik
Lehrstuhl Softwaretechnologie

Betreuerin: Dipl.- Softwaretech. Simone Röttger
Hochschullehrer: Prof. Dr. rer. nat. habil. Heinrich Hußmann

Eingereicht am 30.04.2003

Inhaltsverzeichnis

Kapitel 1

Einleitung

Bei der Entwicklung von Anwendungen steht die Erfüllung von funktionalen Anforderungen im Vordergrund. Diese beschreiben, welche Aufgaben von dem System zu realisieren sind. An die Ausführung des Dienstes werden aber auch nichtfunktionale Anforderungen gestellt. Dabei wird spezifiziert, wie gut die Aufgaben erfüllt werden sollen.[6] Mit der Verarbeitung dieser als Dienstgüte (*Quality of Service*, *QoS*) bezeichneten Merkmale wird sich im Rahmen dieser Arbeit auseinander gesetzt. Es sollen die Eigenschaften eines Dienstes, wie zum Beispiel Daten – und Fehlerraten, überwacht und deren Einhaltung garantiert werden.

Die Spezifikation der *QoS* – Eigenschaften kann mit Hilfe der *Component Quality Modelling Language* (*CQML*,[6]) erfolgen. Diese von Agedal entwickelte deklarative Sprache bietet die Möglichkeit der Beschreibung von Anforderungen, die von einer Komponente an andere Komponenten gestellt werden. In dieser Arbeit wird die Erweiterung dieser Beschreibungssprache ($CQML^+$,[24]) verwendet, welche zusätzlich eine Spezifikation der Ressourcenanforderungen enthält.

Zur Umsetzung der *QoS* – Eigenschaften wird im Rahmen des *COMQUAD* – Projektes [4] ein Container entwickelt. Dieser enthält eine Ressourcen – Verwaltung [22], die die Betriebsmittel reserviert und einer Verarbeitung zur Verfügung stellt. Die Verwaltung und Durchsetzung der nichtfunktionalen Anforderungen ist die Aufgabe einer Komponenten – Verwaltung, die im Rahmen dieser Arbeit zu entwerfen und prototypisch zu implementieren ist. Der Aufbau der in ihr zu verarbeitenden Anwendungen orientiert sich am Komponentenmodell der *Enterprise Java Beans* (*EJB*) – Spezifikation [2]. Für jede Komponente sind *QoS* – Eigenschaften zu verwalten.

Um die Einhaltung der nichtfunktionalen Anforderungen zu gewährleisten, müssen durch die Laufzeitumgebung Verträge [8] zwischen den kommunizierenden Komponenten ausgehandelt und die benötigten Betriebsmittel zur Verfügung gestellt werden. Für die Reservierung der Betriebsmittel schließt die Komponenten – Verwaltung Verträge mit der Ressourcen – Verwaltung ab.

Ziel der Arbeit ist es, Konzepte zur Verwaltung der Komponenten und ihrer *QoS* – Eigenschaften, sowie zur Aushandlung und Durchsetzung der Verträge zu entwickeln. Die Realisierung dieser Mechanismen soll für die Komponenten transparent erfolgen. Es werden daher die Einsatzmöglichkeiten von *Aspektorientierter Programmierung* (*AOP*, [27][1]) zur Integration der Funktionalität der Komponenten – Verwaltung und zur modularen Gestaltung der Laufzeitumgebung untersucht.

Kapitelvorschau

Im folgenden Kapitel wird das Gesamtkonzept des Containers zur Verwaltung der Komponenten und *QoS* – Eigenschaften vorgestellt. Dabei werden die Technologien, die in die Entwicklung der Komponenten – Verwaltung einfließen beziehungsweise von dieser verwendet werden sollen, beschrieben.

Die Konzepte zur Verarbeitung der *QoS* – Eigenschaften und somit zur Gestaltung der Komponenten – Verwaltung werden im dritten Kapitel diskutiert. Es werden verschiedene Lösungsansätze präsentiert und daraus ergebende Entwurfsentscheidungen abgeleitet.

Im Kapitel 4 erfolgt die Umsetzung des Entwurfs. Es wird ein Vorschlag zur Implementierung der Komponenten – Verwaltung mit *Java* unter Verwendung von *AspectJ* sowie eine prototypische Implementierung vorgestellt.

Abschließend werden im Kapitel 5, die während der Arbeit ermittelten Erkenntnisse zusammengefasst. Die Probleme, die sich aus der Arbeit mit *CQML*$^+$ ergaben, werden aufgezeigt sowie Vorschläge zur Erweiterung der entwickelten Konzepte unterbreitet.

Konventionen

Folgende Konventionen zur Schriftgestaltung wurden in dieser Arbeit angewandt:

- `Quellcode` einschließlich `Klassen-, Aspekt-, Attribut-` und `Methodennamen` werden durch `Schreibmaschinentext` gekennzeichnet. Dabei werden bei Methoden gegebenenfalls die abschließenden Klammern und Parameter weggelassen.
- *Verzeichnisse* und *Dateinamen* sind *Kursiv* dargestellt.
- *Fachbegriffe* und *Fremdwörter* werden durch *Schrägstellung* verdeutlicht. Begriffe, die für die Funktionalität wesentlich sind, werden bei der ersten Einführung durch zusätzlichen ***Fettdruck*** hervorgehoben.

Zur Darstellung von Klassen – und Sequenzdiagrammen wird der *UML* – Standard [14] angewandt.

Kapitel 2

Gesamtkonzept

Container

Ziel der Arbeit ist die Gestaltung einer Laufzeitumgebung für Komponenten (Komponenten – Verwaltung) zur Verarbeitung von nichtfunktionalen Anforderungen. Die Architektur des Containers, in den sich diese Laufzeitumgebung eingliedert, wird im ersten Teil dieses Kapitel vorgestellt. Daran anschließend werden die zur Spezifikation der *QoS* – Anforderungen einzusetzende deklarative Sprache ***Component Quality Modelling Language CQML*** sowie ihre Erweiterungen beschrieben.

EJB und AOP

Da sich die in diesem Container zu integrierenden Komponenten an der ***EJB*** – Spezifikation orientieren sollen, wird eine kurze Einführung in deren Aufbau erfolgen. Desweiteren wird in den anschließenden Kapiteln die Einsatzmöglichkeit ***Aspektorientierter Programmierung*** (***AOP***) zur Gestaltung der Laufzeitumgebung erörtert. Da in [10] bereits ein erster Ansatz zur Nachbildung der *EJB* – Funktionalität mit *AOP* ausführlich vorgestellt wurde, werden damit verbundene Besonderheiten am Ende dieses Kapitels kurz zusammengefasst.

2.1 Container – Architektur

Schichtenarchitektur

Der Container zur Verarbeitung der *QoS* – Anforderungen ist aus Schichten aufgebaut. Seine Architektur ist in der Abbildung 2.1 dargestellt. Die Entwicklung der darin integrierten Komponenten – Verwaltung ist Ziel der Arbeit. Zu den Aufgaben dieser Laufzeitumgebung gehören die Verwaltung der *QoS* – Eigenschaften, Aushandlung der Verträge zwischen kommunizierenden Komponenten sowie die Reservierung der Betriebsmittel. Die Erfüllung dieser Anforderungen soll für die Komponenten transparent erfolgen. Ein Entwurf der Komponenten – Verwaltung wird in den folgenden Kapiteln beschrieben.

Für die Reservierung und die Verwendung von Betriebsmitteln muss auf eine Ressourcen – Verwaltung zugegriffen werden. Dessen Funktionsweise wird im Folgenden erläutert.

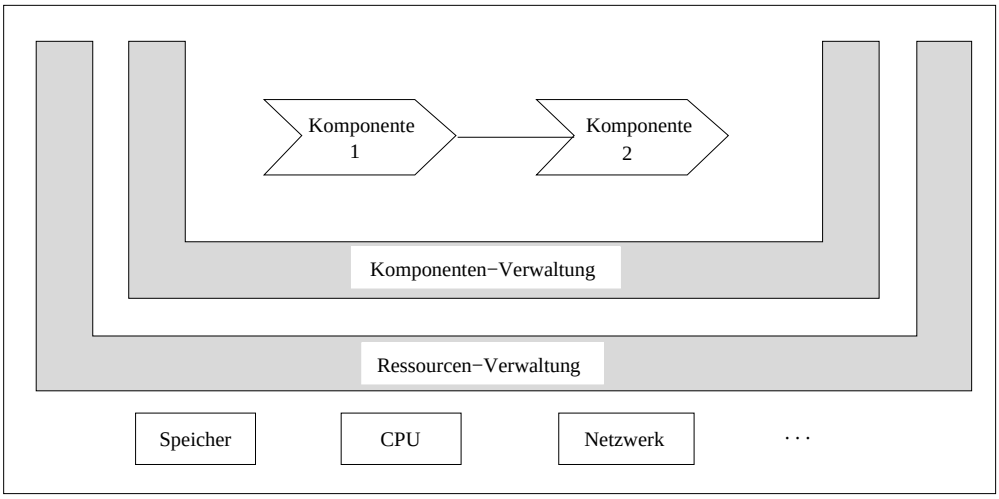


Abbildung 2.1: Container zur Verarbeitung der *QoS* – Anforderungen.

2.1.1 Ressourcen – Verwaltung

DROPS Ressourcen – Verwaltung

Die Organisation des Zugriffs auf die Betriebsmittel (z. B. CPU, Speicher, Netzwerk) erfolgt über eine Ressourcen – Verwaltung. In der am Institut für Betriebssysteme eingereichten Diplomarbeit von Nothnagel [22] wird der Prototyp einer solchen Ressourcen – Verwaltung vorgestellt. Auf deren Funktionalität greift die zu erstellende Komponenten – Verwaltung zu. Da die Implementierung der Ressourcen – Verwaltung auf dem am Lehrstuhl für Betriebssysteme in Entwicklung befindlichen ***D**resden **R**eal Time **Op**erating **S**ystem* (*DROPS*,[17]) erfolgte, für das keine lauffähige Version von Java existiert, kann die bereits umgesetzte Ressourcen – Verwaltung nicht verwendet werden. Es ist daher notwendig eine Implementierung, die der Schnittstelle zum Zugriff auf die Ressourcen – Verwaltung genügt, zu entwickeln. Diese simuliert die Reservierung der Ressourcen. Die Schnittstellen – Spezifikation kann größtenteils übernommen werden. Die notwendigen Modifikationen werden ausführlich im Implementierungs – Kapitel ab Seite 90 beschrieben.

In den folgenden Abschnitten werden die Aufgaben und Schnittstellen der Ressourcen – Verwaltung zusammenfassend vorgestellt.

Aufgaben der Ressourcen – Verwaltung

Hierarchische Verwaltung

Eine Aufgabe der Ressourcen – Verwaltung besteht in der Verwaltung der Betriebsmittel. Zur besseren Organisation ist das Managementsystem hierarchisch aufgebaut, wie in der Abbildung 2.2 an einem Beispiel zu sehen ist. Der *QoS* – Manager registriert alle Ressourcen – Manager und kommuniziert mit der Komponenten – Verwaltung.

Reservierung

Eine weitere Funktion der Ressourcen – Verwaltung ist die Verarbeitung der Ressourcenanfragen der Anwendungen. Anhand dieser ermittelt die Verwal-

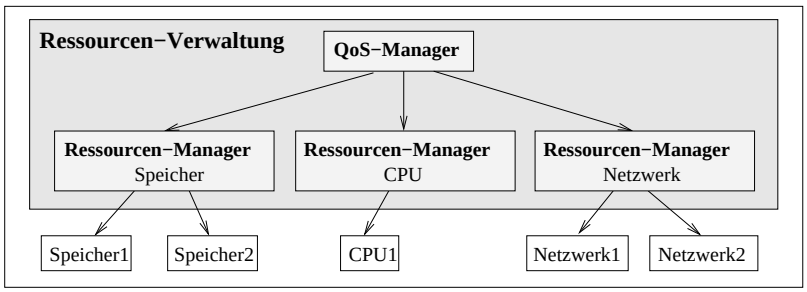


Abbildung 2.2: Beispiel einer Hierarchie von Ressourcen – Managern

tung alle Alternativen der zur Verfügung stehenden Ressourcen und reserviert schließlich die „günstigste“ Kombination.

Nachdem der Anwendung ein Vertrag über die reservierten Ressourcen übergeben wurde, muss die Ressourcen – Verwaltung sicherstellen, dass die einmal zugesicherten Ressourcen zur Verfügung stehen.

Durch Angabe einer Prioritätsstufe müssen in Situationen zu starker Auslastung einige Zusicherungen zurückgesetzt werden können, damit wichtigere Prozesse ausgeführt werden können. Diese Änderungen werden der Komponenten – Verwaltung signalisiert.

Um zu vermeiden, dass einzelne Anwendungen durch die Belegung zu vieler Ressourcen die Arbeit des Gesamtsystems behindern, wird die Anzahl der maximal zu belegenden Ressourcen beschränkt.

Freigabe

Die Komponenten – Verwaltung hat die Aufgabe zu gewährleisten, dass nicht mehr benötigte Ressourcen freigegeben werden. Sie meldet die Freigabe der Ressourcen – Verwaltung.

Schnittstelle der Ressourcen – Verwaltung

Reservierung

Für die Ressourcenanforderung steht der Komponenten – Verwaltung die Schnittstelle `reservation` zur Verfügung. Beim Zugriff auf den Ressourcen – Manager wird ein *Reservierungs – Handle* (`reservation_handle`) verarbeitet. Dieses enthält die in Tabelle 2.1 zusammengefassten Attribute.

Element	Beschreibung
`client`	Diese Zahl dient zur eindeutigen Identifizierung des Clients, der die Ressourcen reserviert hat.
`reservation_number`	Diese Zahl kennzeichnet eine einzelne Reservierung einer Anwendung.
`alternative_number`	Bei einer Reservierung werden von den Komponenten verschiedene Alternativen vorgegeben. Diese Zahl kennzeichnet die reservierte Alternative.
`_number`	Diese Zahl wird für die Verarbeitung einer Reservierung in mehreren Ressourcen – Managern benötigt.

Tabelle 2.1: Elemente der `reservation_handle` – Struktur

Zur Reservierung der Ressourcen wird folgende `negotiate` – Methode ausgeführt.

```
short negotiate(in client_id behalf,
                inout string reservation_request_and_reply,
                out reservation_handle handle);
```

Der Methode wird ein ***XML*** – `String`[16] mit der Beschreibung der Ressourcenanforderungen (entsprechend *request.dtd*) übergeben. Als Antwort erhält die Komponenten – Verwaltung die *XML* – Beschreibung der reservierten Ressourcen (entsprechend *contract.dtd*) als `String` sowie das `reservation_handle`. Die Beschreibung der *DTD*'s erfolgt im Anschluss. Sollte die Reservierung fehlschlagen, liefert die Methode den Wert `0`.

Freigabe

Durch Aufruf der `release` – Methode wird für eine Komponente die Reservierung aufgehoben. Konnten die reservierten Ressourcen nicht freigegeben werden, übermittelt die Methode den Wert `0` zurück.

```
short release(in reservation_handle handle);
```

Zur Freigabe aller Ressourcen einer Client – Anwendung wird die `release_all` – Methode vom Container aufgerufen. Diese Methode liefert `0` zurück, wenn die Freigabe fehl schlug.

```
short release_all(in client_id client);
```

Änderungen

Ändert sich die Verfügbarkeit einer Ressource, so wird die Komponenten – Verwaltung über die Methode `change` der Schnittstelle `notification` informiert.

```
short change(in reservation_handle handle
             in string changed_quality);
```

Wie bereits angesprochen, werden die zugehörigen Informationen beim Zugriff auf die Ressourcen – Verwaltung als `String` übergeben. Der Aufbau von Anfrage und Antwort werden nachfolgend beschrieben.

Ressourcenanfrage

request: Bei der Spezifikation der Anfrage wird im `resource_collection` Bereich eine Menge von alternativen Ressourcenanforderungen (`alternative`) angegeben. Dabei ist mindestens ein Eintrag erforderlich. Dem `alternative` – Element wird das Attribut `alternative_number` zugewiesen. Diese Zahl dient zur eindeutigen Kennzeichnung der Alternativen.

```
<resource_collection>
   <alternative alternative_number = ...>
      . . .
   </alternative>
<resource_collection>
```

Ressourcenbeschreibung

Innerhalb von `alternative` wird mindestens eine benötigte Ressource (`resource`) beschrieben.

```
<resource>
   <res_name> </res_name>
   <property>
      <prop_name> </prop_name>
      <type> </type>
      <unit> </unit>
      <value_range>
         . ..
      </value_range>
   </property>
</resource>
```

Eigenschaften

Der Eintrag `resource` besteht aus einem Ressourcennamen (`res_name`) und kann durch eine Menge weiterer Eigenschaften (`property`) ergänzt werden. Die Eigenschaften sind durch `prop_name`, `type`, `unit` und `value_range` gekennzeichnet.

`prop_name`	Name der Eigenschaft
`type`	Typ der Eigenschaft (z. B. integer, float, boolean, string)
`unit`	Einheit der Eigenschaft
`value_range`	Beschränkung des Wertebereichs

Wertebereiche

Bei der Beschränkung des Wertebereichs kann innerhalb von `value_range` entweder ein Minimum oder Maximum

```
<value_range>
   <minimum> </minimum>
   <maximum> </maximum>
</value_range>
```

oder ein fester Wert

```
<value_range>
   <value> </value>
</value_range>
```

angegeben werden.

Reservierungsbestätigung

contract: Innerhalb des `contract` – Bereiches wird die Zahl der ausgewählten Alternative sowie eine Auflistung der *Handles* (`handle`) angegeben. Es können beliebig viele *Handles* definiert werden. Diese Spezifikationen ermöglichen den Zugriff auf die reservierten Ressourcen.

```
<contract>
   <alternative_number> </alternative_number>
   <handle>
      . . .
   </handle>
</contract>
```

Innerhalb eines `handle` – Bereichs wird die reservierte Ressource (`resource`) beschrieben. Neben dem bereits vorgestellten `resource` – Eintrag werden die Ressourcen – Identifikationsnummer (`resource_id`) und die Reservierungsnummer (`reservation_number`) angegeben.

```
<handle>
   <resource>
      . . .
   </resource>
   <resource_id> </resource_id>
   <reservation_number> </reservation_number>
</handle>
```

Die Werte der in `contract` spezifizierten Einträge `reservation_number` und `alternative_number` werden in die `reservation_handle` – Struktur übernommen.

Mit der Beschreibung der Schnittstelle ist die Vorstellung der Ressourcen – Verwaltung nun abgeschlossen. Im folgenden Abschnitt wird auf einige Sprachen zur Spezifikation der in der Komponenten – Verwaltung zu verarbeitenden *QoS* – Eigenschaften eingegangen. Anschließend wird das in dieser Arbeit verwendete ***XML – Schema*** [15], welches in Anlehnung an die Sprache ***CQML***$^+$ entworfen wurde, kurz vorgestellt.

2.2 Spezifikation der QoS - Anforderungen

Aagedal diskutiert in [6] ausführlich verschiedene Ansätze zur Beschreibung von *QoS* – Eigenschaften. Als Ergebnis seiner Untersuchungen erfolgte die Vorstellung der Beschreibungssprache *CQML*. Nach einer kurzen Einführung in ihre Funktionsweise wird in diesem Abschnitt die Erweiterung *CQML*$^+$ sowie die in dieser Arbeit verwendete *XML* – Abbildung vorgestellt. Da *CQML* auf der Beschreibungssprache ***QML*** basiert, wird auf deren Eigenschaften ebenfalls kurz eingegangen. Eine umfassende Beschreibung und Diskussion der Vor – und Nachteile der Sprachen *QML* und *CQML* ist in [6] zu finden.

2.2.1 Quality of Service Modelling Language (QML)

QML Struktur

Die *Quality of Service Modelling Language* (*QML*) stellt eine allgemeine Sprache zur Beschreibung von *QoS* – Eigenschaften dar. Sie besitzt drei Grundmechanismen zur Spezifikation dieser Eigenschaften: *Contract Type* (Vertragstyp), *Contract* (Vertrag) und *Profile* (Profil). Während der Vertragstyp eine *QoS* – Kategorie (z. B. Performanz, Zuverlässigkeit), welche aus mehreren Dimensionen besteht, beschreibt, stellt ein Vertrag eine Ausprägung eines Vertragstyps dar. Der Vertrag gibt eine genaue *QoS* – Spezifikation für eine bestimmte Dimension wieder. Mit Hilfe eines Profils können die Verträge an Schnittstellen oder Objekten, die diesen Schnittstellen genügen, gebunden werden. Je Service – Anbieter kann genau ein Profil definiert werden.

Funktionsweise

Die Beschreibungssprache für *Quality of Service* – Belange bietet den Vorteil, dass sie die Spezifikation der *QoS* – Anforderungen von der eigentlichen Funktionalität der Anwendung trennt. Ein Nachteil besteht darin, dass zu einem Service – Anbieter nicht mehrere Profile angegeben werden können, aus denen in Abhängigkeit von Laufzeitbedingungen die beste Lösung ausgewählt wird. Nachteilig wirkt auch die unpräzise Spezifikation der *QoS* – Anforderungen. Aus diesem Grund kann der Zusammenhang mit dem Anwendungsmodell nicht definiert werden. Eine Erweiterung der Funktionalität zur Behebung dieser zwei Probleme stellt die Beschreibungssprache *CQML* dar.

Vor- und Nachteile

2.2.2 Component Quality Modelling Language (CQML)

CQML besteht aus vier Konstrukten zur Spezifikation der *QoS* – Eigenschaften:

CQML Struktur

- ***QoS Characteristic***
- ***QoS Statement***
- ***QoS Profile***
- ***QoS Category***

In Abbildung 2.3 ist der Zusammenhang zwischen den Bestandteilen dargestellt. Die *QoS* – Charakteristiken (*QoS Characteristic*) sind vom Nutzer definierte Typen, die zu *QoS* – Spezifikationen zusammengefasst werden. In einer *QoS* – Anweisung (*QoS Statement*) werden die Wertebereiche der Charakteristiken begrenzt.

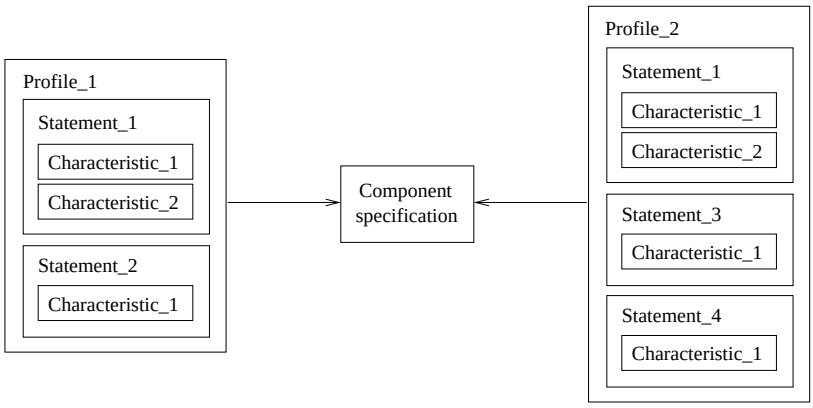


Abbildung 2.3: CQML Überblick [6]

Die Zuordnung der Spezifikationen zu konkreten Komponenten eines Systems erfolgt in *QoS* – Profilen (*QoS Profile*). Die auf diese Weise definierten *QoS* – Eigenschaften werden in *QoS* – Kategorien (*QoS Category*) gruppiert.

QoS–Charakteristik

QoS Characteristic (QoS – Charakteristik): Die Charakteristik stellt den Grundbaustein für eine *QoS*–Spezifikation dar und wird nach dem Schlüsselwort `quality_characteristic` deklariert. Sie ist durch einen eindeutigen Bezeichner gekennzeichnet und durch einen Wertebereich definiert. Zur Auswahl stehen die Typen `numeric`, `set` und `enum`. Im folgenden Beispiel ist eine Charakteristik zur Beschreibung der Antwortzeit einer Operation zu sehen.

```
quality_characteristic response_time (op: Operation) {
  domain: numeric real [0..) milliseconds;
  values: (op.SE->last().time() - op.SR->last().time()).abs();
}
```

Wertebereiche

Der Typ `numeric` repräsentiert eine Menge von reellen Werten. Der Wertebereich kann durch die Angabe von `real` (Real – Werte), `integer` (Integer – Werte) `natural` (natürliche Zahlen) konkret spezifiziert werden. In *CQML* sind per Grundeinstellung die natürlichen Zahlen inklusive null definiert.

Bei Verwendung von `numeric` kann der Wertebereich durch die Intervalnotationen (`(,)` oder `[, ]`) beschränkt werden. Desweiteren sind die Elemente im Wertebereich `numeric` linear geordnet. Dabei kann angegeben werden, inwieweit sich der semantische Wert (Qualität) der Einträge zu dem ursprünglichen Wert der Elemente verhält. Bei Angabe von

`increasing`:	steigt der semantische Wert eines Elementes, und mit
`decreasing`:	sinkt der semantische Wert eines Elementes

wenn ein höherer Betrag erreicht wird.

Zum Vergleich der Werte stehen die Vergleichsoperatoren ‘`<`’, ‘`>`’, ‘`=`’, ‘`<>`’, ‘`>=`’ und ‘`<=`’ zur Verfügung.

Die Typen `enum` (Enumeration) und `set` sind vom Nutzer frei definierte Mengen von Namen. Es besteht die Möglichkeit, die Ordnung der Einträge zu spezifizieren. In dem Fall muss die semantische Ordnungsrichtung angegeben werden.

Erweiterungen

Die Ableitung einer Charakteristik ermöglicht eine weitere Beschränkung der ursprünglich definierten Domäne. Desweiteren kann die funktionale Abhängigkeit einer *QoS*–Charakteristik von anderen Charakteristiken beschrieben werden.

Es besteht ebenfalls die Option, der Charakteristik Parameter zuzuweisen. Ein Parameter kann zum Beispiel ein Objekt, eine Methode, ein Fluss oder ein Signal beschreiben. Aus den Parametern wird der aktuelle Wert der Charakteristik errechnet (in *values*–Klausel).

Desweiteren können statistische Aspekte angegeben werden, wie z. B. `maximum`, `percentile` oder `distribution`. Diese werden aus einer Folge von Werten über einen bestimmten Zeitraum ermittelt.

Semantik

Die Semantik der Charakteristik wird in der `values` – Klausel definiert. Desweiteren können *Invarianten* spezifiziert werden. Die nach dem Schlüsselwort `invariant` beschriebenen Eigenschaften müssen sich immer zu einer wahren Aussage überführen lassen.

Komposition

In einem System treten Abhängigkeiten zwischen Komponenten auf. Zum Beispiel kann die Funktionalität einer Komponente durch mehrere andere Komponenten bestimmt werden. Die Auswirkungen auf die Charakteristik wird nach dem Schlüsselwort `composition` beschrieben. Es können sequentielle beziehungsweise parallele Kompositionen auftreten.

Im Fall einer Sequenz existieren Abhängigkeiten zwischen den Komponenten. Die Abarbeitung erfolgt in einer definierten Reihenfolge. Bei der Parallelität existieren keine Abhängigkeiten, d. h. Komponenten werden unabhängig voneinander ausgeführt. Dabei treten die in Tabelle 2.2 beschriebenen Spezialfälle auf.

Komposition	Beschreibung
`parallel-or`	Es stehen zum Beispiel mehrere gleichartige Komponenten zur Auswahl. Für die Fortführung kann die eine oder die andere Komponente ausgewählt werden.
`parallel-and`	Die Komponenten bieten zwar unterschiedliche, unabhängige Funktionalität an, beschreiben aber zusammen die Gesamtfunktionalität der Vorrichtung (z. B. verschiedene Ausgabegeräte).

Tabelle 2.2: Parallele Komposition von Komponenten

QoS – Anweisung

QoS Statement (QoS – Anweisung): Mit Hilfe der *QoS* – Anweisung wird der Wertebereich der Charakteristiken für die aktuelle Anwendung beschränkt. Dadurch können die Eigenschaften der Komponenten beschrieben werden. Im folgenden Beispiel wird eine Antwortzeit kleiner `200` gefordert (Wertebereich `numeric`).

```
quality fast_operation_response (op: Operation) {
  response_time (op) < 200;
}
```

Der übergebene Wert ist bei Verwendung der `set` – Domain eine Teilmenge der nutzerdefinierten Menge. Bei einem `enum` – Wertebereich wird ein einzelner Eintrag ausgewählt.

Wertzuweisung

Neben fest zugesicherten Werten (Schlüsselwort `guaranteed`), können innerhalb einer Anweisung auch best mögliche Werte (Schlüsselwort `best-effort`) definiert werden. Die Einhaltung dieser Zusagen wird nicht garantiert, da die Werte nur unter besten Bedingungen erreicht werden können. Auf diese Weise besteht die Möglichkeit, *QoS* – Eigenschaften zu definieren, zu deren tatsächlichen Werten noch keine Entscheidung getroffen werden konnte oder sollte. Für die *best – effort* – Anweisung stehen zwei Erweiterungen (*Threshold* und *Compulsory*) zur Verfügung, die jeweils mit einem Mindestwert (`limit`) arbeiten.

Bei einer einfachen Auflistung der Charakteristiken müssen alle Werte eingehalten werden. Optional ist es möglich, mehrere Charakteristiken durch logische Operatoren zu verbinden.

Anweisungen können spezialisiert werden. In dem Fall werden die Wertebereiche der Charakteristiken weiter beschränkt oder neue Charakteristiken aufgenommen.

QoS – Profil

QoS Profile (QoS – Profil): In dem Profil werden die *QoS* – Anweisungen einer Komponente zugewiesen. Dabei wird unterschieden, welche Anforderungen die Komponente an andere stellt und welche Eigenschaften sie anbietet. Diese zwei Aspekte werden durch die Bezeichner `uses` und `provides` gekennzeichnet.

`uses` - *QoS* – Eigenschaften, die eine Komponente von anderen Komponenten fordert
`provides` - *QoS* – Eigenschaften, die eine Komponente anbietet

Die den Schlüsselwörtern `uses` und `provides` zugewiesenen Parameter stellen Funktionseinheiten der Anwendung (z. B. Attribute der Komponenten) dar. Sie werden benötigt, um den Wert einer Charakteristik zu ermitteln. Die Parameter werden von den Anweisungen an die Charakteristiken weitergeleitet.

In dem folgenden Beispiel stellt die `SubscriptionManager` – Implementierung eine Forderung an die Methode `read` in `database`. Der Bezeichner `database` verweist auf eine Schnittstelle der Anwendung. Diese Beziehungen können in ***CORBA IDL*** [13] spezifiziert werden.

```
profile good_responsiveness for SubscriptionManager {
  uses fast_operation_response (database.read);
}
```

Die Profile können abgeleitet werden. In diesem Fall werden entweder die geforderten Eigenschaften herabgesetzt, die angebotenen Qualitätsmerkmale erhöht oder beides.

QoS – Verträge

CQML dient nur zur Beschreibung von Eigenschaften. Die Aushandlung von Verträgen und deren Überwachung muss durch die Laufzeitumgebung erfolgen. Eine Kommunikation zwischen den Komponenten läuft über die Schnittstellen. Diesen sind unterschiedliche Realisierungen hinterlegt, die zwar verschiedene Eigenschaften aufweisen aber die gleiche Semantik umsetzen. Dadurch besteht die Möglichkeit, aus verschiedenen Angeboten die passende Implementierung auszuwählen.

Die Auswahl der Partner kann aber auch eingeschränkt werden. Es können sich beispielsweise zwei Komponenten gegenseitig aufrufen (z. B. *callback* – Methoden). Diese bidirektionale Kommunikation läuft nur zwischen ihnen ab. Es darf keine andere Komponente angesprochen werden, die die Anforderungen besser erfüllt. Solche Beziehungen werden über die `invariant` – Klausel in den Profilen definiert.

Kann bei der Suche nach einer Komponente keine geeignete Implementierung ermittelt werden, müssen die Qualitätsanforderungen herabgesetzt werden. Zur Behebung des Problems können einer Komponente mehrere Profile zugewiesen werden. Erleichtert wird die Auswahl durch die Sortierung der unterschiedlichen Profile, somit immer auf die nächst bessere Lösung zugegriffen wird. Diese Ordnung spiegelt sich in der Auflistung nach dem Schlüsselwort `precedence` wider.

Ist ein Profil zur Laufzeit nicht mehr realisierbar, muss ein Wechsel erfolgen. Alle zulässigen Übergänge werden in Klauseln beschrieben, die durch das Schlüsselwort `transition` gekennzeichnet sind. Dabei besteht für die *Komponenten – Verwaltung* die Möglichkeit, die dort angegebenen Methoden auszuführen. Bei der Spezifikation der Übergänge werden die Namen der einfachen Profile oder das Schlüsselwort `any`, was für jedes beliebige Profil steht, verwendet.

QoS – Kategorie

QoS Catagory (QoS – Kategorie): Für verschiedene Anwendungensbereiche können unterschiedliche *QoS* – Terminologien verwendet werden. Um die zu einem Kontext gehörenden *QoS* – Charakteristiken, *QoS* – Anweisungen und *QoS* – Profile zusammenzufassen, werden *QoS* – Kategorien definiert.

CQML$^+$: Das ursprüngliche Modell von Aagedal, in dem das einfache Profil nur `uses` – und `provides` – Klauseln enthält, wurde um das Schlüsselwort `resources` erweitert. Es dient zur Beschreibung der von der Komponente benötigten Betriebsmittel. Verschiedene Alternativen von Ressourcenanforderungen können so definiert werden, von denen nur eine erfüllt werden muss. Die im `resources` – Bereich verwendeten Parameter verweisen auf Ressourcen.

In der Komponenten – Verwaltung wird zu Spezifikation der *QoS* – Eigenschaften ein *XML* – Schema verwendet, das in Anlehnung an *CQML*$^+$ entworfen wurde. Es wird im folgenden Abschnitt vorgestellt.

2.2.3 Definition der QoS – Spezifikation in XML

CQML im XML – Schema

Da die *CQML* – Grammatik Informationen enthält, die die Verarbeitung innerhalb der Laufzeitumgebung erschweren, wurden Umformungen vorgenommen. Die für Auswertung der *QoS* – Eigenschaften notwendigen Informationen wurden in ein *XML* – Schema übertragen.

Bei der Gestaltung der *XML* – Abbildung wurde sich an der *CQML* – Grammatik orientiert [30]. Die notwendigen Änderungen und Erweiterungen werden nun kurz vorgestellt. Auf der beiliegenden CD befindet sich das vollständige Schema.

Modifikationen

In *CQML* können die Anweisungen in einem Profil beliebig durch logische Operatoren verbunden werden. Bei der Umsetzung des *XML* – Schemas wurden die Kombinationsmöglichkeiten auf die *Konjunktive Normalform* (*KNF*) beschränkt. Diese wird durch die explizite Verwendung von einem `<and>` – und mehrerer `<or>` – *Tags* umgesetzt.

Die durch logische Operatoren verbundenen bewerteten Charakteristiken einer Anweisung sollen ebenfalls nur in der *KNF* auftreten. Dazu stehen die in einem `<simpleConstraint>` aufgelisteten `<singleConstraint>` in einer *OR* – Beziehung. Die *AND* – Beziehung wird durch Aufzählung mehrerer `<simpleConstraint>` oder `<qualifiedConstraint>` umgesetzt.

Einfache Profile (`<simpleProfile>`) können in der *XML* – Spezifikation nicht mehr direkt an Komponenten gebunden werden. Sie werden immer einem `<compoundProfile>` zugewiesen. Im einfachsten Fall enthält dieses nur einen Eintrag.

CQML$^+$ *im XML – Schema*

Die Erweiterungen aus *CQML*$^+$ werden in der *XML* – Spezifikationen bereits umgesetzt. Da nur eine Alternative erfüllt werden muss, werden die unabhängigen Ressourcenanforderungen in `<resources>` mit Hilfe eines `<or>` – und mehreren `<and>` – Tags beschrieben (*Disjunktive Normalform*).

Nachdem die Vorstellung der Sprachen zur Beschreibung der *QoS* – Eigenschaften abgeschlossen ist, folgt nun eine Einführung in das Komponentenmodel der in die Laufzeitumgebung zu integrierenden Anwendungen. Da sich an der *EJB* – Spezifikation orientiert werden soll, erfolgt eine kurze Vorstellung der zu berücksichtigenden Eigenschaften. Desweiteren werden die Erfahrungen zur Nachbildung der *EJB* – Funktionalität mit *AOP* zusammengefasst.

2.3 Verwendung des EJB – Komponentenmodels

Bei der Gestaltung der in die Komponenten – Verwaltung einzubindenen Anwendungen wird sich an der *EJB* – Spezifikation [2] orientiert. Die daraus resultierenden Anforderungen an Aufbau und Funktionsweise der Komponenten werden in diesem Abschnitt beschrieben.

Da in [10] bereits ein Teil der Funktionalität zur Verarbeitung von *Entity Beans* mit *Aspektorientierter Programmierung* [28][9] nachgebildet wurde, sollen die Erkenntnisse in diese Arbeit einfließen. Die mit ***AspectJ*** [3] entwickelte *AOP* – Umgebung wird am Ende dieses Abschnitts kurz vorgestellt.

2.3.1 Umzusetzende Konzepte des EJB – Containers

EJB – Konzepte

In der *Enterprise JavaBeans* – Spezifikation existieren folgende Komponententypen:

- *Entity Bean*
- *Stateful Session Bean*
- *Stateless Session Bean*

Jede Komponente besitzt eine *Home* – und eine *Remote* – Schnittstelle. Über die erstgenannte kann ein Client oder eine andere Komponente Instanzen anfordern. Die zweite Schnittstelle bietet Methoden zu Ausführung der Geschäftslogik an.

In dem verwendeten Komponentenmodell werden nur *Stateless* und *Stateful Session Beans* umgesetzt. Außerdem erfolgt keine Realisierung der durch den *EJB* – Container angebotenen Funktionseinheiten wie *Transaktionsverwaltung* oder *Zugangskontrolle*.

Zusätzlich wurde das Model um eine strombasierte Kommunikation zwischen den Komponenten erweitert. Dabei werden mit Hilfe der *CORBA IDL* [13] eingehende und ausgehende Ströme spezifiziert.

2.3.2 AOP – Umgebung für Entity Beans

AOP – Umgebung

In [10] wurden einige Funktionen des *EJB* – Containers zur Gestaltung von *Entity Beans* nachgebildet. Da die Funktionalität mit Hilfe von *AOP* integriert werden konnte, war es möglich, die Anwendung möglichst unabhängig von der Verarbeitung der zusätzlichen Funktionalität zu gestalten. Die für den Entwurf der Komponenten – Verwaltung wiederverwendbaren Konzepte der *AOP* – Umgebung werden in diesem Abschnitt zusammengefasst.

Namenskonventionen

Die *Entity Beans* in der *AOP* – Umgebung implementieren ihre eigenen Schnittstellen. Um diese Beziehung verarbeiten zu können, wurden Namenskonventionen eingeführt. Alle *Entity Beans* und ihre Schnittstellen werden durch den gleichen Namen zusammengefasst und müssen folgende Endungen annehmen:

Entity Bean – Klasse	- Endung: `Entity`
Home – Schnittstelle	- Endung: `RemoteHome`
Remote – Schnittstelle	- Endung: `Remote`

Für ein *Entity Bean* `PassengerEntity` müssen die Schnittstellen zum Beispiel `PassengerRemote` und `PassengerRemoteHome` lauten.

Integration

Mit Hilfe der Schnittstellen `EntityAOP`, `RemoteAOP`, `RemoteHomeAOP` und `ClientAOP` wird die Funktionalität der Laufzeitumgebung umgesetzt. Die Implementierung dieser Schnittstellen und damit die Integration in der *AOP* – Umgebung wird von Aspekten durch Auswertung der Namenskonventionen realisiert.

Kommunikation

Zur Realisierung des verteilten Zugriffs wird von der *AOP – Umgebung* die `getRemoteHome` – Methode zu Verfügung gestellt. Bei deren Ausführung wird der Name des geforderten *Entity Beans* übergeben.

Wiederverwendung von Konzepten zur Gestaltung der Komponenten – Verwaltung: Durch die in [10] erstellte *AOP* – Umgebung wurde gezeigt, dass mit Hilfe von *Aspektorientierter Programmierung* eine Laufzeitumgebung gestaltet werden kann. Bei der Umsetzung der Komponenten – Verwaltung werden grundlegende Konzepte, wie die Einführung von Namenskonventionen und der Methode zum Zugriff auf die Komponenten, leicht modifiziert übernommen.

Da sich die Eigenschaften von *Session Beans* aber sehr wesentlich von den *Entity Beans* unterscheiden, sind Erweiterungen vorzunehmen. Die mit Hilfe von *AOP*

umgesetzten Funktionen der Komponenten – Verwaltung werden im Kapitel 4 vorgestellt.

2.4 Zusammenfassung

In diesem Kapitel wurden die für die Gestaltung der Komponenten – Verwaltung zu berücksichtigenden Technologien zusammengefasst.

Wesentlich ist dabei die Architektur des Containers sowie der Aufbau und die Funktionsweise der Schnittstelle zur Ressourcen – Verwaltung.

Desweiteren wird eine Sprache zur Spezifikation der *QoS* – Eigenschaften benötigt. Aufgrund der Beschreibungsmöglichkeiten von $CQML^+$ bietet sich diese für die Lösung der Aufgabe an und wird in der weiteren Arbeit eingesetzt.

Für die Gestaltung und Verarbeitung der Anwendungen ist ein Komponentenmodel in Anlehnung an *EJB* zu verwenden. Um die Funktionalität der Komponenten – Verwaltung transparent zu integrieren, wird auf die Konzepte von *AOP* zurückgegriffen.

Auf Basis der beschriebenen Technologien kann der Entwurf einer Komponenten – Verwaltung erstellt werden. Darauf geht das nächste Kapitel ausführlich ein.

Kapitel 3

Entwurf einer Komponenten - Verwaltung

In diesem Kapitel erfolgt die Vorstellung des Entwurfs der Laufzeitumgebung zur Verarbeitung der *QoS* – Anforderungen. Diese Komponenten – Verwaltung ist in die im Kapitel 2 vorgestellte Infrastruktur zu integrieren. In den folgenden Betrachtungen werden Konzepte zur Umsetzung der Komponenten – Verwaltung erarbeitet und Lösungsalternativen bewertet.

Begriffsklärung

Zum besseren Verständnis der in dieser Arbeit verwendeten Begriffe werden die wesentlichen in der folgenden Tabelle 3.1 kurz definiert.

Begriff	**weitere Bezeichnung**	**Beschreibung**
Komponenten – Spezifikation	- Schnittstelle	Das ist die Schnittstellenbeschreibung einer Gruppe von Komponenten – Implementierungen, die die gleiche Semantik umsetzen. Die Implementierungen setzen die Funktionalität unterschiedlich um und bieten verschiedene *QoS* – Eigenschaften an.
Komponenten – Implementierung	- Implementierung - Komponenten – Realisierung - Realisierung	Es wird die konkrete Implementierung einer Komponente, die einer Schnittstelle (Komponenten – Spezifikation) genügt, beschrieben.
Instanz	- Objekt	Sie wird zur Laufzeit für eine konkrete Komponenten – Implementierung erstellt.
Komponenten – Netz	- Netze	Die zu einer Anfrage gehörenden kommunizierenden Objekte von Komponenten – Implementierungen werden in einen Komponenten – Netz zusammengefasst. Alle Objekte, die aufgrund der `uses` – Anforderungen an ein anderes Objekt gebunden sind, werden als 'abhängige Objekte' bezeichnet.

Fortsetzung auf der nächsten Seite

Fortsetzung von vorheriger Seite		
Vertrag		Bei der Erstellung der Komponenten – Netze werden zwischen kommunizierenden Komponenten – Implementierungen Verträge ausgehandelt. Es werden drei Typen von Verträgen vorgestellt (Abschnitt 3.5.1).
Komponente		'Abgeschlossener, binärer Software-Baustein, der eine anwendungsorientierte, semantisch zusammengehörende Funktionalität besitzt, die nach außen über Schnittstellen zur Verfügung gestellt wird.'[7] Im aktuellen Aufgabenkontext ist es die Zusammenfassung einer Komponenten – Implementierung und ihrer Spezifikation.

Tabelle 3.1: Begriffsklärung

Im nächsten Abschnitt erfolgt die Vorstellung einer kleiner Beispielanwendung. Daran anschließend wird die Architektur der Komponenten – Verwaltung vorgestellt und die Eigenschaften jeder Funktionseinheit detailliert beschrieben.

3.1 Die Beispielanwendung: Börsenticker

Um die nachfolgend beschriebenen Entwurfsentscheidungen und Problemsituationen besser verstehen zu können, soll eine Anwendung zur Veranschaulichung dienen. Als als ein mögliches Umsetzungsbeispiel wird der Börsenticker vorgestellt.

Aufgaben der Anwendung

Der Börsenticker hat die Aufgabe, die Aktienkurse auszuliefern. Der Nutzer kann sich bei ihm über einen `SubscriptionManager` anmelden. Dazu können die nichtfunktionalen Anforderungen wie zum Beispiel Antwortzeit, Prioritäts- und Sicherheitsstufe spezifiziert werden. Ein `StockQuotationDistributor` empfängt die aktuellen Aktienkurse und muss sie an alle registrierten Nutzer weiterleiten.

Aufgaben der Laufzeitumgebung

Die Aufgabe der Komponenten – Verwaltung ist die Auswahl eines geeigneten `StockQuotationDistributor`. Dazu werden die nichtfunktionalen Anforderungen ausgewertet. Nach Abschluss der Registrierung muss die Einhaltung der Verträge überwacht beziehungsweise durchgesetzt werden.

Auf den folgenden Seiten werden zur Veranschaulichung verschiedener Situationen unterschiedliche Umsetzungen dieser Beispielanwendung beschrieben.

3.2 Architektur der Komponenten – Verwaltung

Die zu entwickelnde Laufzeitumgebung ist Bestandteil des im Kapitel 2 vorgestellten Containers (Abb. 2.1). Der Aufbau dieser Komponenten – Verwaltung ist in Abbildung 3.1 veranschaulicht.

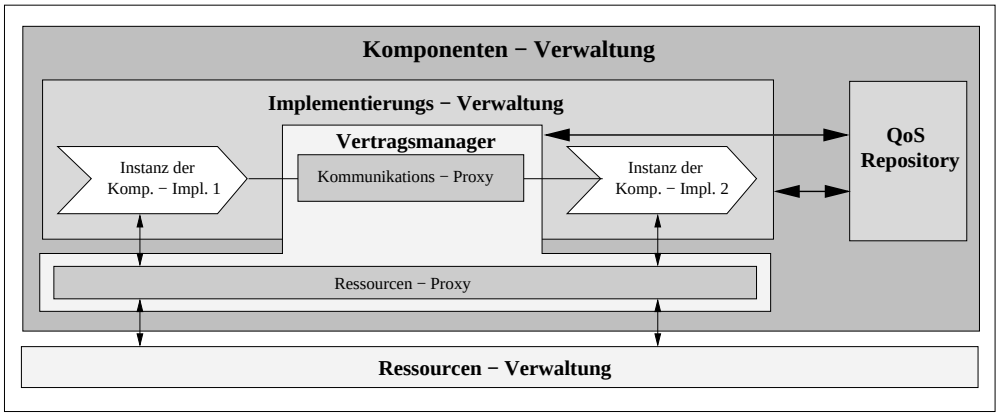


Abbildung 3.1: Komponenten – Verwaltung

Funktionseinheiten

Eine **Implementierungs – Verwaltung** hat die Aufgabe, die Komponenten – Implementierungen und Instanzen zu verwalten.

Die aufgrund der Verarbeitung der *QoS* – Anforderungen erfolgende Aushandlung und Durchsetzung der Verträge übernimmt der **Vertragsmanager**. Für die Durchsetzung der Verträge sind seine Funktionsbereiche **Kommunikations – Proxy** und **Ressourcen – Proxy** verantwortlich.

In jeder der zuvor genannten Funktionseinheiten muss eine Verarbeitung von *QoS* – Eigenschaften erfolgen. Deren Verwaltung erfolgt im **QoS – Repository**.

In den folgenden Abschnitten werden alle Funktionseinheiten ausführlich beschrieben.

3.3 QoS – Repository

QoS – Repository

Zu den Aufgaben des *QoS – Repository* gehören das Einlesen und Verwalten der *QoS* – Eigenschaften. Der Aufbau der einzulesenden Datei orientiert sich an der $CQML^+$ – Grammatik. Um die Eigenschaften zur Laufzeit leicht verarbeiten zu können, werden sie eingelesen, ausgewertet und auf objektorientierte Strukturen übertragen. Die dazu entwickelten Abbildungen werden in den folgenden Abschnitten vorgestellt.

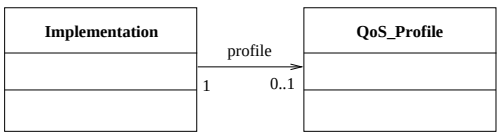


Abbildung 3.2: Bindung der *QoS* – Eigenschaften an die Implementierungen

3.3.1 Zugriff auf die QoS – Eigenschaften

Profilzuweisung Damit auf die im *QoS* – Repository verwalteten Eigenschaften zur Laufzeit zugegriffen werden kann, ist eine Zuweisung zu den Komponenten – Implementierungen notwendig. Dazu wird an jede Komponente (`Implementation`) ein Profil (`QoS_Profile`), das die *QoS* Eigenschaften beschreibt, gebunden (Abb. 3.2).

Auswertung Während der Aushandlung der Verträge müssen die *QoS* – Eigenschaften ausgewertet und miteinander verglichen werden. Die dabei zu verarbeitetenden Funktionen werden bei der Vorstellung des Vertragsmanagers näher erläutert (S. 39 ff.).

3.3.2 Abbildung der QoS – Eigenschaften

Profil *Profil:* Ein Profil (`QoS_Profile`) kann aus mehreren einfachen Profilen (`QoS_SimpleProfile`) zusammengesetzt sein (Abb. 3.3), die nach ihrer Wertigkeit sortiert sind. Diese Ordnung wird der *CQML*$^+$ – Spezifikation entnommen.

Einfaches Profil *Einfaches Profil:* Das einfache Profil ist durch einen Namen gekennzeichnet und enthält die Beschreibung der geforderten (`uses`) und angebotenen (`provides`) Eigenschaften.

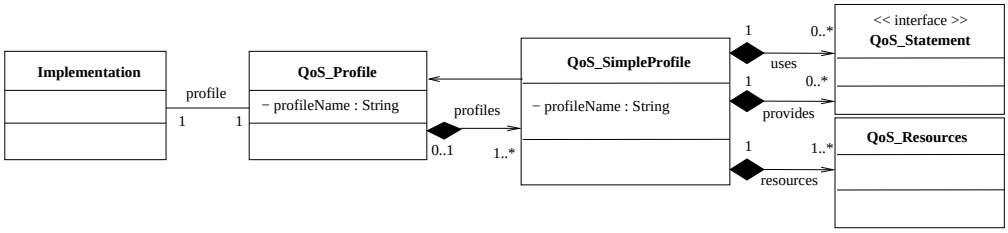


Abbildung 3.3: Verwaltung von Profilen

Ressourcen *Ressourcenanforderung:* Desweiteren werden die Ressourcenanforderungen spezifiziert (`resources`). Da sich die Objektabbildung an der Anfrage, die an die Ressourcen – Verwaltung zu stellen ist, orientiert, weicht der Aufbau stark von der *CQML*$^+$ – Struktur ab (Abb. 3.4).

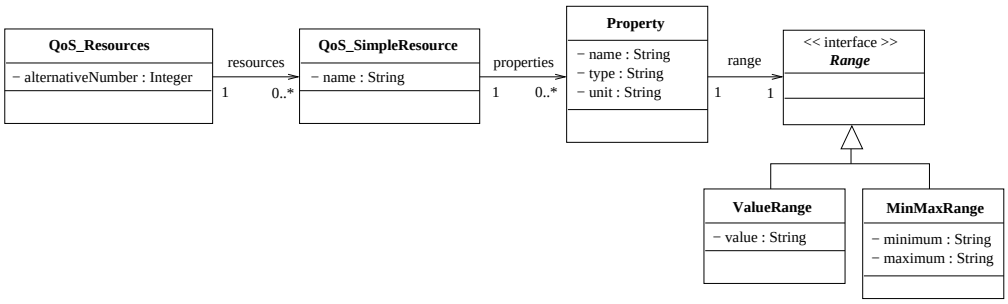


Abbildung 3.4: Ressourcenanforderungen

Beschränkungen in `resources`

Da die Beschreibungsmöglichkeiten der Anfragen an die Ressourcen – Verwaltung eingeschränkt sind, müssen bei der Spezifikation der Ressourcenanforderungen mit $CQML^+$ folgende Restriktionen beachtet werden:

- Der Qualifier ist immer `guaranteed`.
- Für die Wertzuweisung stehen nur folgende Varianten zur Auswahl:
  - Es wird der statistische Wert `Range` spezifiziert (Zuweisung nur mit `=`).
  - Es wird ein Wert festgelegt (Zuweisung nur mit `=`).
- Die Ressourcenanforderungen in einer Alternative müssen alle erfüllt werden. Es darf keine *OR* – Beziehung auftreten. Demzufolge wird in `<simpleContraint>` immer nur ein `<singleContraint>` definiert.[30]
- Spezialisierung und funktionale Abhängigkeiten werden nicht verarbeitet.
- Da nur einzelne Werte an die Ressourcen – Verwaltung übergeben werden, können Mengen nicht verarbeitet werden. Mögliche Datentypen sind `integer`, `float`, `boolean` oder `string`. `Boolean` – und `String` – Werte können durch Instanzen der `EnumerationDomain` beschrieben werden.

Alternativen

Zu einem Profil einer Implementierung können mehrere Alternativen für Ressourcenanforderungen spezifiziert werden. Es ist jedoch mindestens eine erforderlich. Jede Alternative wird in einem `QoS_Resources` – Objekt beschrieben. Zur Identifizierung der Alternative wird ein `Integer` – Wert generiert. Die Reihenfolge der Alternativen spiegelt ihre Wertigkeit wieder. Zuerst aufgezählte stellen die besseren Ressourcenkombinationen dar.

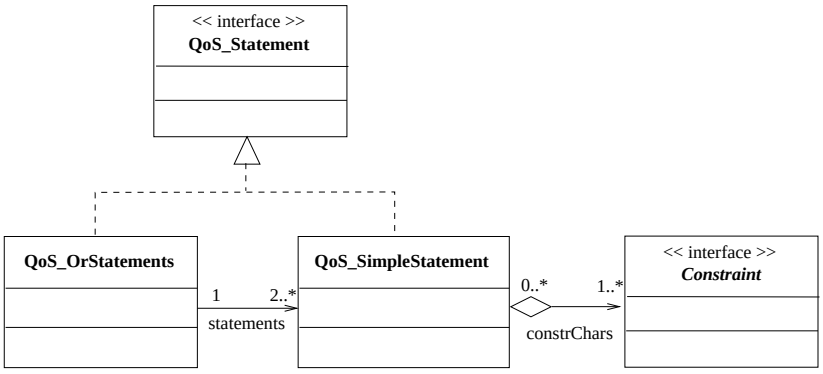


Abbildung 3.5: Einfache und durch *OR* – Beziehung zusammengefasste Anweisungen

Anweisungen

Anweisung: Die Anweisungen in den `uses` – oder `provides` – Klauseln werden als eine *KNF* zusammengefasst. Die dabei auftretenden *OR* – Beziehungen werden durch Objekte der `QoS_OrStatements` – Klasse (Abb. 3.5) und die *AND* – Beziehung durch Auflistung der Anweisungen in `QoS_SimpleProfile` umgesetzt.

Bewertete Charakteristik

Einfache Anweisung: Eine einzelne Anweisung (`QoS_SimpleStatement`) enthält eine Auflistung bewerteter Charakteristiken (`Constraint`), die ebenfalls in der *KNF* auftreten. In dem Fall wird die *OR* – Beziehung durch die `OrConstraints` – Klasse realisiert. Der Aufbau bewerteter Charakteristiken, die in einer Anweisung verwaltetet werden, ist in Abbildung 3.6 zu sehen.

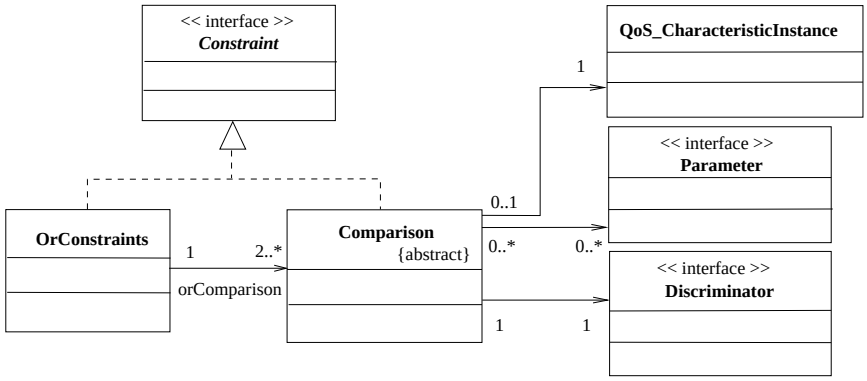


Abbildung 3.6: Zusammensetzung der bewerteten Charakteristiken

Die doppelte Verwaltung der *Konjunktiven Normalformen* für Anweisungen und bewertete Charakteristiken wurde aus der *CQML*$^+$ – Spezifikation übernommen. In zukünftigen Versionen sollte das Einlesen so optimiert werden, dass die Einträge zu einer einzigen *KNF* umgewandelt werden. Das hat aber zur Folge, dass die Gruppierung in den Anweisungen auseinander bricht. In diesem Fall kann auf die Verwaltung der Anweisungen vollständig verzichtet werden.

Comparison

Bewerte Charakteristik: Den Charakteristiken werden Werte und Parameter zugewiesen. Die Zusammenfassung der Eigenschaften einer einzelnen bewerteten Charakteristik erfolgt in einem `Comparison` – Objekt. Da die Untersuchung der Eignung eines Profils von den verwendeten Vergleichsoperatoren (`=`, `<`, `>`, `<=`, `>=` oder `<>`) beeinflusst wird, werden mehrere Ableitungen der `Comparison` – Klasse umgesetzt (Abb. 3.7). Auf die beim Vergleich zu beachtenden Besonderheiten wird bei der Vertragsaushandlung eingegangen (S. 44 ff.).

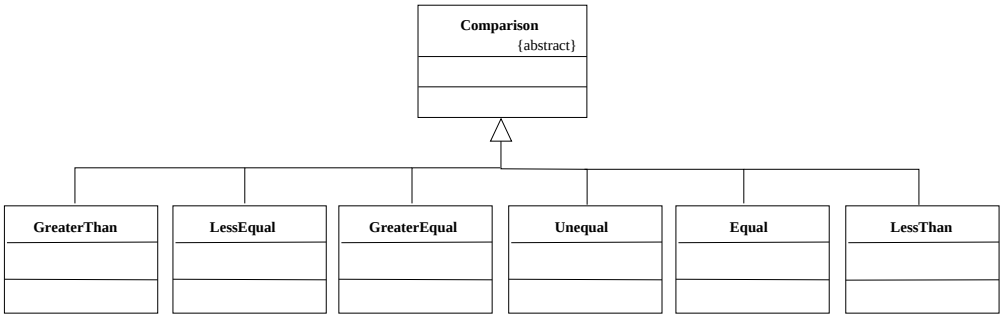


Abbildung 3.7: Verteilung der Funktionalität auf verschiedene Vergleichsoperatoren

Diskriminator: Die einer Charakteristik zugewiesenen Werte können verschiedene *Diskriminatoren* aufweisen. Sie werden durch die Klassen umgesetzt, die die Schnittstelle `Discriminator` implementieren (Abb. 3.8). Die Bedeutung der verschiedenen Alternativen – Wertzuweisung sowie ihr Einfluss auf den Vergleich von zwei bewerteten Charakteristiken wird ab Seite 50 vorgestellt. *Diskriminator*

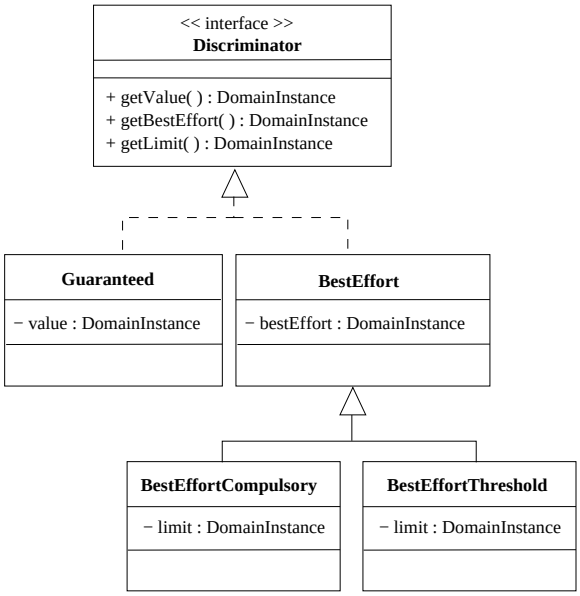


Abbildung 3.8: Klassifizierung der Wertzuweisung (`DomainInstance`, Abb. 3.11)

In der *CQML*$^+$–Spezifikation werden die *Qualifier* (`guaranteed` und `best-effort`) mehreren bewerteten Charakteristiken gleichzeitig zugewiesen. Desweiteren können die '`best-effort with limit`' – Einträge nie in *OR* – Beziehungen auftreten. Dies ist in der Objektabbildung möglich. Außerdem wird der *Diskriminator* jeder Charakteristik einzeln zugewiesen, um den Vergleich für alle einheitlich gestalten zu können. Da auf diese Weise die Darstellungsmöglichkeiten nicht beschränkt werden, stellen die Abwandlungen kein Problem dar. *Abweichung von CQML*$^+$

Parameter: Zu den bewerteten Charakteristiken müssen desweiteren die Parameter eingelesen werden. Es wird der in der Charakteristik verwendete Parametertyp gespeichert. Dazu müssen alle auf die in der *CORBA IDL* spezifizierten Schnittstellen oder Ereignistypen ausgeführten Ableitungen ausgewertet werden. Die Informationen zu den in den Charakteristiken verwendeten Parametern werden auf die Klassenstruktur im Diagramm 3.9 abgebildet. *Parameter*

Bei der einfachsten Ableitung wird durch einen Parameter eine Operation einer Schnittstelle oder ein Ereignistyp beschrieben. Der in dem Fall abgespeicherte Parameter ist eine `Operation` oder ein `Flow`. Zu der `Operation` wird der Schnittstellenname (`specName`) und der Name der Operation (`opName`) gespeichert. Bei Verwendung von `Flow` wird der Name des in der *IDL* definierten Ereignistyps (`flowName`) verwaltet.

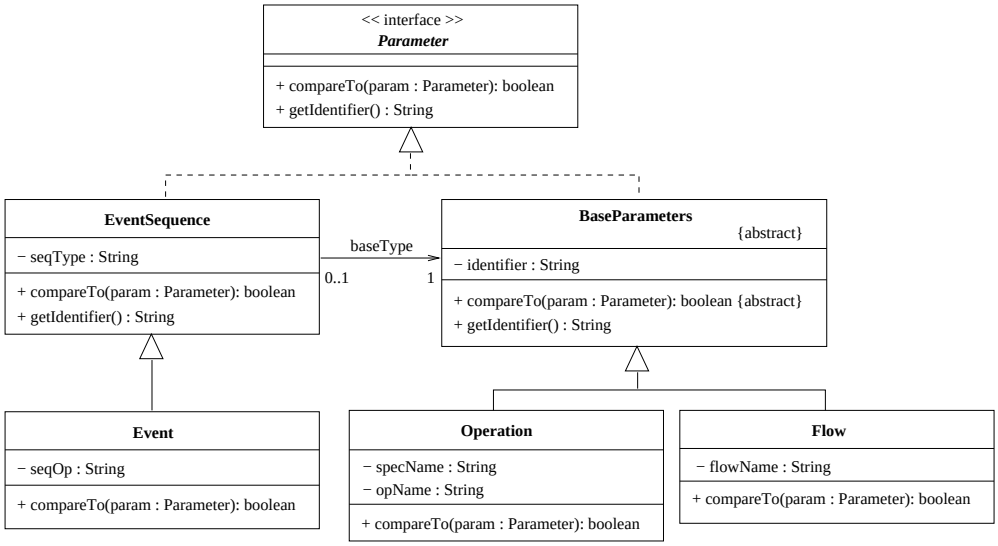


Abbildung 3.9: Parameter

Parameter können aber noch weiter ausgewertet werden. Durch Angabe von `SE` oder `SR` wird die `EventSequence` abgeleitet. In dem Fall wird der Typ der `EventSequence` (`seqType`) angegeben – `SE` oder `SR`. Werden auf die `EventSequence` weitere Operationen ausgeführt, wird ein `Event` beschrieben. Die Operation (`seqOp`) wird ebenfalls als Zeichenkette gespeichert.Für eine vollständige Analyse der Parameter müsste eine Auswertung dieser Operation erfolgen. Da die Auswertung der Semantik sehr aufwendig ist, wird sie ihm Rahmen dieser Arbeit nicht durchgeführt.

Zum Verweis auf die Schnittstellen und Ereignistypen werden in der *CORBA IDL* Bezeichner spezifiziert, die in der *CQML*$^+$ – Beschreibung verwendet werden. Diese werden zusätzlich in den Parametern verwaltet (`identifier`), da sie bei der Vertragsaushandlung (siehe Seite 39) ausgewertet werden müssen.

Soll eine Auswertung der `values` – Klausel erfolgen, müssen zusätzlich die in der Charakteristik verwendeten Parameternamen gespeichert werden, denn ansonsten ist eine Zuordnung der Parameter zu den in der Klausel verwendeten Namen nicht möglich. In der aktuellen Version der Komponenten – Verwaltung werden diese Parameternamen nicht verwaltet.

Beispiel Am folgenden Beispiel soll die Auswertung der Parameter dokumentiert werden.

```
component StockQuotationManager {
  provides StockQuotationDistributor outgoing;
}
```

Der `StockQuotationManager` bietet einen Service über die Schnittstelle `StockQuotationDistributor` an. Der dafür spezifizierte Bezeichner `outgoing` wird im Profil `good_processing` verwendet.

```
profile good_processing for StockQuotationManager {
  provides high_event_rate(outgoing.update.SE->last());
}
```

Aufgrund der Verwendung des Bezeichners `outgoing` wird erkannt, dass in der `high_event_rate` – Anweisung die `update` – Methode der Schnittstelle `StockQuotationDistributor` zu überwachen ist. Es wird durch Ausführung der `last()` – Operation die `SE` – `EventSequence` dieser Methode ausgewertet.

```
quality high_event_rate(event : Event) {
  event_rate(event) > 10000;
}
```

Bei Auswertung der Anweisung kann schließlich abgeleitet werden, dass dieser Parameter in der Charakteristik `event_rate` verwendet wird. In dem Beispiel wird der Charakteristik `event_rate` demzufolge ein `Event` – Objekt mit den oben abgeleiteten Eigenschaften als Parameter zugewiesen.

Es können in den Anweisungen auch mehrere Charakteristiken, denen die Parameter zuzuweisen sind, definiert werden. Die Zuordnung innerhalb der Anweisung erfolgt auf Grundlage der Namensgleichheit. Bei der Übertragung vom Profil zur Anweisung wird die Position in der Parameterliste ausgewertet.

Instanz der Charakteristik: Die einer Komponente zugewiesene Charakteristik wird durch die `QoS_CharacteristicInstance` – Klasse beschrieben (Abb. 3.10). Sie beinhaltet einen Verweis auf die Definition der Charakteristik und spezifiziert den verwendeten statistischen Wert. In der zu entwerfenden Komponenten – Verwaltung werden nur die statistischen Werte *Minimum*, *Maximum* und *Range* betrachtet. Enthält `statistic` keinen Eintrag wird der Wert der Charakteristik direkt zugewiesen. *Instanz einer Charakteristik*

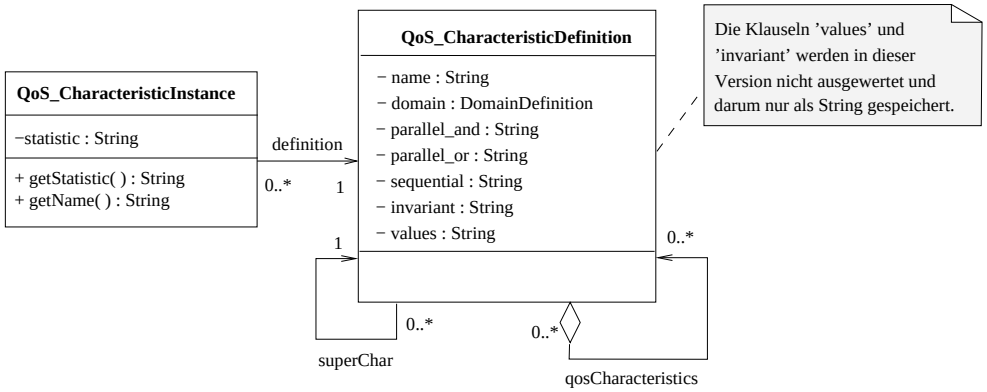


Abbildung 3.10: Verwaltung der Informationen zu einer Charakteristik

Definition der Charakteristik: Im Charakteristik – Definitionsteil werden Name, Wertebereich, *Invariante*, *values* – Klausel sowie die Beschreibungen der Kompositionen verwaltet (`QoS_CharacteristicDefinition`). Eine Charakteristik kann von anderen Charakteristiken funktional abgeleitet werden. Diese Abhängigkeit wird in der Assoziation `qosCharacteristics` hinterlegt. Desweiteren ist *Definition einer Charakteristik*

eine Spezialisierung der Charakteristiken möglich. Die übergeordnete Charakteristik wird in `superChar` gespeichert.

Da davon ausgegangen werden kann, dass eine Überprüfung der korrekten Verwendung der Charakteristik erfolgt ist, kann in dieser Arbeit auf die Klasse `QoS_CharacteristicDefinition` verzichtet werden. In dem Fall hat die `QoS_CharacteristicInstance` – Klasse den Namen der Charakteristik aufzunehmen. Werden aber in einer Erweiterung der in dieser Arbeit umgesetzten Funktionalität die `values` – und `invariant` – Klauseln ausgewertet, muss auf den Definitionsteil zugegriffen werden. Daher wird in der aktuellen Version der Komponenten – Verwaltung diese zuvor beschriebene Beziehung zwischen der an einer Komponente gebundenen Charakteristik und ihrem Definitionsteil umgesetzt.

Die einer Charakteristik zugewiesenen Werte müssen einen definierten Wertebereich (`DomainDefinition`) einhalten (Abb. 3.11).

Wertebereich

Wertebereich / Datentyp: Das Diagramm 3.11 stellt alle in *CQML*$^+$ verwendeten Datentypen dar. Es wird davon ausgegangen, dass die Typ – Kontrolle und die Überprüfung der Einhaltung des Wertebereichs bereits vor dem Einlesen der Werte erfolgt ist. Daher können die *Numeric* – Datentypen (*Natural*, *Integer* und *Real*) zusammengefasst werden. Desweiteren finden die Datentypen *Enum* (*Enumeration*) und *Set* Anwendung.

Die Beschreibung der Datentypen wird in einen Definitions – und einen Instanzenteil unterschieden. Im Definitionsteil werden die allgemeinen Informationen zu den in den Charakteristiken spezifizierten Wertebereichen verwaltet. Bei der Zuweisung einer Charakteristik zu einer Komponente, werden Instanzen dieser Wertebereiche gebildet. Während es im Fall von *Numeric* und *Enum* nur einzelne Einträge sind, bilden Instanzen von Mengen ebenfalls Mengen[1].

Die Gliederung in den Wertebereichen *Enum* und *Set* wird durch geordnete Paare (`valueOrder`) beschrieben. In der *CQML*$^+$ – Spezifikation kann die Ordnungsrichtung festgelegt werden, welche im Definitionsteil durch das Attribut `direction` wiedergegeben wird.

Eine wichtige Funktion stellt der Vergleich von Objekten eines Datentyps dar. Auf die Umsetzung dieser Funktionalität wird im Kapitel 4 detailliert eingegangen (S. 86 ff.).

In diesem Abschnitt wurden die Aufgaben des *QoS* – Repository vorgestellt. Neben einer Beschreibung der Abbildung der *CQML*$^+$ – Spezifikation auf objektorientierte Klassenstrukturen wurde eine Möglichkeit zur Bindung der Informationen an die Komponenten – Implementierungen veranschaulicht. Im folgenden Abschnitt wird die Funktionsweise der Implementierungs – Verwaltung erläutert.

[1] Diese Eigenschaft ist in der *CQML*$^+$ – Spezifikation nicht umgesetzt, wurde aber in der *XML* – Abbildung eingeführt.

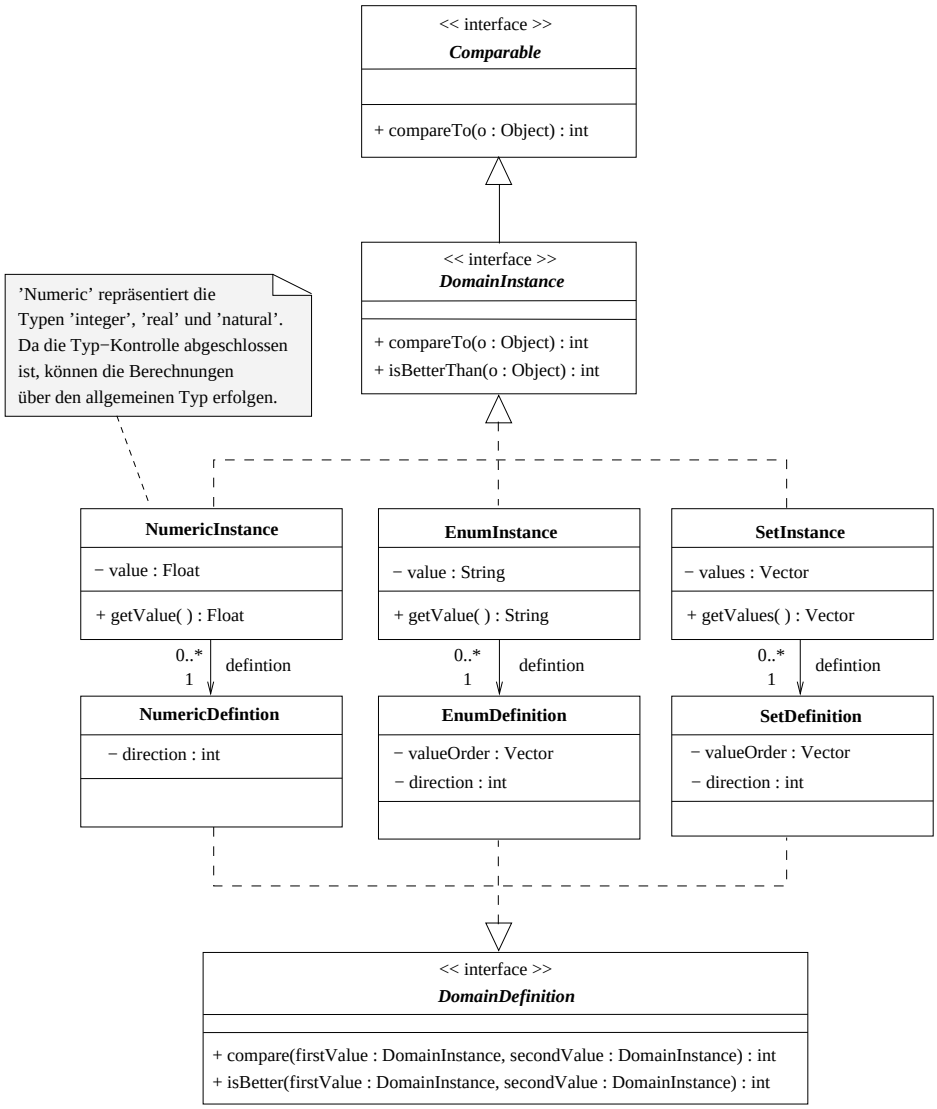


Abbildung 3.11: Definition der Datentypen und Instanzen dieser Datentypen

3.4 Implementierungs – Verwaltung

Die Aufgaben der Implementierungs – Verwaltung bestehen in der:

- Verwaltung der Komponenten – Implementierungen und
  - Einlesen und Entfernen von Komponenten – Implementierungen
  - Verwaltung der von einer Komponenten – Implementierung angebotenen Schnittstelle
  - Verwaltung der Abhängigkeiten der Komponenten – Implementierungen von anderen Schnittstellen (Beziehungen)
- Verwaltung der zur Laufzeit erstellten Instanzen.

Die aufgelisteten Funktionsbereiche werden in den folgenden Abschnitten ausführlich beschrieben.

3.4.1 Verwaltung der Komponenten – Implementierungen

Bevor auf die Funktionsweise der Verwaltung eingegangen wird, werden die zur Verarbeitung erforderlichen Informationen vorgestellt.

Komponenten – Spezifikation

Komponenten – Spezifikation: Die Kommunikation der Komponenten erfolgt über ihre Schnittstellen (Komponenten – Spezifikationen). Da diese Beziehungen von der Komponenten – Verwaltung zu überwachen sind, ist es erforderlich, diese Spezifikation zu verwalten (Abb. 3.12). Jede Schnittstelle kann von mehreren Klassen implementiert werden.

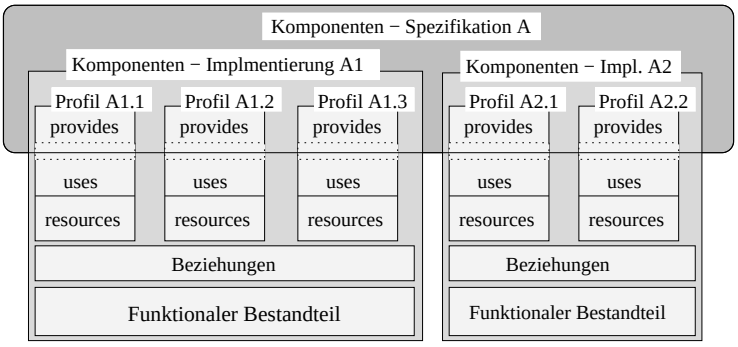


Abbildung 3.12: Zusammenfassung von Komponenten – Implementierungen zu einer Komponenten – Spezifikation

Komponenten – Implementierung: Die Implementierungen einer Spezifikation setzen die gleiche Semantik auf unterschiedliche Weise um. Die Besonderheit bei der Arbeit mit *QoS* – Anforderungen besteht darin, dass sich die Implementierungen nicht nur in der Umsetzung des funktionalen Bereichs unterscheiden, sondern auch verschiedene *QoS* – Eigenschaften anbieten. Jede Implementierung kann über mehrere alternative Profile verfügen, die unterschiedliche *QoS* – Anforderungen beschreiben. *Komponenten – Implementierung*

Profil: Ein einzelnes Profil besteht aus den bereits auf den Seiten 12 und 13 beschriebenen Bereichen *uses*, *provides* und *resources*.

Beziehungen zwischen Implementierungen: Eine Implementierung kann über die Spezifikationen auf andere Komponenten – Implementierungen zugreifen. Diese Abhängigkeiten werden in dem Bereich *Beziehungen* zusammengefasst. Durch eine separate Verwaltung der Beziehungen wird deren Auswertung erleichtert. Es müssen nicht bei jeder Anfrage die *QoS* – Eigenschaften ausgewertet werden, um die abhängigen Spezifikationen abzuleiten. Außerdem werden auf diese Weise alle abhängigen Schnittstellen erfasst, auch jene, an die keine *QoS* – Anforderungen gestellt werden. *Beziehungen*

Verwaltung der Informationen: Die zuvor beschriebenen Informationen zu den Implementierungen werden auf die im Diagramm 3.13 dargestellte Klassenstruktur abgebildet. *Verwaltung*

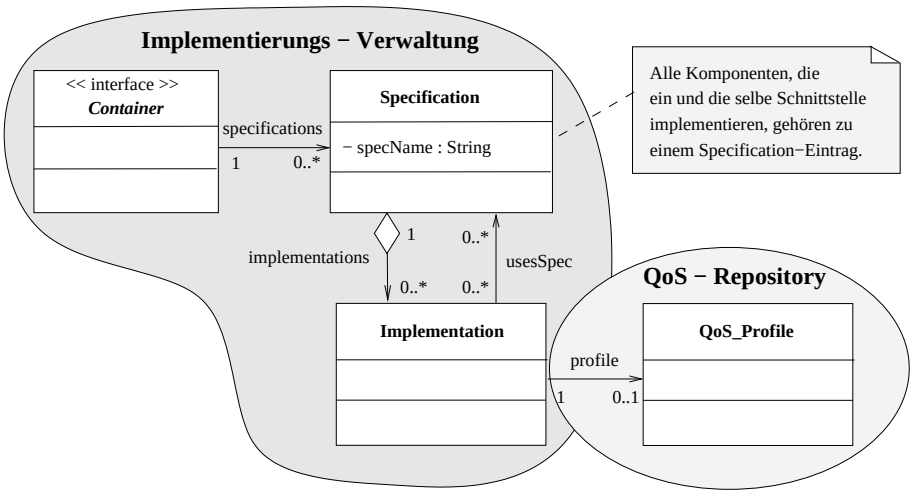


Abbildung 3.13: Verwaltung der Komponenten – Implementierungen

Im Container (`Container`) werden alle Spezifikationen (`Specification`) verwaltet. Diese enthalten wiederum die Implementierungen (`Implementation`), die der beschriebenen Schnittstelle genügen. In der `Implementation` – Klasse werden außerdem die von einer Komponenten – Implementierung verwendeten (abhängigen) Schnittstellen verwaltet.

Um die zuvor beschriebenen Informationen ableiten zu können, müssen durch den Anwendungsentwickler die Beziehungen spezifiziert werden. Der erstellte *Beziehungs – Deskriptor* wird beim Einlesen der Server – Anwendung ausgewertet. Auf die Details der zum Zeitpunkt des Einlesens benötigten Informationen wird nachfolgend eingegangen.

Beziehungsspezifikation

Spezifikation von Beziehungen: Als Voraussetzung für die Verarbeitung der Beziehungen zwischen Komponenten müssen für alle Implementierungen folgende Informationen zur Verfügung stehen:

- angebotene Schnittstelle
- geforderte (abhängige) Schnittstellen

Beziehungen zwischen Instanzen

In manchen Situationen muss die Spezifikation der Beziehungen auf Instanzenebene ausgeweitet werden können. Es sollte zum Beispiel erkennbar sein, ob zwei Implementierungen, die auf die gleiche Spezifikation verweisen, die selbe Instanz benötigen. Eine ausführliche Diskussion dieses Problems erfolgt ab Seite 58.

In dieser Arbeit wird von der Existenz einer *XML* – Datei ausgegangen, die in Anlehnung an *CORBA IDL* [13] gestaltet wurde (Beziehungs – Deskriptor). Mit Hilfe dieser kann aber keine Beschreibung der Beziehungen auf Instanzenebene erfolgen. Sie besitzt `provides` – und `uses` – Klauseln, über die die von einer Implementierung angebotene (eine) und verwendeten (mehrere) Schnittstellen spezifiziert werden. Während der Vertragsaushandlung müssen für die `uses` – Einträge Instanzen von geeigneten Komponenten – Implementierungen reserviert werden. Mit Hilfe der `consumes` – und `publishes` – Klauseln wird der Empfang und das Aussenden von Informationsflüssen dokumentiert. In der aktuellen Version der Komponenten – Verwaltung erfolgt bei der Verarbeitung dieser strombasierten Kommunikation keine Reservierung von Instanzen der Komponenten – Implementierungen. Es wird davon ausgegangen, dass die Abhängigkeiten über eingehende und ausgehende Ströme bei der Aushandlung der Verträge zu den `uses` – Klauseln mit ausgewertet werden.

Einlesen der Implementierungen

Einlesen der Komponenten – Implementierungen: Die Komponenten – Implementierungen können zum Start des Servers aber auch während der Laufzeit der Server – Anwendung eingefügt werden.

Beim Einlesen müssen die Implementierungen, die Schnittstellen und die *XML* – Dateien zur Spezifikation der Beziehungen und der *QoS* – Eigenschaften zur Verfügung stehen.

Für jede Komponenten – Implementierung werden daraufhin folgende Arbeitsschritte ausgeführt:

1. Ermittlung der Implementierung,
2. Ermittlung der angebotenen Komponenten – Spezifikationen (Schnittstelle für Zugriff auf die Geschäftsmethoden),

3. Ermittlung der verwendeten Schnittstellen,
4. Auslesen des *QoS* – Eigenschaften zur Implementierung und
5. Instanziierung einer Verwaltungsklasse für die zur Laufzeit zu erstellenden Objekte dieser Implementierung.

Entfernen von Implementierungen

Entfernen der Komponenten – Implementierungen: Zum Entfernen einer Komponenten – Implementierung ist es notwendig, ihre Instanzen zu löschen und die Referenz auf die Implementierungsbeschreibung (`Implementation`) zu verwerfen.

Entfernen zur Laufzeit

Soll eine Implementierung während der Laufzeit der Komponenten – Verwaltung entfernt werden, muss überprüft werden, ob eine ihrer Instanzen gerade ausgeführt wird, oder innerhalb einer Anfrage noch verwendet werden soll. In dem Fall führt das Löschen der Instanzen ohne Rückmeldung an die aufrufenden Implementierungen zu Fehlern.

Es stehen in einer solchen Situation folgende Lösungsalternativen zu Verfügung:

1. Sofortiges Sperren aller Zugriffe auf die Objekte der Komponenten – Implementierung. In diesem Fall müssen für die verwendeten Instanzen alternative Implementierungen, die der selben Schnittstelle genügen und geeignete *QoS* – Eigenschaften anbieten, gefunden werden.
2. Abwarten, bis alle Anfragen an die Objekte dieser Komponenten – Implementierung beendet sind. Desweiteren wird die Komponenten – Implementierung als nicht mehr verfügbar gekennzeichnet. Dies kann zum Beispiel durch Angabe eines *Flags* erfolgen.

Bei Nutzung der ersten Alternative treten folgende Probleme auf:

- Die Auswahl einer alternativen Implementierung einschließlich aufwendiger Aushandlung der Verträge (auch mit den verbundenen Komponenten) ist notwendig.
- Bei bereits vollständiger Aushandlung der Verträge und Start der Verarbeitung der Nutzeranfrage, muss diese unterbrochen werden, bis die alternativen Instanzen reserviert wurden. Dies bringt zusätzlich Probleme, wenn *QoS* – Zusagen bezüglich der Antwortzeit getätigt wurden, da diese in einer solchen Situation sehr wahrscheinlich gebrochen werden.
- Sollte sich die Ausführung gerade innerhalb der zu entfernenden Komponenten – Implementierung befinden, muss die Ausführung zurückgesetzt werden.

Der Vorteil der zweiten Alternative besteht in dem geringeren Arbeitsaufwand und wird daher in der Komponenten – Verwaltung umgesetzt. Sollte eine Komponenten – Implementierung allerdings aufgrund eines Ausfalls nicht mehr verfügbar sein, muss auf die erste Variante zurückgegriffen werden. In dieser Arbeit

wird ein Prototyp der Komponenten – Verwaltung umgesetzt, in dem von ausfallsicheren Komponenten – Implementierungen ausgegangen wird.

Nachdem keine weiteren Verweise auf die Instanzen einer Implementierung mehr existieren, können diese gelöscht werden. Die Informationen zu der Implementierung können ebenfalls entfernt werden (das `Implementation` – Objekt).

Sind alle Informationen zu den Implementierungen vorhanden, können Instanzen erstellt werden. Auf deren Verwaltung wird im folgenden Abschnitt eingegangen.

3.4.2 Verwaltung der Instanzen

Eine weitere wichtige Funktion der Implementierungs – Verwaltung ist die Verwaltung der zur Laufzeit erstellten Instanzen.

Komponententypen

Der Aufbau und die Funktionsweise der Komponenten orientiert sich an *Enterprise JavaBeans*. Sie besitzen demzufolge ein Schnittstelle zum Zugriff auf die Geschäftsmethoden. Mit Hilfe von *CQML*$^+$ können den Geschäftsmethoden einer Komponenten – Implementierung unterschiedliche *QoS* – Eigenschaften zugewiesen werden.

Instanzenerzeugung

Wie bereits beschrieben werden die Komponententypen *Stateful* und *Stateless Session Bean* in der Komponenten – Verwaltung umgesetzt. Die Anzahl der zu bildenden Instanzen hängt von der Art des *Session Beans* ab (Tabelle 3.2).

Session Bean	**Anzahl Komponenten**
Stateful Session Bean	Je *Session* wird eine eigene Instanz gebildet.
Stateless Session Bean	Zu jedem Profil wird ein eigenes Objekt erstellt. Ein Objekt kann von allen Clients verwendet werden. Zur Verteilung der Last können mehrere Instanzen je Profil verwaltet werden.

Tabelle 3.2: Anzahl der zur Laufzeit verwalteten Instanzen je Komponente

Obwohl alle Nutzer die Möglichkeit haben ein und dasselbe *Stateless Session Bean* – Objekt zu verwenden, kann während der Ausführung einer Geschäftsmethode nur ein Nutzer auf diese Instanz zugreifen. Die anderen Nutzer müssen in dieser Situation die Beendigung des Zugriffs abwarten. Dieses Blockieren ist unerwünscht, vor allem da sie mit Zusicherungen bezüglich der Antwortzeiten kollidieren. Zur Vermeidung solcher Situationen werden mehrere Instanzen je *Stateless Session Bean* erstellt und so die Last verteilt.

Instanzen – Verwaltung

Desweiteren ist zu beachten, dass in den einfachen Profilen unterschiedliche Ressourcenanforderungen erfüllt werden müssen. Aus diesem Grund werden zu jedem dieser Profile eigene Verwaltungen der Instanzen eingeführt. Diese Aufgabe übernehmen Instanzen – Manager (Abb. 3.14). Die in ihnen verwalteten Komponenten genügen der `Component` – Schnittstelle.

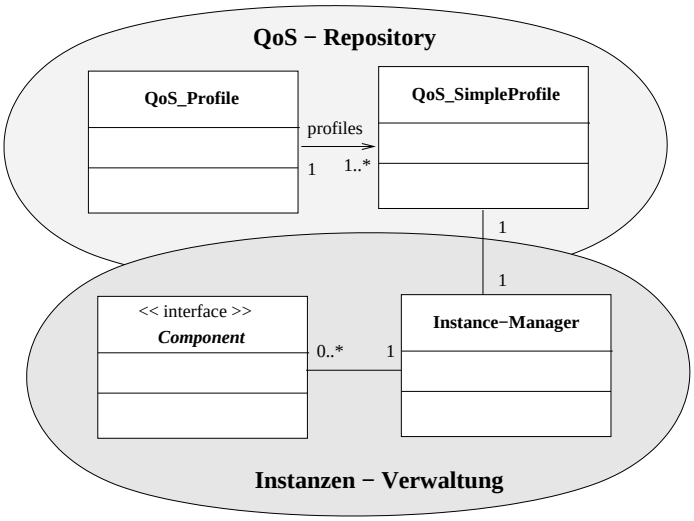


Abbildung 3.14: Verwaltung der Instanzen

Lebenszyklen

In diesem Abschnitt werden Eigenschaften und Lebenszyklen der *Session Beans* erläutert. Die vorgestellten Lebenszyklen orientieren sich an der *Enterprise JavaBeans* – Spezifikation.[21]

Stateful Session Beans

Stateful Session Beans

In einem *Stateful Session Bean* wird während der Kommunikation mit dem Client[2] ein Konversationszustand verwaltet. Dieser Zustand bleibt erhalten, bis der Client das Objekt freigibt. Jene Eigenschaft hat zur Folge, dass jeder Client seine eigene Instanz je *Stateful Session Beans* erhalten muss.

Bei der Gestaltung des Lebenszyklus eines *Stateful Session Beans* in der Komponenten – Verwaltung (Abb. 3.15) wurden die Zustände 'Existiert Nicht' und 'Bereit für Zugriff' umgesetzt.

Lebenszyklus

Fordert ein Client ein *Stateful Session Bean* an, wird ein neues Objekt erstellt. Dieses nimmt den Zustand 'Bereit für Zugriff' an, und der Client kann die Geschäftsmethoden ausführen. Beendet er seine Arbeit, muss er das Objekt freigeben. Die Referenz auf das Objekt wird daraufhin entfernt, und die reservierten Ressourcen werden sofort durch den Container freigegeben.

Timeout

Sollte eine Client – Anwendung ausfallen oder vergessen eine reservierte Instanz freizugeben, würde diese im dargestellten Lebenszyklus erhalten bleiben, bis die Server – Anwendung ihre Arbeit beendet. Zur Vermeidung dieser Situation muss eine *Timeout* – Funktion eingeführt werden.

[2] Es kann auch ein *Session Bean*, das auf dieses *Stateful Session Beans* zugreift, als Client betrachtet werden.

Abbildung 3.15: Lebenszyklus eines *Stateful Session Beans* in der Komponenten – Verwaltung

Caching

In der Abbildung 3.16 wurde der Lebenszyklus für ein *Stateful Session Bean* um einen weiteren Zustand 'Objekt erstellt und Ressourcen reserviert' erweitert.

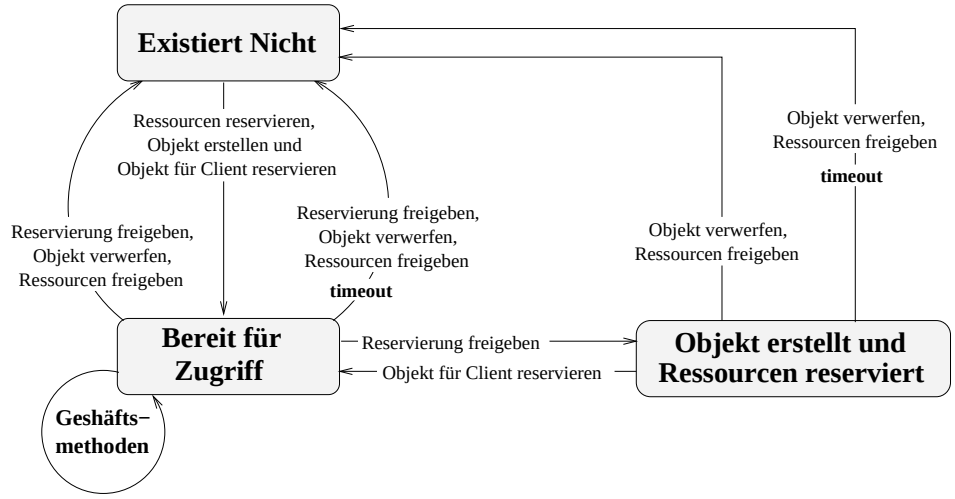


Abbildung 3.16: Erweiterung des Lebenszyklus eines *Stateful Session Beans* in der Komponenten – Verwaltung.

In dieser Erweiterung werden die Objekte nicht verworfen, nachdem ein Client sie freigegeben hat. Sie bleiben erhalten und können von anderen Clients wiederverwendet werden. In dem Fall müssen die Objekte einen definierten Anfangszustand annehmen, wenn sie in den Zustand 'Objekt erstellt und Ressourcen reserviert' übergehen.

Da die Ressourcen während dieses Zustandes reserviert bleiben, kann an dieser Stelle ebenfalls eine *Timeout* – Funktion eingeführt werden. In dem Fall wird das Objekt gelöscht, wenn es für einen langen Zeitraum nicht reserviert wurde.

Der Übergang vom Zustand 'Bereit für Zugriff' zu 'Existiert Nicht', der nicht durch einen *Timeout* eingeleitet wird, tritt zum Beispiel ein, wenn eine Komponenten – Implementierung entfernt wird oder die Server – Anwendung ihre Arbeit beendet.

In der Komponenten – Verwaltung wird die erweiterte Variante mit Zwischenspeicherung der Instanzen umgesetzt.

Stateless Session Beans

Stateless Session Beans

Da *Stateless Session Beans* keinen Konversationszustand verwalten, können alle Clients auf ein und dasselbe Objekt zugreifen. Beendet ein Client seinen Zugriff, muss das Objekt nicht entfernt werden, da es noch von anderen Clients wiederverwendet werden kann. Allerdings ist bei der Arbeit mit *Stateless Session Beans* zu beachten, dass nicht mehrere Clients gleichzeitig auf die Methoden des Objektes zugreifen können.

Alternativen der Reservierung

Für die Reservierung von Instanzen können zwei Alternativen umgesetzt werden:

1. Ein Client reserviert eine konkrete Instanz, die aber von vielen Clients gleichzeitig reserviert worden sein kann.
   Bei dieser Lösung wird von jeder einzelnen Instanz selbst registriert, von wie vielen Clients sie verwendet wird.
   Desweiteren hat diese Alternative zur Folge, dass ein Client bei allen Methodenaufrufen immer auf die selbe Instanz zugreift.

2. Die Reservierung des Clients verweist auf den Instanzen – Manager. Soll ein Zugriff auf eine Geschäftsmethode erfolgen, wird eine verfügbare Instanz ausgewählt. Die Instanzen – Verwaltung muss überwachen, wenn die Ausführung der Methode beendet wird, damit die Blockierung wieder aufgehoben werden kann.
   Bei dieser Lösung werden die Reservierungen von dem Instanzen – Manager registriert. Außerdem können aufeinander folgende Methodenaufrufe von unterschiedlichen Instanzen bearbeitet werden.

Bei Verwendung der ersten Variante wird die reservierte Instanz einmal übergeben. Die darauf folgenden Aufrufe der Geschäftsmethoden müssen durch die Instanzen – Verwaltung nicht überwacht werden. Wird die zweite Alternative genutzt, muss bei jedem Aufruf einer Geschäftsmethode eine neue Instanz ausgewählt werden. Zu beachten ist, dass die Auswahl einer Instanz nicht nur zusätzlich Systemressourcen kostet, sondern auch die Zugriffszeiten verlängert.

Die zweite Alternative hat jedoch den Vorteil, dass die Auslastung der Instanzen besser verteilt wird. Außerdem kann bei Blockierung einer Instanz eine andere ausgewählt werden.

Obwohl in der Komponenten – Verwaltung die zweite Variante umgesetzt wird, werden anschließend die Lebenszyklen beider Alternativen vorgestellt.

Alternative 1: Die *Stateless Session Beans* können ebenfalls die Zustände 'Existiert nicht' und 'Bereit für Zugriff' annehmen (Abb. 3.17).

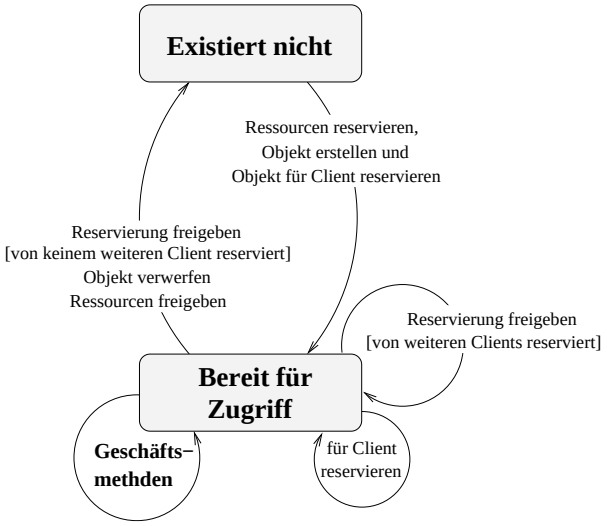


Abbildung 3.17: Lebenszyklus eines *Stateless Session Beans* in der Komponenten – Verwaltung

Lebenszyklus Alternative 1

Der Übergang vom Zustand 'Existiert Nicht' zu 'Bereit für Zugriff' erfolgt, wenn der Client das erste Mal eine Instanz anfordert. Bei der Erstellung werden die geforderten Ressourcen reserviert.

Wird ein *Stateless Session Bean* angefordert, für das bereits eine Instanz existiert, kann diese vom Client reserviert werden. Greifen jedoch zu viele Clients auf eine Instanz zu, müssen weitere Objekte erstellt werden. Für diese Funktion wird ein Grenzwert der maximal zulässigen Reservierungen angegeben. Beendet ein Client seine Bearbeitung, gibt er seine Reservierung frei. Durch die Registrierung der Anzahl der Reservierungen kann festgestellt werden, wenn die letzte Reservierung aufgehoben wird. In dem Fall kann das Objekt gelöscht und die reservierten Ressourcen freigegeben werden. Fordert ein Client im Anschluss daran ein Objekt an, muss dieses neu erstellt und die Ressourcen reserviert werden.

Caching

Eine Erweiterungsmöglichkeit bietet das Zwischenspeichern der Instanzen. In dem Fall bleiben die Ressourcen reserviert und die Instanzen erhalten, auch wenn kein Client auf sie verweist (Abb. 3.18). Um unnötiges blockieren von Ressourcen zu vermeiden, sollte eine *Timeout* – Funktion eingefügt werden.

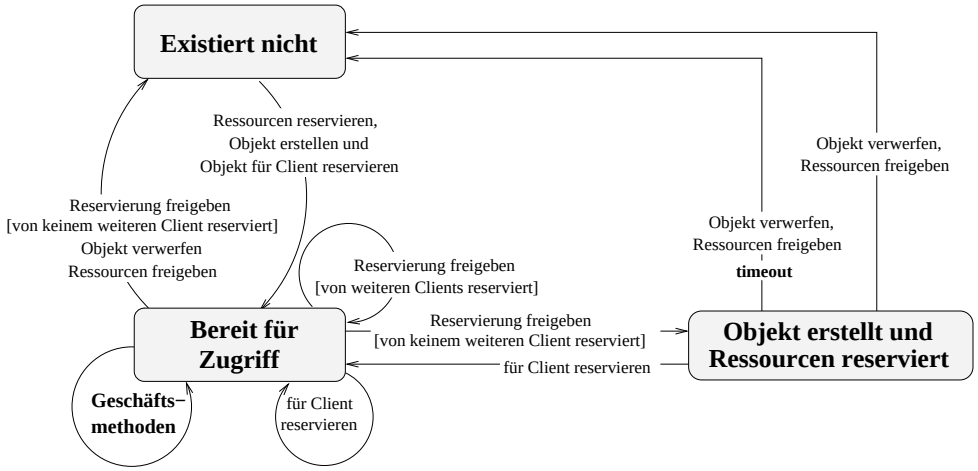


Abbildung 3.18: Alternative 1: Erweiterung des Lebenszyklus eines *Stateless Session Beans* in der Komponenten – Verwaltung

Alternative 2: Die Lebenszyklen verändern sich bei Verwendung dieser Alternative dadurch, dass eine Instanz nur zu Verarbeitung einer Methodenanfrage einem Client zugewiesen wird.

Lebenszyklus Alternative 2

Der Lebenszyklus (Abb. 3.19) ist mit der erweiterten Variante für *Stateful Session Beans* vergleichbar, da der Zustand 'Bereit für Zugriff' dem 'Objekt erstellt — Ressourcen reserviert' entspricht. Diese Situation tritt ein, wenn ein Objekt vom Container erstellt, aber keinem konkreten Client zugewiesen wurde. Dies erfolgt erst bei Ausführung einer Methode. Während dieser Zeit nimmt das Objekt den Zustand 'Zugriff erfolgt' an.

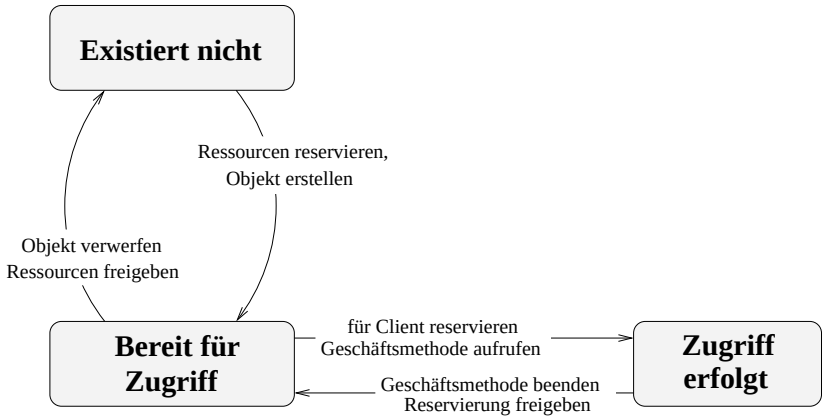


Abbildung 3.19: Alternative 2: Erweiterung des Lebenszyklus eines *Stateless Session Beans* in der Komponenten – Verwaltung

Fordert ein Client ein Objekt an, wird überprüft, wie viele Instanzen verfügbar sind und wie viele Reservierungen für diese registriert wurden. Ergibt die Prüfung, dass ein zulässiger Grenzwert überschritten wurde, muss ein neues Objekt erstellt werden. An den Client wird eine Referenz auf den Instanzen – Manager übergeben.

Grenzwert

Zur Ermittlung des Grenzwertes müssten von der Komponenten – Verwaltung die verfügbaren Systemressourcen und *QoS* – Anforderungen ausgewertet werden. Zur Vereinfachung dieses Verfahrens wird durch den Anwendungsentwickler ein maximales und minimales Verhältnis aus erstellten Instanzen und Reservierungen spezifiziert.

Instanzen – Manager

Instanzen – Manager

Da sich die Verwaltung der Instanzen in *Stateless* und *Stateful Session Beans* unterscheidet, wird die abweichende Funktionalität in `InstanceManagerStateless` und `InstanceManagerStateful` umgesetzt (Abb. 3.20).

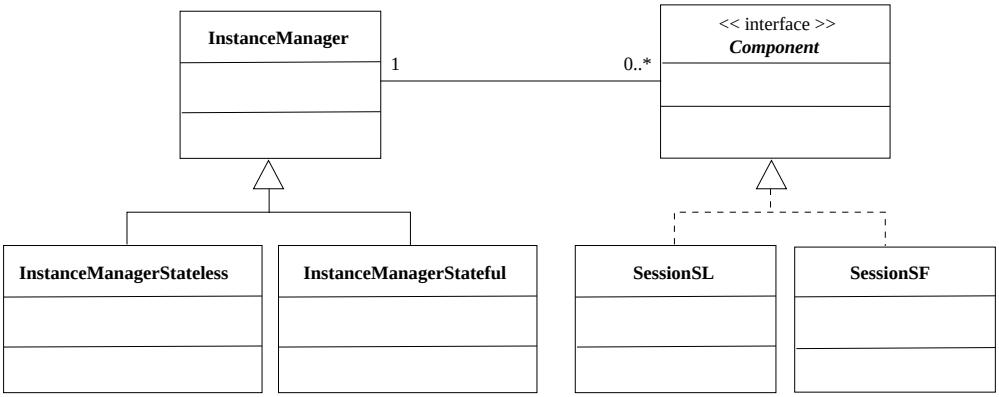


Abbildung 3.20: Verwaltung der Instanzen

In den vorangegangenen Abschnitten wurden die Funktionsbereiche *QoS* – Repository und Implementierungs – Verwaltung vorgestellt. In ihnen werden die Möglichkeiten zur Verwaltung der Komponenten und ihrer *QoS* – Eigenschaften geschaffen. Nachdem diese Informationen zur Verfügung stehen, kann die Anfrage eines Clients bearbeitet werden. Die darauf folgende Vertragsaushandlung und – durchsetzung ist Aufgabe des Vertragsmanagers. Seine Funktionsweise wird im folgenden Abschnitt vorgestellt.

3.5 Vertragsmanager

Die Aufgabe des Vertragsmanagers besteht in der Aushandlung und Durchsetzung von Verträgen. Bevor seine Funktionsweise in diesem Abschnitt ausführlich beschrieben wird, werden die verschiedenen Vertragstypen kurz vorgestellt. *Vertragsmanager*

Es besteht die Möglichkeit folgende Verträge in der Komponenten – Verwaltung umzusetzen: *Vertragstypen*

1. Ein *Vorvertrag* (*Precontract*) wird beim Einlesen der *QoS* – Eigenschaften zwischen kommunizierenden Komponenten – Implementierungen abgeschlossen, um den Aufwand für die Vertragsaushandlung während der Laufzeit zu verringern (S. 42).

2. Ein Vertrag (*OfferedContract*), der bei der Untersuchung der Eignung eines Profils einer Implementierung erstellt wird .

3. Der letzte Vertrag (*ExpectedContract*) wird von der aufrufenden an die aufgerufene Instanz übergeben und entspricht den geforderten Eigenschaften. Seine Einführung dient folgenden Zweck:
   Während der Ausführung der Anfrage sind von der aufgerufenen Instanz die zugesicherten Eigenschaften einzuhalten. Da die Anforderungen eventuell weniger einschränkend als die angebotenen Eigenschaften sind, sind stärkere Schwankungen zulässig, als die ursprüngliche Spezifikation vorschreibt. Die Funktionseinheit, die die Überwachung der Einhaltung von Zusicherungen übernimmt, wird durch den Vertrag über den größeren Toleranzbereich informiert.

In der aktuellen Version der Komponenten – Verwaltung wird nur der zweite Vertragstyp umgesetzt.

3.5.1 Vertragsaushandlung

Der Vertragsmanager wird aktiv, sobald eine Client – Anwendung eine Referenz auf ein *Session Bean* – Objekt anfordert (Abb. 3.21). *Initialisierung*

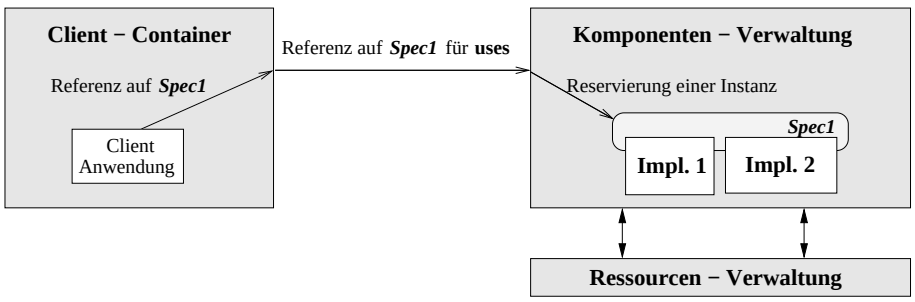


Abbildung 3.21: Bearbeitung einer Clientanfrage

Dazu spezifiziert er, zu welcher Schnittstelle er eine Instanz benötigt. Die Client – Laufzeitumgebung liest die geforderten *QoS* – Eigenschaften aus und übergibt die Anfrage an die Komponenten – Verwaltung im Server.

Vertragsaushandlung

Empfängt die Komponenten – Verwaltung die Anfrage beginnt die Vertragsaushandlung. Dazu werden folgende Arbeitsschritte ausgeführt:

1. Auswahl einer Implementierung der angeforderten Schnittstelle, die die benötigten *QoS* – Eigenschaften erfüllt,
2. Ermittlung aller Spezifikationen, die die ausgewählte Implementierung benötigt und Reservierung der abhängigen Instanzen,
3. Reservierung der Ressourcen über die Ressourcen – Verwaltung und
4. Erstellung einer Instanz sowie Reservierung dieser für die Client – Anwendung.

Komponenten – Netz

Durch Aushandlung der Verträge zwischen den abhängigen Komponenten – Implementierungen entsteht ein Netz aus kommunizierenden Instanzen. Ein Beispiel für ein solches Komponenten – Netz ist in der Abbildung 3.22 dargestellt.

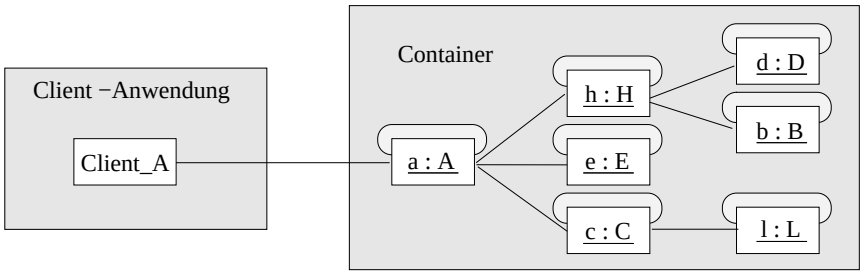


Abbildung 3.22: Komponenten – Netz

Nach Erstellung des Netzes und Reservierung der Ressourcen wird die Referenz auf die Wurzel an den aufrufenden Client übergeben.

Die Konzepte zur Aushandlung der Verträge zwischen kommunizierenden Komponenten – Implementierungen und mit der Ressourcen – Verwaltung werden nun vorgestellt.

3.5.1.1 Suche nach geeigneten Implementierungen

Implementierungsauswahl

Zu einer Komponenten – Implementierung wird registriert, welche Komponenten – Spezifikationen sie in ihrer Verarbeitung verwendet. Für jede abhängige Spezifikationen muss eine geeignete Implementierung ausgewählt und eine Instanz reserviert werden. Eine Implementierung gilt als geeignet, wenn sie ein Profil besitzt, dessen angebotenen *QoS* – Eigenschaften alle geforderten erfüllen.

Damit eine Eigenschaft erfüllt ist, hat der angebotene Wertebereich einer bewerteten Charakteristik gleich oder mehr einschränkend als der geforderte zu sein. Auf die Überprüfung der Profile wird ab Seite 44 ausführlich eingegangen.

Die Anordnung der Profile einer Implementierung, welche durch den Anwendungsentwickler zu spezifizieren ist, entspricht ihrer Güte. Bei einer Anfrage werden die angebotenen Profile der Reihe nach untersucht und alle geeigneten ausgewählt. Auf diese Weise kann sichergestellt werden, dass jede mögliche Realisierung überprüft wird. Durch Beibehaltung der Ordnung, der nach erfolgreicher Überprüfung erstellten Verträge, kann ihre Wertigkeit weiterhin berücksichtigt werden.

Suchalgorithmen

Für die Suche nach geeigneten Implementierungen stehen verschiedene Algorithmen zur Auswahl.

Ansätze für Algorithmen:

- Naiver Ansatz: Bei der Auswahl einer Implementierung aus der Menge aller Implementierungen einer Spezifikation wird in der Reihenfolge ihrer Registrierung vorgegangen. Sobald eine passende Implementierung gefunden wurde, wird die Suche abgebrochen. Die Überprüfung der angebotenen Profile wird ebenfalls abgebrochen, sobald ein geeignetes gefunden wurde.
- Eine bessere Lösung ist die Ermittlung aller möglichen Implementierungen mit anschließender Auswahl der besten Alternative. Für dieses Verfahren stehen zwei Umsetzungsvarianten zur Verfügung:
  1. Die Auswahl erfolgt für jede geforderte Schnittstelle separat. Es werden in dem Fall alle für diese Schnittstelle geeigneten Implementierungen herausgesucht und im Anschluss die Beste ausgewählt. Zu dieser wird daraufhin wieder für jede geforderte Spezifikation der Vergleich einzeln ausgeführt.
  2. Bei Anfrage eines Clients werden alle Lösungsvarianten für das gesamte Komponenten – Netz zusammengestellt. Im Anschluss kann aufgrund unterschiedlicher Kriterien (z. B. geringsten Ressourcenbedarf) das beste Netz ausgewählt werden.

Wertefunktion

Für den im zweiten Algorithmus vorgeschlagenen Vergleich der Lösungsalternativen ist eine Wertefunktion über die *QoS* – Eigenschaften erforderlich. Da die Analyse einer Wertefunktion außerhalb des Aufgabenbereichs dieser Arbeit liegt, werden von der Komponenten – Verwaltung zwar alle zulässigen Implementierungen ermittelt, aber eine beliebige ausgewählt. Ein Vergleich der Eigenschaften an dieser Stelle kann in einer erweiterten Version der Komponenten – Verwaltung eingefügt werden.

Optimierung der Vertragsaushandlung: Im Folgenden werden einige Möglichkeiten der Optimierung der Vertragsaushandlung vorgestellt. Die dargelegten Verbesserungsvorschläge werden aber im Implementierungs – Kapitel nicht berücksichtigt.

Vorvertrag

Eine Optimierungsmöglichkeit besteht darin, bereits beim Einlesen der Beziehungen zwischen Implementierungen und ihren geforderten Schnittstellen, mögliche Verträge auszuhandeln. Es werden die am besten geeigneten Implementierungen ausgewählt.

Die so ermittelten realisierbaren Beziehungen werden gespeichert (*Vorvertrag*) und zur Laufzeit ausgelesen. Sollen daraufhin für die von einer Implementierung geforderten Spezifikationen Verträge ausgehandelt werden, müssen die *QoS*– Eigenschaften nicht mehr verglichen werden.

Dieses Verfahren hat den Vorteil, dass die zeitaufwendige Überprüfung zur Laufzeit eingespart werden kann. Probleme treten aber auf, wenn Implementierungen entfernt oder hinzugefügt werden. In dem Fall müssen alle Vorverträge, die mit der zugehörigen Spezifikation abgeschlossen wurden, überarbeitet werden. In einer möglichen Lösung bleiben die alten Vorverträge erhalten, bis die neuen vollständig erstellt wurden, um den Verlauf der Vertragsaushandlung nicht zu behindern. Versucht ein Client in dieser Zeit eine nicht mehr verfügbare Implementierung zu reservieren, bekommt er eine Fehlermeldung. Er versucht daraufhin mit einem alternativen Vorvertrag eine Instanz zu reservieren.

Auswahl des optimalen Profils

Bei der Überprüfung der Profile einer Implementierung wird in der Reihenfolge ihrer Anordnung vorgegangen. Da die Profile sortiert sind, wird wahrscheinlich immer das beste Profil ausgewählt, dass eventuell auch die meisten Ressourcen benötigt. Damit Betriebsmittel eingespart werden, sollte nach einem Profil gesucht werden, welches die Anforderungen eventuell gerade so erfüllt. Dieses Verfahren kann durch eine dementsprechende Sortierung der Profile umgesetzt werden. Zum Beispiel wird das Profil als erstes aufgelistet, das die wenigsten Ressourcen benötigt.

Dieses Verfahren kann ebenfalls auf den Vergleich der geeigneten Implementierungen einer angeforderten Spezifikation ausgebaut werden. Es wird nicht der beste Vertrag ausgewählt, sondern der, der den Anforderungen gerade so entspricht.

In den anschließenden Erläuterungen wird der einfache Ablauf einer Suche nach einer geeigneten Implementierung veranschaulicht.

Beispiel Suche

Ablauf des Suchalgorithmus: Zur Abkürzung des Suchverfahrens wird in diesem Abschnitt die naive Lösung vorgestellt, d. h. die Suche wird beendet, sobald eine geeignete Implementierung einer Spezifikation gefunden wurde. Dieses Verfahren ist aber leicht zu erweitern, indem alle möglichen Implementierungen ausgewählt werden, um sie im Anschluss zu vergleichen.

In den folgenden Abbildungen wird die bereits vorgestellte Darstellung für Komponenten – Spezifikationen angewandt. Es müssen für alle im Beziehungsbereich eingetragenen Schnittstellen passende Implementierungen gefunden werden.

In dem in Abbildung 3.23 dargestellten Beispiel benötigt die Implementierung `A` eine Instanz einer Implementierung der Spezifikation `s`.

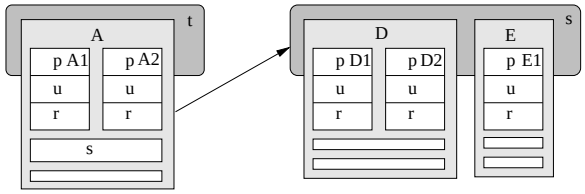


Abbildung 3.23: Suche nach einer geeigneten Implementierung (Beispiel 1)

Prüfung

Bei der Suche nach einer Implementierung wird das aktuell verwendete Profil *A1* in `A` mit den angebotenen Profilen in den Implementierungen der Spezifikation *s* verglichen. Dabei wird in der Reihenfolge der Registrierung der Implementierungen in der Spezifikation vorgegangen. Diese entspricht keiner Ordnung. In Abbildung 3.24 wird der Ablauf des Profilvergleichs dargestellt. Die Untersuchung der Profile wird ab Seite 44 detailliert vorgestellt.

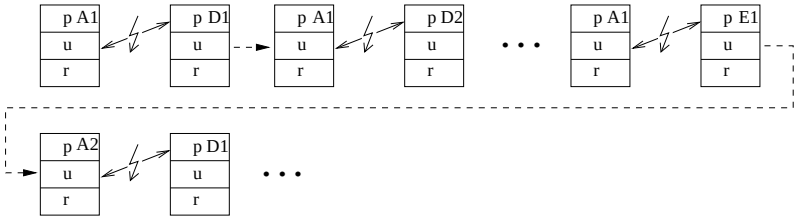


Abbildung 3.24: Vergleich der Profile

Im dargestellten Beispiel werden als Erstes die `uses` – Eigenschaften in *A1* mit den
`provides` – Eigenschaften im Profil *D1* verglichen. Entsprechen die Eigenschaften in *D1* nicht den Anforderungen von *A1*, wird die Eignung von *D2* überprüft. Ist auch *D2* ungeeignet ist die gesamte Implementierung nicht einsetzbar. Es wird daher die nächste Implementierung getestet. Kann keine Implementierung gefunden werden, die den Ansprüchen genügt, wird das Scheitern an die aufrufende Implementierung `A` signalisiert. Diese startet daraufhin die Suche erneut, falls ein alternatives Profil (*A2*) verfügbar ist.

Komplexe Abhängigkeiten

Wurde `A` in einem komplexeren Kontext von einer anderen Implementierung aufgerufen (siehe *Z1* in Abb. 3.25), kann nicht einfach ein alternatives Profil ausgewählt werden, da die Eignung von *A2* überprüft werden muss. Es stehen folgenden Alternativen zur Verfügung:

1. Da der Implementierung `A` die geforderten Eigenschaften *Z1* nach der Prüfung von *A1* immer noch bekannt sind, kann sie selbst untersuchen, ob ihr Profil *A2* den Anforderungen genügt oder

2. `A` meldet an `Z` das Scheitern. Diese Implementierung sucht daraufhin nach alternativen Implementierungen zu `Z1`.

Da die erste Variante die Möglichkeit der Einführung des verbesserten Suchalgorithmus ausschließt, indem die Implementierung `Z` bei der Auswahl des besten Vertrages umgangen wird, wird die zweite Variante bevorzugt und umgesetzt.

Abschluss oder Rückschritt

Hier wird die Suche fortgeführt, bis eine geeignete Implementierung gefunden wurde. Kann keine die Anforderungen erfüllen, muss schrittweise im reservierten Netz zurückgegangen werden. Bei jedem Schritt zurück werden alle bereits reservierten Ressourcen wieder freigegeben.

In dem in der Abbildung 3.25 dargestellten Beispiel müssen für das Profil `A1` in `A` geeignete Implementierungen der Schnittstellen `q` und `s` gefunden werden.

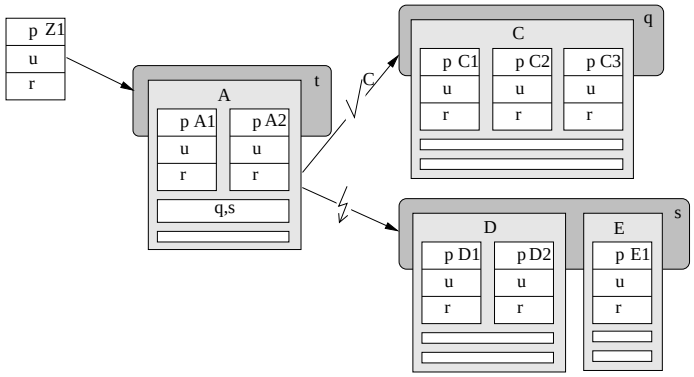


Abbildung 3.25: Suche nach einer geeigneten Implementierung (Beispiel 2)

Während für die geforderte Schnittstelle `q` eine Instanz der Implementierung `C` erfolgreich reserviert werden konnte, enthält `s` keine geeignete Implementierung. Daher muss ein Schritt zurück gegangen werden. Dabei wird die bereits reservierte Instanz der Implementierung `C` freigegeben.

Untersuchung der Eignung eines Profils

Profilprüfung

Unabhängig von der Art des eingesetzten Mechanismus zur Auswahl der Implementierungen muss ein Vergleich der *QoS*–Eigenschaften erfolgen. Dabei werden die von der zu überprüfenden Implementierung angebotenen Eigenschaften (`provides`) mit den von der aufrufenden Implementierung geforderten Eigenschaften (`uses`) verglichen.

Anweisungsprüfung

Da innerhalb einer Anweisung mehrere Charakteristiken, die sich auf unterschiedliche Spezifikationen beziehen, zusammengefasst werden können, ist eine Auswahl der Anweisungen, die vollständig von einer Spezifikation zu erfüllen sind, unmöglich. Daher erfolgt die Auswahl nur auf Ebene der bewerteten Charakteristiken.

Charakteristik-prüfung

Obwohl sich Charakteristiken ebenfalls auf mehrere Schnittstellen beziehen könnten, soll dieser Sonderfall in den Betrachtungen nicht einbezogen werden. Andernfalls müssten bei der Untersuchung eines Profiles alle beschriebenen Komponenten gleichzeitig geprüft werden, da ihre Zusammenarbeit den Wert beschreibt. Das erhöht die Komplexität des Vergleichs, und kann mit dem aktuellen Konzept nicht umgesetzt werden.

Nachfolgend wird der Algorithmus zur Überprüfung der Eignung eines Profils zur Erfüllung der geforderten Eigenschaften vorgestellt. Die dabei verwendeten Begriffe *'genügen'* oder *'erfüllen der Eigenschaften'* beschreiben, dass der angebotene Wertebereich gleich groß oder stärker einschränkend als der geforderten ist und somit die Anforderungen erfüllt. Dieser Zusammenhang soll an einem kleinen Beispiel verdeutlicht werden.

Es werden Werte im Bereich von `(10..50)` gefordert. In dem Fall sind `(30..50)` oder `(20)` zulässige Wertebereiche, die den Anforderungen genügen. Dagegen erfüllt `(5..50)` nicht die geforderten Eigenschaften, weil Werte auftreten können, die außerhalb des zulässigen Wertebereichs liegen.

Für jede von der untersuchten Spezifikation geforderte Eigenschaft muss mindestens eine angeboten werden, die die Anforderungen erfüllt (Antwort `true`), und es darf keine gegen die Forderungen verstoßen (Antwort `false`). Da einer Charakteristik innerhalb der `provides` – Klausel mehrfach Werte zugewiesen werden können, müssen alle angebotenen Eigenschaften überprüft werden. Lassen sich zwei bewertete Charakteristiken nicht vergleichen, da sie sich zum Beispiel auf unterschiedliche Charakteristik – Definitionen beziehen, wird eine Fehlermeldung erzeugt. Auf weitere Abbruchbedingungen wird im weiteren Verlauf dieser Untersuchungen eingegangen. Die Ergebnisse der Untersuchungen von Eigenschaften und die daraus resultierenden Maßnahmen werden in der Tabelle 3.3 zusammengefasst.

Ergebnis	**Beschreibung**	**Bedeutung für Vergleich**
mindestens ein `true`	Es war mindestens ein Eintrag enthalten, der die geforderten Eigenschaften erfüllt.	Die geforderte Eigenschaft gilt als erfüllt und die Untersuchung wird mit den anderen Eigenschaften fortgesetzt.
ein `false`	Die gerade untersuchte angebotene Eigenschaft hat gegen die Anforderungen verstoßen.	Die Suche wird abgebrochen! Die geforderte Eigenschaft gilt als nicht erfüllt und somit das gesamte Profil als ungeeignet.
am Ende nur Fehlermeldungen	Keine der angebotenen Eigenschaften kann die Einhaltung der Anforderung garantieren.	Die geforderte Eigenschaft gilt als nicht erfüllt und somit das gesamte Profil als ungeeignet.

Tabelle 3.3: Ergebnisse der Überprüfung einer geforderten Eigenschaft

Auf den folgenden Seiten wird der Vergleich von zwei bewerteten Charakteristiken beschrieben. Da in der $CQML^+$ – Spezifikation sowohl bewertete Charakteristiken als auch Anweisungen in einer *Konjunktiven Normalform* (*KNF*)

zusammengefasst werden, müssen diese Beziehungen bei der Untersuchung der Eignung eines Profils beachtet werden. Auf die sich daraus ergebenen Besonderheiten für angebotene und geforderte *QoS* – Eigenschaften wird im Anschluss an die Beschreibung des Vergleichs von Charakteristiken eingegangen (S. 51 ff.).

Einschränkungen

Bedingungen für den Vergleich von Charakteristiken: Werden den Charakteristiken in den `uses` – oder `provides` – Bereichen mehrfach Werte zugewiesen, muss sichergestellt werden, dass ihre Aussagen sich nicht widersprechen. Beim Vergleich der Charakteristiken wird davon ausgegangen, dass diese Überprüfung bereits erfolgt ist.

Die Untersuchung von Charakteristiken erfolgt nur auf einer Spezialisierungsebene. Soll die Prüfung auf über – und untergeordnete Charakteristiken erweitert werden, ist durch Auswertung der *values* – Klauseln zu ermitteln, ob sie die gleiche Semantik beschreiben, d. h. dass bei gleichen Eingabewerten das gleiche Ergebnis erzielt wird. Diese Untersuchung schließt die Prüfung der Parameter ein. Diese müssen nicht nur auf die gleichen Schnittstellen und / oder Ereignistypen verweisen, sondern auch auf die gleiche Weise in die Berechnung einfließen. Eine Änderung ihrer Reihenfolge wirkt sich auch auf das Ergebnis aus.

Da diese komplexen Untersuchungen nicht Bestandteil dieser Arbeit sind, werden nur Charakteristiken einer Ableitungsebene verglichen.

Beim Vergleich muss als Erstes festgestellt werden, ob sich die geforderte Charakteristik auf die gerade zu reservierende Spezifikation bezieht. Dazu wird der Bezeichner, der auf die zu reservierende Schnittstelle verweist, ausgewertet. Er wird beim Einlesen zusammen mit den Parametern gespeichert.

Charakteristikprüfung

Prüfung Charakteristik: Bevor eine angebotene Charakteristik mit einer geforderten verglichen werden kann, muss ihre Gleichartigkeit festgestellt werden. Dazu wird als Erstes überprüft, ob es sich um die gleichen Charakteristiken handelt. Dies geschieht durch einen Vergleich der Namen. Es können nur Charakteristiken mit gleichem Namen miteinander verglichen werden.

Nachdem festgestellt wurde, dass die gleichen Charakteristiken verglichen werden sollen, müssen die Parameter überprüft werden. Dabei wird untersucht, ob sich die Eigenschaft auf das gleiche Element der Anwendung bezieht. Es kann zum Beispiel eine Antwortzeit zu unterschiedlichen Methoden gefordert worden sein.

Parameterprüfung

Prüfung Parameter: Die von der Komponeten – Verwaltung untersuchten Parametertypen sind in der Abbildung 3.9 auf Seite 24 dargestellt. Bei ihrem Vergleich werden die Einflüsse der Übertragungsstrecke nicht berücksichtigt.

`EventSequence`

Es kann davon ausgegangen werden, dass eine von der aufrufenen Komponente ausgehende `EventSequence` (*SE*) einer empfangenen `EventSequence` (*SR*) in der aufgerufenen Komponente gegenübersteht (oder umgekehrt). Da bei der Überprüfung eines Profils immer eine aufrufende mit einer empfangende Komponente verglichen wird, können sich zwei Parameter vom Typ `EventSequence` nur entsprechen, wenn sie gegensätzliche Sequenztypen (*SE* oder *SR*) aufweisen.

Zeichenkettenvergleich

Bei der Prüfung anderer Parametertypen werden die Zeichenketten verglichen. Sind sie identisch, werden die gleichen Parameter beschrieben. Die Reduzierung der Untersuchung auf den Vergleich der Zeichenketten ist im Fall des `Event` – Typs eigentlich unzureichend. Es können innerhalb der gespeicherten Operation (`seqOp`) andere Parameter verwendet werden, die dann auch ausgewertet werden müssten. Da in den Profilen sehr wahrscheinlich unterschiedliche Namen für die Parameter verwendet werden, löst die Untersuchung des *Event* – Typs in dem Fall Fehlermeldungen aus, auch wenn die gleichen Ereignisse beschrieben wurden. Für den prototypischen Entwurf der Komponenten – Verwaltung soll der Vergleich der Zeichenketten aber ausreichen.

Wurden die zuvor beschriebenen Überprüfungen erfolgreich abgeschlossen, kann die Untersuchung der angebotenen Charakteristik durchgeführt werden. Dabei müssen Restriktionen beachtet werden, die im folgenden ausführlich erläutert werden.

Vergleich unter Beachtung der Restriktionen: Den Charakteristiken werden in den Anweisungen Werte zugewiesen. Dabei stehen in $CQML^+$ folgende Beschreibungsmittel für eine Beschränkung der Werte zur Verfügung:

- Vergleichsoperatoren (`=`, `<`, `<=`, `>`, `>=` und `<>`)
- Ermittlung statistischer Werte über einen definierten Zeitraum, z. B.
  - `Minimum`
  - `Maximum`
  - `Range`
- Einstufung der Wertzuweisung
  - garantierter Wert (`guaranteed`)
  - Wert unter besten Bedingungen (`best-effort`)
  - Wert unter besten Bedingungen mit Grenzwert (`best-effort with limit`)

Im Rahmen der Untersuchung, ob die durch ein Profil angebotenen Eigenschaften den Anforderungen eines anderen Profils entsprechen, sind die Zusammenhänge zwischen den Wertebeschränkungen der beiden Vertragspartner zu ermitteln. Diese Abhängigkeiten werden nachfolgend herausgestellt. Dabei werden die angebotenen Eigenschaften durch die Fußnote p für *provides* und die geforderten durch u für *uses* beschrieben (z. B. delay_p).

Vergleichspoperatoren

Vergleichsoperatoren: Die Anforderungen, die sich aufgrund der Vergleichsoperatoren, für die angebotenen Eigenschaften aus den geforderten ergeben, sind in der folgenden Tabelle 3.4 beschrieben. Es werden die Werte einer Charakteristik `x` verglichen.

geforderte Eigenschaft	**Anforderungen an angebotene Eigenschaften**
$x_u = 5$	Der angebotene Wert muss genau dem geforderten entsprechen, d. h. $x_p = 5$
$x_u < 5$	Der angebotene Wert muss kleiner sein, z. B. $x_p < 5$, $x_p < 3$ oder $x_p = 4$
$x_u <= 5$	Der angebotene Wert muss kleiner oder gleich dem Wert sein, z. B. $x_p < 5$, $x_p < 3$ oder $x_p = 5$
$x_u > 5$	Der angebotene Wert muss größer sein, z. B. $x_p > 5$, $x_p > 8$ oder $x_p = 6$
$x_u >= 5$	Der angebotene Wert muss größer oder gleich dem Wert sein, z. B. $x_p > 5$, $x_p > 8$ oder $x_p = 5$
$x_u <> 5$	Der angebotene Wert muss ungleich dem Wert sein, z. B. $x_p > 5$, $x_p < 5$ oder $x_p = 8$

Tabelle 3.4: Anforderungen an den Wert angebotener Eigenschaften

Die sich daraus ergebenden Bedingungen für den Vergleich zweier Charakteristiken werden in der Tabelle 3.5 zusammengefasst. Es werden die Vergleichsoperatoren der angebotenen (`p`) und der geforderten (`u`) Charakteristiken gegenübergestellt. Die davon abgeleiteten notwendigen Verhältnisse der Beträge sind als Ergebnisse der Gegenüberstellung dargestellt. Wird zum Beispiel ein Wert `x` $<_u$ `50` gefordert, und die angebotene Eigenschaft verwendet den Vergleichsoperator $\leq_p$, muss der angebotene Wert kleiner `50` sein.

		geforderter Relationen					
		$=_u$	$<_u$	$\leq_u$	$>_u$	$\geq_u$	$<>_u$
angebotene Relation	$=_p$	$=$	$<$	$\leq$	$>$	$\geq$	$<>$
	$<_p$	—	$\leq$	$\leq_{+1}$	—	—	$\leq$
	$\leq_p$	—	$<$	$\leq$	—	—	$<$
	$>_p$	—	—	—	$\geq$	$\geq_{-1}$	$\geq$
	$\geq_p$	—	—	—	$>$	$\geq$	$>$
	$<>_p$	—	—	—	—	—	$=$

Tabelle 3.5: Beschreibung der benötigten Verhältnisse der Beträge bei Verwendung der angegebenen Vergleichsoperatoren

Ergebnisse der Prüfung

Wird das Verhältnis eingehalten, garantiert die angebotene Charakteristik die Einhaltung der Anforderung und es wird der Erfolg der Untersuchung gemeldet. Kann diese Eigenschaft nicht erreicht werden, so wird aber nicht der Verstoß gegen die Anforderungen festgestellt, da der angebotene Wertebereich noch durch eine andere Charakteristik beschränkt werden kann. Die Suche nach einer geeigneten Charakteristik wird daher fortgesetzt. Dieser Zusammenhang wird an einem Beispiel beschrieben. Es wird x_u `> 50` gefordert. Die angebotene Eigenschaft x_p `> 30` enthält unzulässige Werte wie zum Beispiel `31` und `32` und wird daher abgelehnt. Der Charakteristik können mehrere Werte zugewiesen wer-

den, die den Wertebereich von $\texttt{x}_p$ weiter einschränken (z. B. $\texttt{x}_p$ `>` `60`). Dies ist zulässig und garantiert in diesem Beispiel die Erfüllung der Anforderung.

Wenn aufgrund der verwendeten Vergleichsoperatoren keine festen Zusagen über die Eignung einer Charakteristik gemacht werden können (siehe '—' in Tabelle 3.5), wird der Vergleich abgebrochen und mit der nächsten angebotenen Charakteristik fortgesetzt. Da in dem Fall keine Aussage über Einhaltung oder Verstoß gegen die Anforderungen gemacht werden kann, wird eine Fehlermeldung ausgegeben. Die beschriebene Situation tritt zum Beispiel ein, wenn $\texttt{x}_u$ `<` `50` gefordert und $\texttt{x}_p$ `>` `10` angeboten wird. Der angebotene Wert erfüllt nur teilweise die Anforderungen, da in dem Fall auch die Werte `50, 51, ...` auftreten können. Der Abbruch an dieser Stelle bedeutet nicht, dass die angebotene Charakteristik gegen die geforderte Eigenschaft verstößt, sondern nur, dass keine Garantie über die Einhaltung der Werte gegeben werden konnte.

Grenzwerte

In der Tabelle werden die Ausdrücke '$\leq_{+1}$' oder '$\geq_{-1}$' verwendet. Sie beschreiben, dass um zu gewährleisten, dass wirklich alle gültigen angebotenen Werte ermittelt werden, der geforderte Wert um die kleinste Einheit des Wertebereichs erhöht oder verringert werden muss. Diese Situation soll an einem kleinen Beispiel verdeutlicht werden.

Geforderte Eigenschaft: $\text{delay}_u <= 20$

Wird bei der angebotenen Eigenschaft der Vergleichsoperator '$<$' verwendet, erfüllen alle Werte die kleiner oder gleich 20 sind die Anforderungen. Die folgende Eigenschaft ist aber auch zulässig (*Integer* – Wertebereich):

Angebotene Eigenschaft: $\text{delay}_p < 21$

Bei Einbeziehung dieser Grenzwerte ergeben sich Probleme in den Datentypen, weshalb die Ungenauigkeit beim Vergleich in der Komponenten – Verwaltung nicht berücksichtigt wird. Zu den Problemen gehören:

1. *Numeric*: In dem Wertebereich werden *Integer*, *Real* und *Natural* zusammengefasst. Da die kleinste Einheit nicht in allen übereinstimmt, müssen die Wertebereiche wieder getrennt betrachtet werden.

2. *ungeordnete Enumeration und Set*: Da keine Ordnung existiert, können die Wert nicht erhöht oder verringert werden.

3. *geordnete Enumeration und Set*: Die kleinste Einheit wird nicht definiert.

Verstoß gegen Anforderungen

In den vorherigen Betrachtungen war es nur möglich zu ermitteln, ob eine Anforderung eingehalten werden kann. Die eindeutige Feststellung, dass gegen eine Anforderung verstoßen wird, ist nur bei Verwendung der Vergleichsoperatoren *equal* und *unequal* möglich. Wird zum Beispiel $\texttt{delay}_u$ `<>` `21` gefordert, löst der angebotene Wert $\texttt{delay}_p =$ `21` eine Fehlermeldung aus. Das gleiche gilt auch für $\texttt{delay}_u =$ `21` und $\texttt{delay}_p$ `<>` `21`.

Statistische Werte: In der aktuell zu entwickelnden Komponenten – Verwaltung werden die statistischen Werte *Range*, *Minimum* und *Maximum* ausgewertet. *Range* wird beim Einlesen in die zwei Einträge *Minimum* und *Maximum* zerlegt, wobei beide Werte erfüllt werden müssen. Befindet sich die bewertete Charakteristik in einer *OR* – Beziehung, muss diese aufgesplittet werden. Bei der Überführung in *Minimum* und *Maximum* werden die runden Klammern '(' und ')' in '<' und '>', sowie die eckige Klammern '[' und ']' in '<=' und '>=' umgewandelt. Dabei ist zu beachten, dass für *Range* nur von der Verwendung des Vergleichsoperator '=' ausgegangen wird.

Range

Min und Max

Die statistischen Werte *Minimum* und *Maximum* werden über einen bestimmten Zeitraum ermittelt. Bei der Untersuchung der Eignung einer Charakteristik unterscheidet sich ihre Bedeutung aber nicht zu der einer direkten Wertzuweisung. Daher werden sie bei der Überprüfung einer Charakteristik nicht ausgewertet.

Diskriminator

Diskriminatoren: Beim Vergleich der Diskriminatoren ist die Auswertung ihrer Bedeutung wesentlich. Diese ist für geforderte und angebotene Eigenschaften in der Tabelle 3.6 gegenübergestellt.

Diskriminator	**Bedeutung für**	
	Geforderte Eigenschaften	**Angebotene Eigenschaften**
`Guaranteed`	Die Eigenschaft muss erfüllt werden.	Die Eigenschaft kann auf jeden Fall garantiert werden.
`BestEffort`	BestEffort beschreibt den idealen Wert.	Es ist der Wert, der unter besten Umständen angeboten werden kann.
`BestEffort` mit `Limit`	BestEffort beschreibt den idealen Wert. Das Limit muss auf jeden Fall erfüllt werden.	BestEffort beschreibt den Wert, der unter besten Bedingungen erfüllt werden kann. Limit ist der Grenzwert, der auf jeden Fall erreicht werden sollte. Wird er unterschritten, wird der Client informiert (`Threshold`) oder der Service abgebrochen (`Compulsory`).

Tabelle 3.6: Unterscheidung der Diskriminatoren

Die Verwendung von *best – effort* zur Beschreibung des geforderten Wertes ist nicht sehr aussagekräftig, da in [6] keine Maßnahmen bei Abweichung vorgegeben sind. Bei der Untersuchung der Eignung eines einzelnen Profils gelten daher alle geforderten *best – effort* Werte als erfüllt. Genügt das Profil aber den anderen gestellten Anforderungen, wird ein Vertrag erstellt. Nachdem die aufrufende Komponente von allen geeigneten Implementierungen der Spezifikation einen Vertrag erhalten hat, können diese verglichen werden. Dabei wird untersucht, welche Implementierung die geforderten *best – effort* – Anforderungen am Besten erfüllt. Wird ein *best – effort* Wert mit *Limit* spezifiziert, wird immer die Einhaltung des *Limits* überprüft.

Bei der Überprüfung der bewerteten Charakteristiken werden die Beträge in den *Diskriminatoren* verglichen. Die beim Vergleich verwendeten Einträge werden in der folgenden Tabelle 3.7 dargestellt. *Diskriminatorvergleich*

angebotener Discriminator		**geforderter Discriminator**		
		Guaranteed (Guaranteed$_u$)	**BestEffort**	**BestEffort mit Limit** (limit$_u$)
	Guaranteed	Guaranteed$_p$	kein Vergleich	Guaranteed$_p$
	BestEffort	BestEffort$_p$	kein Vergleich	BestEffort$_p$
	BestEffort mit Limit	Limit$_p$	kein Vergleich	Limit$_p$

Tabelle 3.7: Zu vergleichende Werte bei Verwendung der Beschränkungen

Beim Vergleich von garantierten und bestmöglichen Werten werden vom Anwendungsentwickler Vergleichsoperatoren spezifiziert, für die alle zuvor beschriebenen Bedingungen zu beachten sind. Für die Arbeit mit *Limits* hat Aagedal in [6] keine Angaben zu dem Vergleichsoperator gemacht. Es wurde daher entschieden, den Operator des *best – effort* Wertes zu übernehmen. Laut Spezifikation [6] besteht der *Limit* – Wert aus einem Element. Dies gilt auch für den *Set* – Wertebereich.

Die bewerteten Charakteristiken können in den Anweisungen durch *OR* – und *AND* – Beziehungen zusammengefasst werden. Das gleiche gilt für die Verwaltung der Anweisungen in den Profilen. Die sich daraus für den Vergleich ergebenen Bedingungen werden im folgenden Abschnitt beschrieben.

Auswertung der Einträge in der KNF: Bei Verwendung der *Konjunktiven Normalform* sind zur Auswertung der Einträge, die sich in einer *OR* – Beziehung befinden, besondere Bedingungen zu beachten. Da sich die erforderlichen Maßnahmen bei geforderten und angebotenen Eigenschaften unterscheiden, werden sie gesondert vorgestellt.

Geforderte Eigenschaften: Stehen geforderte bewertete Charakteristiken in einer *OR* – Beziehung, ergibt sich für den Vergleich von Profilen, dass die gesamte Aussage *wahr* ist, sobald eine der Anforderungen erfüllt ist. Daraus leitet sich folgendes Problem ab: *KNF in uses*
Beziehen sich die enthaltenen Charakteristiken auf unterschiedliche Spezifikationen, kann beim Vergleich einer ausgewählten Implementierung, nicht überprüft werden, ob die 'fremden' Eigenschaften erfüllt werden können.

In dieser Arbeit wurde dieses Problem gelöst, indem eine der bewerteten Charakteristiken, die sich auf die aktuelle Spezifikation bezieht, erfüllt werden muss. Diese Erweiterung verschärft die Anforderungen, garantiert aber ein auf jeden Fall zulässiges Ergebnis. Bezog sich keine Eigenschaft auf die Spezifikation, muss auch der gesamte Ausdruck nicht erfüllt werden.

Die Anforderungen können auf Anweisungen ausgebaut werden, d. h. sobald alle Eigenschaften einer Anweisung der *OR* – Beziehung erfüllt sind, gilt der gesamte Ausdruck als erfüllt.

KNF in provides

Angebotene Eigenschaften: Werden angebotene bewertete Charakteristiken, die sich in einer *OR* – Beziehung befinden, überprüft, ob sie einer geforderten Eigenschaft genügen, muss nur ein Eintrag den geforderten Wert erfüllen. Die anderen dürfen auch unzulässige Werte anbieten (z. B. $\texttt{x}_p$ `<>5` bei $\texttt{x}_u$ `= 5`). Desweiteren könnten verschiedene geforderte Charakteristiken durch unterschiedliche Einträge erfüllt werden. Das ist zulässig, da sie nicht in einer *XOR* – Beziehung stehen. Es ist in dieser Situation aber schwierig zu bestimmen, welche Eigenschaften zur Laufzeit tatsächlich eingehalten werden können. Falls nicht alle zugesicherten Werte erfüllt werden, löst das Fehler aus. Aus diesem Grund ist es erforderlich, dass der Vertrag mit den geforderten Eigenschaften der Instanz zugewiesen wird (S. 39).

Verstößt jede angebotene bewertete Charakteristik gegen die Forderung (Prüfung ergibt `false`), gilt der gesamte Ausdruck als Verstoß. Hier wird die Suche abgebrochen und das Profil gilt als ungeeignet. War mindestens ein Eintrag enthalten, der nicht direkt gegen die Forderungen verstieß, sondern 'nur' als ungeeignet abgeleitet wurde, da er sich zum Beispiel auf eine andere Charakteristik bezog, wird dieser ausgewählt. In dem Fall wird der gesamten *OR* – Ausdruck als ungeeignet gekennzeichnet und die Suche nach geeigneten bewerteten Charakteristiken, die die Anforderungen erfüllt, fortgesetzt.

Die gleiche Verfahrensweise wird auch in Anweisungen, die sich in einer *OR* – Beziehung befinden, umgesetzt.

Abschluss der Prüfung eines Profils: Konnte für alle geforderten Eigenschaften die Erfüllung ihrer Anforderungen abgeleitet werden, wird ein Vertrag über die angebotenen Eigenschaften erstellt und an die aufrufende Implementierung übergeben.

Nach Auswahl des besten Vertrages wird eine Instanz der Implementierung angefordert. Auf die eventuell notwendige Reservierung der Ressourcen soll nun eingegangen werden.

3.5.1.2 Reservierung der Ressourcen

Reservierung

Nachdem aus allen geeigneten Angeboten eine Implementierung ausgewählt wurde, muss eine Instanz reserviert und ins Komponenten – Netz der Client – Anfrage eingefügt werden. Als Erstes wird geprüft, ob eine zwischengespeicherte Instanz verfügbar ist. Sollte das nicht der Fall sein, müssen die benötigen Ressourcen reserviert und ein neues Objekt erstellt werden. Dafür wird die *resources* – Klausel des ausgewählten Profils ausgewertet und die für den Zugriff auf die Ressourcen – Verwaltung notwendige Ressourcenanfrage generiert und übergeben.

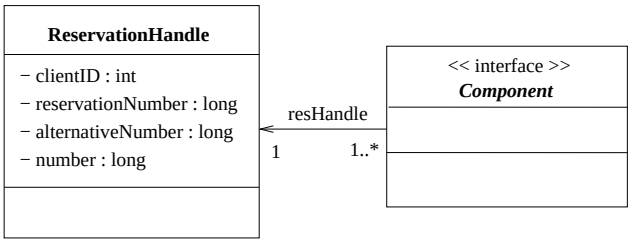


Abbildung 3.26: Handle zur Kennzeichnung einer Reservierung

Handle

Können die geforderten Ressourcen reserviert werden, gibt die Ressourcen – Verwaltung einen *XML* – `String` mit der Beschreibung der reservierten Ressourcen zurück. Diese Informationen werden für den Zugriff auf die Ressourcen benötigt. Der Vertragsmanager wertet sie aus und erstellt ein *Handle*, der an die Instanzen gebunden wird (Abb. 3.26). Eine ausführliche Beschreibung der Einträge im *Handle* ist der Vorstellung der Schnittstelle der Ressourcen – Verwaltung ab zu entnehmen (S. 5 ff.).

Nach Abschluss der Reservierung übernimmt das *Ressourcen – Proxy* die Aufgabe, die Verwendung der reservierten Ressourcen durchzusetzen. Zu beachten ist, dass bereits bei der Erstellung der Instanz die reservierten Ressourcen zu verwenden sind.

Stehen die benötigten Ressourcen nicht zur Verfügung meldet die Ressourcen – Verwaltung das Scheitern. Die ausgewählte Implementierung kann somit nicht verwendet werden, und die Suche ist fortzusetzen.

3.5.1.3 Komponenten – Netze

Verwaltung abhängiger Instanzen

Nach Aushandlung aller Verträge entsteht ein Netz aus kommunizierenden Instanzen. Die Verweise auf die abhängigen Instanzen werden in jedem Objekt selbst verwaltet (Abb. 3.27).

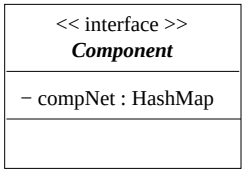
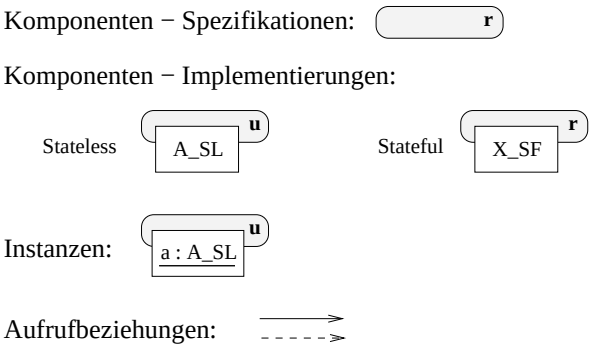


Abbildung 3.27: Verwaltung der abhängigen Instanzen in `compNet`

Besonderheiten der Netze

In den nun folgenden Betrachtungen sollen die Besonderheiten dieser Komponenten – Netze diskutiert werden. Dabei wird auf die zwei unterschiedlichen Typen von Komponenten *Stateful* und *Stateless Session Beans* eingegangen.

In den Abbildungen von Komponenten – Netzen wird folgende Symbolik verwendet:



Netze aus Stateful Session Beans

Nur *Stateful Session Beans*: Unproblematisch ist die Erstellung von Netzen, die nur aus *Stateful Session Beans* bestehen, ist unproblematisch. Wie der Abbildung 3.28 zu entnehmen ist, reserviert jeder Client seine eigenen Instanzen.

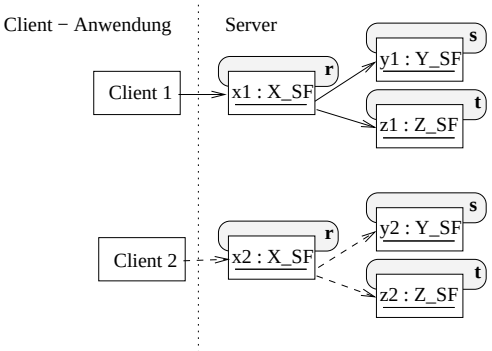


Abbildung 3.28: Komponenten – Netze: nur aus *Stateful Session Beans* bestehend

Die Arbeit mit zu großen Netzen aus diesen Beans sollte aber vermieden werden, da über den gesamten Verarbeitungszeitraum der Konversationszustand erhalten bleiben muss. Wird eine der eingebundenen Instanzen für längere Zeit nicht angesprochen, kann sie ihren *Timeout* – Zeitpunkt erreichen. Bei *Timeout* wird die Instanz zerstört und ihr Konversationszustand geht verloren. Auf diese Weise kann das gesamte Komponenten – Netz unbrauchbar werden.

'The chain of stateful session beans is only as strong as its weakest link.'[21]

Netze aus Stateless Session Beans

Nur *Stateless Session Beans*: Die Verarbeitung von Netzen, die nur aus *Stateless Session Beans* bestehen, weist ebenfalls keine Probleme auf (Abb. 3.29). Die Instanzen können von allen Clients gleichzeitig genutzt werden.

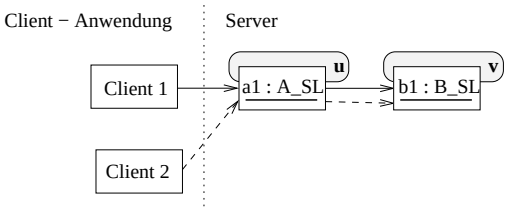


Abbildung 3.29: Komponenten – Netze: nur aus *Stateless Session Beans* bestehend

Bei der für die Komponenten – Verwaltung gewählten Lösung wird jedoch nicht direkt auf die *Stateless Session Beans* verwiesen, stattdessen wird ein Verweis auf den Instanzen – Manager in das Netz aufgenommen. Versucht ein Client oder eine Instanz aus dem Komponenten – Netz auf die reservierte Verwaltungsklasse zuzugreifen, wählt diese eine verfügbare Instanz aus und leitet den Aufruf weiter. Ist kein Objekt verfügbar, muss gegebenenfalls ein neues erstellt werden. Da dafür die Ressourcen und abhängigen Instanzen zu reservieren sind, wird wahrscheinlich gegen die eventuell spezifizierte *QoS* – Anforderung 'Antwortzeit' verstoßen. Weil in dem Fall die gesamte Ausführung der Anfrage scheitert, sollten die *QoS* – Eigenschaften schon bei Reservierung der Instanzen beachtet werden, so dass genügend Objekte zur Verfügung stehen.

Kombination von *Stateful* und *Stateless Session Beans*: In den folgenden Abbildungen werden Netze, die aus *Stateful* und *Stateless Session Beans* bestehen, betrachtet.

Stateless folgt auf Stateful

a) Stateless Session Beans folgen auf Stateful Session Beans: In Abbildung 3.30 folgen nur noch *Stateless Session Beans* auf *Stateful Session Beans*. Diese Netze stellen ebenfalls keine Probleme dar. Es sollte aber das beschriebene Problem bei Ketten von *Stateful Session Bean* beachtet werden.

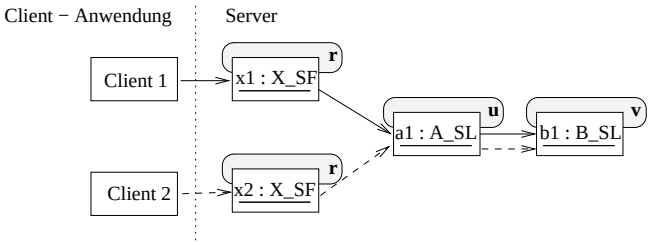


Abbildung 3.30: Komponenten – Netze: nach *Stateful Session Beans* folgen *Stateless Session Beans*

Stateful folgt auf Stateless

b) Stateful Session Beans folgen auf Stateless Session Beans: Die Abbildung 3.31 zeigt ein Beispiel für diese Beziehung. Obwohl eine solche Kombination theoretisch auftreten kann, ist dieser Aufbau eigentlich nicht sinnvoll, denn

die Besonderheit von *Stateful Session Beans* der Konversationszustand kann bei unabhängigen Methodenaufrufen in dem vorangestellten *Stateless Session Beans* nicht übernommen werden.

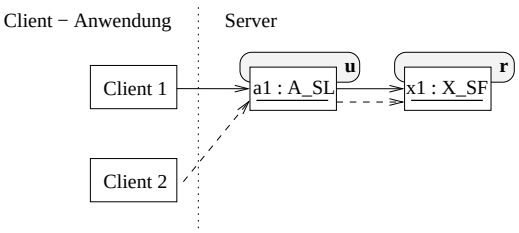


Abbildung 3.31: Komponenten – Netze: nach *Stateless Session Beans* folgen *Stateful Session Beans*

Da eine solche Beziehung zusätzliche Probleme bei der Realisierung der Komponenten – Verwaltung aufweist, und die Funktionalität des *Stateful Session Beans* auch von einem *Stateless Session Beans* umgesetzt werden kann, wird diese Kombination in Netzen grundsätzlich ausgeschlossen.

Caching von Netzen

Zwischenspeichern von Netzen: Werden die Instanzen zwischengespeichert, besteht die Möglichkeit die Verweise auf die abhängigen Instanzen ebenfalls zwischenzuspeichern. Der Vorteil dieses Verfahrens ist, dass die Verträge nicht bei jeder Anfrage neu ausgehandelt werden müssen. Das gesamte Komponenten – Netz kann wiederverwendet werden. Dieses Verfahren weist aber auch Probleme auf.

Wird im abhängigen Komponenten – Netz auf *Stateful Session Beans* verwiesen, sind diese für anderweitige Zugriffe blockiert. Sie gelten weiterhin als reserviert, auch wenn das aufrufende Objekt freigegeben wurde. Wird die aufrufende Instanz für einen langen Zeitraum nicht reserviert, werden Ressourcen unnötig blockiert. Beim Zwischenspeichern von abhängigen *Stateless Session Bean* – Instanzen tritt dieses Problem nicht auf, da die Reservierung nicht direkt auf das Objekt sondern auf den Instanzen – Manager erfolgt.

Eine Lösung dafür ist die Spezifikation einer maximalen Anzahl von nicht reservierten, zwischengespeicherten Instanzen. Bei Überschreitung dieses Grenzwertes wird eine Instanz gelöscht. Die Überprüfung des Wertes erfolgt bei jeder Freigabe einer Reservierung.

Eine alternative Lösung des Problems bei *Stateful Session Beans* bietet die *Timeout* – Funktionalität. Wurde eine Instanz für einen längeren Zeitraum nicht reserviert, wird sie gelöscht und die reservierten abhängigen Instanzen freigegeben.

Timeout

Zur Umsetzung der Funktionalität wird in allen Instanzen ein *Timeout* verwaltet, welches die maximale Zeitspanne zwischen zwei Reservierungen beschreibt. Beim Start des Servers muss der Wert festgelegt werden. Gibt ein Client seine Reservierung frei, wird die Stoppuhr gestartet. Läuft die Zeit ab, und es ist keine Reservierung erfolgt, wird der Instanzen – Manager informiert, dass diese

Instanz zu löschen ist. Erfolgt eine Reservierung der Instanz, wird das Warten auf den *Timeout* abgebrochen.

Bedingungen für Abbruch

Beendigung der Reservierung und Schleifenbildung: Die Reservierung von abhängigen Komponenten wird fortgesetzt bis eine der folgenden Bedingungen eintritt:

1. Es wird eine Instanz einer Komponenten – Implementierungen reserviert, die keine weiteren Spezifikationen in ihrer Verarbeitung benötigt.
2. Es wird auf eine Schnittstelle verwiesen, die bereits in der Kette eingebunden wurde. In dem Fall können zwei Situationen eintreten, die in Abbildung 3.32 visualisiert sind:
   (a) Verweis auf existierende Instanz oder
   (b) neue Instanz reservieren.

Schleifenbildung

Die Alternative 2a beschreibt den Sonderfall der Schleifenbildung. Da bei Verwendung der *CORBA IDL* zur Spezifikation der Abhängigkeiten keine Beziehungen auf Instanzenebene spezifiziert werden können, kann bei der Vertragsaushandlung nicht erfasst werden, ob auf die gleiche Instanz zu verweisen ist, oder eine neue benötigt wird. Es müsste daher folgende naiver Lösungsvariante angewandt werden.

Naiver Ansatz

Naiver Ansatz: Da keine Spezifikation der Beziehungen auf Instanzenebene vorhanden ist, wird die Reservierung abgebrochen, wenn auf eine Komponenten – Spezifikation verwiesen wird, die bereits in das Netz eingebunden wurde. Andernfalls kann eine immer wiederkehrende Reservierung der gleichen Kette auftreten (Abb. 3.32), da keine Abbruchbedingungen spezifiziert wurden. Die Reservierung der Ressourcen wird bei diesem Abbruch als erfolgreich beendet behandelt.

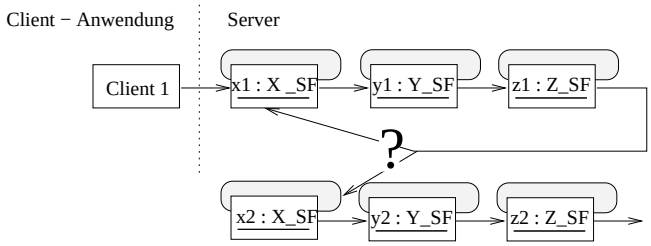


Abbildung 3.32: Komponenten – Netze, mit Schleifenbildung

Für die geforderte Komponenten – Spezifikation wird keine Instanz reserviert, da die Möglichkeit besteht, dass das bereits existierende Objekt zu verwenden ist. In dem Fall wird die Instanz zum Beispiel als Parameter übergeben. Wird ein neues Objekt benötigt, muss das Netz erweitert werden.

Netzerweiterungen

Für solche Situationen der Netzerweiterung müssen der Komponenten – Verwaltung genügend Ressourcen zur Verfügung stehen, so dass die Reservierung

schnell und auch erfolgreich durchgeführt werden kann, um die bereits begonnene Anfrage nicht scheitern zu lassen. Die Erweiterung der Netze während der Ausführung einer Anfrage ist ein komplexes Problem, das im Rahmen dieser Arbeit nicht weiter behandelt werden kann.

Um Schleifenbildung während der Reservierung zu erkennen, müsste im Komponenten – Netz zurückgegangen werden. Weil dieses aber noch nicht erstellt wurde, wird eine Netzbeschreibung ausgewertet. Da nur der aktuell ausgeführte Pfad von Interesse ist, genügt es bei jeder Reservierungsanfrage eine Liste der Namen eingebundener Schnittstellen zu übergeben und diese in jeder Implementierung zu erweitern. Wird eine Komponenten – Spezifikation benötigt, deren Name in der Liste steht, wird die Reservierung dieser Kette beendet.

Da eine Erweiterung der Netze während der Ausführung einer Anfrage vermieden werden sollte, muss die Möglichkeit geschaffen werden, folgende erweiterte Lösung umzusetzen.

Erweiterter Ansatz

Erweiterter Ansatz: Es existiert eine Spezifikation, die die Beziehungen auf Instanzenebene beschreibt. Somit kann während der Reservierung leicht erkannt werden, ob eine neue Instanz benötigt wird.

Da diese Beschreibungssprache momentan nicht zur Verfügung steht, sollen nachfolgend einige Möglichkeiten der Erweiterung der *CORBA IDL* untersucht werden.

Problem: Beziehungsspezifikation

Diskussion zur Spezifikation von Beziehungen: In der Komponenten – Verwaltung wird zur Spezifikation der Beziehungen eine *XML* – Datei 'Beziehungs – Deskriptor' benötigt, die in Anlehnung an der *CORBA IDL* zu gestalten ist. Da das *Schema* dieser Datei noch nicht existiert, wird in den folgenden Beschreibungen auf die Notation der *CORBA IDL* zurückgegriffen.

Bei der Aushandlung der Verträge zwischen kommunizierenden Implementierungen sind Abhängigkeiten auf Instanzenebene auszuwerten. Da diese mit der aktuell verwendeten Spezifikation nicht beschrieben werden können, sind Erweiterungen oder Alternativen zu suchen.

Auf den folgenden Seiten sollen die Grenzen der Beschreibungsmöglichkeiten der *CORBA IDL* aufgezeigt werden. Es werden folgende Probleme betrachtet.

1. Eine Implementierung benötigt von einer Komponenten – Spezifikation mehrere Instanzen. Gegebenenfalls sollen an diese Instanzen unterschiedliche *QoS* – Anforderungen gestellt werden.

2. Es wird eine Instanz durch mehrere Instanzen einer *Session* gemeinsam verwendet.

3. In diesem Spezialfall der Schleifenbildung wird direkt an die aufrufende Implementierung zurückverwiesen (***Callback***). In dem Fall müssen die Implementierungen ihre *QoS* – Eigenschaften gegenseitig erfüllen.

In der zweiten und dritten Problemsituation existieren Beziehungen zwischen den Instanzen, die zur Laufzeit durchgesetzt werden, indem zum Beispiel die Referenzen auf die Instanzen als Methoden – Parameter und / oder als Rückgabewert übergeben werden. Zum Zeitpunkt der Vertragsaushandlung müssen aber Sonderfälle beim Vergleich der *QoS* – Eigenschaften beachtet werden.

Im Folgenden sollen die Problemsituationen detailliert beschrieben und einige Lösungsansätze angesprochen werden. Dabei wird auf die *CORBA IDL* zurückgegriffen und notwendige Erweiterungen vorgeschlagen.

Das aktuelle Konzept besitzt folgende Beschreibungsmöglichkeiten: *CORBA IDL*

```
component KomponentenName {
  provides SchnittstellenName AusgangsName;
  uses SchnittstellenName EingangsName;
};
```

Diese Spezifikation beschreibt für eine Komponenten – Implementierung, über welche Schnittstellen sie kommuniziert. Während `provides` die Schnittstellen wiedergibt, welche diese Komponenten – Realisierung implementiert, beschreibt `uses` die Schnittstellen, die sie in ihrem funktionalen Bereich verwendet. Durch Gebrauch der Bezeichner für die Eingangs – und Ausgangs – Schnittstellen (`EingangsName` und `AusgangsName`) ist es möglich, ihnen direkt *QoS* – Eigenschaften zuzuordnen. Sie werden den Charakteristiken als Parameter zugewiesen. Während der Vertragsaushandlung werden die Bezeichner ausgewertet, um die geeigneten Implementierungen auszuwählen.

1. Mehrere Instanzen: Benötigt eine Komponenten – Implementierung mehrere Instanzen einer Komponenten – Spezifikation, kann sie dies zum Beispiel in der *IDL* durch Angabe mehrerer *uses* – Einträge für die gleiche Schnittstelle umsetzen. *Lösungsvorschlag: Mehrere Instanzen*

```
uses myInterface firstUsed;
uses myInterface secondUsed;
```

Da die *uses* – Einträge durch unterschiedliche Bezeichner repräsentiert werden, können ihnen so auch verschiedene *QoS* – Anforderungen zugewiesen werden. Um in dem Fall sicherzustellen, dass zur Laufzeit die richtigen Instanzen zu einer Schnittstelle verwendet werden, wird bei der Anforderung nicht der Schnittstellenname sondern der Bezeichner aus der *IDL* verwendet.

Der zuvor beschriebene Lösungsansatz kann nur eingesetzt werden, wenn die genaue Anzahl der Instanzen spezifiziert werden kann. Ist die Zahl von Laufzeitbedingungen abhängig, muss die benötigte Anzahl an Instanzen zur Laufzeit adaptiert werden. Die dynamische Erweiterung der Netze während der Ausführung der Anfrage ist aber ein prinzipielles Problem der Aufgabenstellung, und wie bereits beschrieben, im Rahmen dieser Arbeit nicht gelöst werden.

Problem: Gemeinsame Instanzen

2. Gemeinsam genutzte Instanzen: Das Problem soll an zwei kleinen Beispielen beschrieben werden.

In dem in Abbildung 3.33 dargestellten Fallbeispiel wird die Instanz `b` von der Instanz `a` aufgerufen. In `b` wird `x` erstellt. Als Antwort erhält `a` eine Referenz auf `x`. Im weiteren Verlauf von `a` ruft sie das Objekt `c` auf und übergibt `x`.

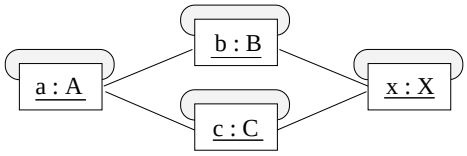


Abbildung 3.33: Beispiel für gemeinsam genutzte Instanzen: Die Instanzen `b` und `c` verwenden die gleiche Instanz `x`. Sie stellen auch beide *QoS*–Anforderungen an diese Instanz.

Da `c` keine eigene Instanz von `x` erstellt, sondern nur die übergebene verarbeitet, bestände die Möglichkeit, die Abhängigkeit zwischen `c` und `x` auf Spezifikationsebene nicht zu beachten, da diese Beziehung zur Laufzeit umgesetzt wird. Allerdings ist zu vermuten, dass `c` auch *QoS*–Anforderungen an `x` stellt. Diese Eigenschaften müssen zum Zeitpunkt der Reservierung der Instanz beachtet werden.

Bei Erstellung des Komponenten–Netzes müssen daher die Anforderungen von `b` und `c` erfüllt werden. Die Instanzen nehmen daraufhin dasselbe Objekt `x` in ihre abhängigen Netze auf.

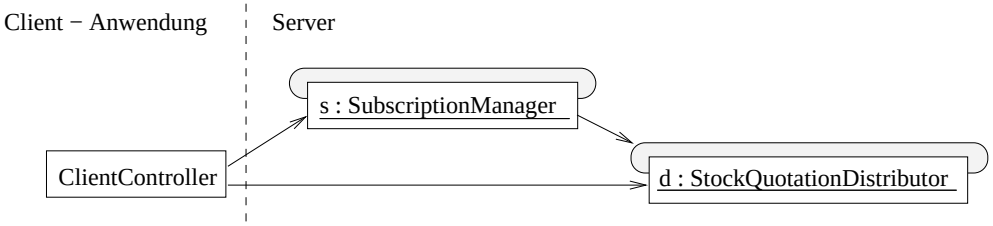


Abbildung 3.34: Börsentickerbeispiel für gemeinsam genutzte Instanzen: `ClientController` und `SubscriptionManager` verwenden die selbe `StockQuotationDistributor`–Instanz

Eine ähnliche Beziehung tritt beim Börsentickerbeispiel in folgender Situation auf (Abb. 3.34). Ein Client möchte die Kurse einer Aktie beziehen und stellt eine Anfrage beim `SubscriptionManager`. Dieser versucht daraufhin, den Client bei einer Implementierung der `DistributorInterface`–Schnittstelle (zum Beispiel `StockQuotationDistributor`) anzumelden. Dabei stellt der `SubscriptionManager` die nichtfunktionale Anforderung, dass die Rückmeldung, ob die Auslieferung der Kurse durchgeführt werden kann oder nicht, bis zu einem bestimmten Zeit-

punkt erfolgt. Nachdem der Client beim `StockQuotationDistributor` angemeldet wurde, empfängt er die Aktienkurse. Der Client stellt ebenfalls nichtfunktionale Anforderungen bezüglich der Auslieferung der Kurse an die Implementierung der `DistributorInterface` – Schnittstelle (z. B. maximale Fehlerrate).

Eine weitere Besonderheit dieses Beispiels ist, dass der Client Anforderungen an den `StockQuotationDistributor` stellt, diese aber auf eine vom Client angebotene Methode zugreift, um ihn über die aktuellen Kurse zu informieren.

Zur Spezifikation der zuvor beschriebenen Beziehungen muss zusammengefasst werden, welche Komponenten – Implementierungen gemeinsam ein und dieselbe Instanz verwenden, um auf diese Weise alle *QoS* – Anforderungen, die sie an diese Instanz stellen, zu überprüfen.

Lösungsvorschlag

Zur Kennzeichnung der gemeinsam genutzten Instanz, wird das Schlüsselwort `sharedInstance` eingeführt.

```
sharedInstance ScbnittstellenName Bezeichner;
```

Im Beispiel des Börsentickers ergibt sich daraus folgende Beschreibung:

```
sharedInstance DistributorInterface stockQuotDistributor;

component SubscriptionManager {
  provides SubscribeUnsubscribe subscriptionInterface;
  uses stockQuotDistributor distributor;
};

component ClientController {
  uses SubscribeUnsubscribe subscriptionManager;
  uses stockQuotDistributor stockDistributor;
};
```

Beim Einlesen dieser Beschreibungen werden durch die Implementierungs – Verwaltung die gemeinsam genutzten Instanzen sowie die Implementierungen, von denen sie verwendet werden, registriert. Wird dann die Instanz während der Vertragsaushandlung von einer Implementierung angefordert, überprüft der Vertragsmanager als Erstes, ob die zugehörige Instanz schon erstellt wurde. Dies wird ermöglicht, indem die gemeinsam genutzten Instanzen im Container über ihren Bezeichner verwaltet werden. Ist die Instanz noch nicht erstellt worden, müssen bei der Vertragsaushandlung die *QoS* – Eigenschaften aller aufrufenden Implementierungen (`SubscriptionManager`, `ClientController`) überprüft werden.

Eine Erweiterung der Problemstellung des Beispiels in Abbildung 3.33, bei dem die Instanz `c` zusätzlich zu der Instanz `x` der Implementierung `X` eine eigene benötigt, kann durch Kombination der Lösungsvorschläge aus 1 und 2 bewältigt werden.

Diese Spezifikation stellt auch eine Lösung des in der Abbildung 3.32 (S. 57) beschriebenen Problems dar. Soll ein bereits existierendes Objekt verwendet werden, wird auf ein `sharedInstance` – Eintrag verwiesen. Andernfalls wird immer eine eigene Instanz benötigt.

Problem: Callback

3. Callback: Wenn von einer Implementierung auf die aufrufende Instanz zurückverwiesen werden soll, stellt das einen weiteren Spezialfall dar, der durch den Beziehungs – Deskriptor beschrieben werden sollte. Diese Abhängigkeiten sind in einfacher Form in Abbildung 3.35 dargestellt.

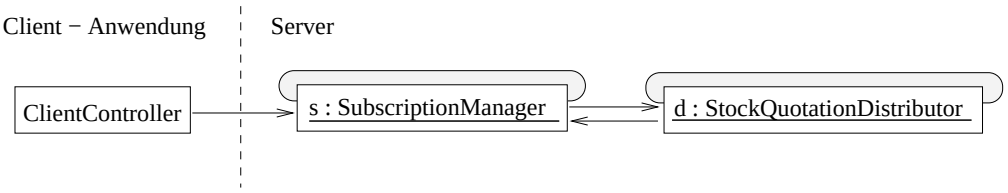


Abbildung 3.35: Callback im Börsentickerbeispiel

Der `SubscriptionManager` ist in dem Fall ein *Stateful Session Bean* – Objekt, das alle Daten über den Client verwaltet. Wenn es den Client bei einem `StockQuotationDistributor` anmeldet, greift dieser auf den `SubscriptionManager` zu, um zum Beispiel zu ermitteln, wie viele Abonnements der Client insgesamt bezieht.

Lösungsvorschlag

Um diese Beziehung zu spezifizieren, wurde das Schlüsselwort `CALLBACK` eingeführt.

```
component SubscriptionManager {
  provides SubscribeUnsubscribe subscriptionInterface;
  uses DistributorInterface distributor;
};
component StockQuotationDistributor {
  provides DistributorInterface stockQuotDistributor;
  uses SubscribeUnsubscribe subscriptionManager CALLBACK;
};
```

`CALLBACK` beschreibt, dass die Implementierung `StockQuotationDistributor` bei der Reservierung der abhängigen Komponenten – Spezifikation auf die Instanz verweisen muss, von der sie aufgerufen wurde (`SubscriptionManager`). Es muss in dem Fall überprüft werden, ob der `SubscriptionManager` den Anforderungen, die der `StockQuotationDistributor` an die `SubscribeUnsubscribe` – Schnittstelle stellt, genügt. Werden die Eigenschaften nicht erfüllt, kann keine Instanz vom `StockQuotationDistributor` reserviert werden. In dem Fall wird nach einer alternativen Implementierung der `DistributorInterface` – Schnittstelle gesucht. Steht keine zur Verfügung, ist die Reservierung der `SubscriptionManager` – Implementierung mit dem aktuellen Profil nicht möglich.

Um das Reservieren und Zurücksetzen zu reduzieren, sollte bereits bei der Vertragsaushandlung für den `SubscriptionManager` überprüft werden, ob er die `CALLBACK` – Bedingungen erfüllt. Daher muss diese Beziehung auch bei der aufrufenden Implementierung gekennzeichnet werden. Das dafür eingeführte Schlüsselwort `CALL`, wird im Beispiel folgendermaßen angewandt.

```
component SubscriptionManager {
  provides SubscribeUnsubscribe subscriptionInterface;
  uses DistributorInterface distributor CALL;
};
```

Dieses Beschreibung kann auch in Verbindung mit dem Problemfall ‘Gemeinsam genutzte Instanzen’ (`sharedInstance`, S. 58 ff.) verwendet werden.

Mit Hilfe der vorgestellten Beschreibungsmöglichkeiten können einige Probleme bei der Vertragsaushandlung gelöst werden. Konnte durch den Vertragsmanager das Netze aus kommunizierenden Instanzen vollständig erstellt werden, kann der Client auf die reservierten Objekte zugreifen. Zu diesem Zeitpunkt muss der Vertragsmanager wieder aktiv werden, um die Einhaltung der abgeschlossenen Verträge durchzusetzen.

3.5.2 Durchsetzung der Verträge

Vertragsabschluss & -durchsetzung

Nachdem alle Verträge für eine Client – Anfrage ausgehandelt wurden, wird dem Client die Referenz auf die Wuzel des reservierten Netzes übergeben. Woraufhin dieser die Geschäftsmethoden der Instanz ausführen kann. Der Vertragsmanager übernimmt die Aufgabe die Verträge durchzusetzen, sobald auf die reservierten Instanzen zugegriffen wird. Diese Überwachung unterteilt sich in die Funktionsbereiche *Kommunikations – Proxy* und *Ressourcen – Proxy*.

Der Kommunikations – Proxy hat die Aufgabe, die reservierten Instanzen zuzuweisen, während der Ressourcen – Proxy die Verwendung der reservierten Betriebsmittel durchsetzt. Die Funktionsweise dieser Bestandteile des Vertragsmanagers wird nachfolgend vorgestellt.

3.5.2.1 Kommunikations – Proxy

Kommunikations – Proxy

Während im Client bei der Anforderung einer Instanz zu einer Spezifikation die Reservierung des Netzes ausgelöst wird, werden bei diesem Aufruf im Server, die bereits reservierten Instanzen zugewiesen. Diese Aufgabe übernimmt der Kommunikations – Proxy.

Funktionsweise

Da die Reservierung über den in der *CORBA IDL* verwendeten Bezeichner erfolgt, muss er bei der Instanzenanforderung übergeben werden. Daraufhin wird in der für die aktuell ausgeführte Instanz reservierten Menge aus abhängigen Instanzen nach dem Eintrag zu diesem Bezeichner gesucht. Nachdem das Objekt an die aufrufende Instanz übergeben wurde, kann sie auf dessen Geschäftsmethoden zugreifen.

In der aktuell zu entwerfenden Komponenten – Verwaltung wird davon ausgegangen, dass sich die Server – Anwendung auf einem Knoten befindet. Die Reservierung des Komponenten – Netzes erfolgt daher nur lokal. Um dieses Funktionalität zu erweitern, stehen zwei Alternativen zur Auswahl.

In beiden Fällen muss zu jeder Komponente spezifiziert werden, auf welchem Server sie sich befindet.

1. Während der Reservierung des Netzes werden auch die Komponenten – Implementierungen eingebunden, die sich auf einem anderen Server befinden.

2. Nach Anfrage eines Clients werden nur die Objekte auf dem einen Server reserviert. Wird die Anfrage ausgeführt und auf Komponenten in einem anderen Knoten zugegriffen, wird das Netz erweitert. In dem Fall fungiert die Komponente im Server als Client für die andere Server – Anwendung.

Der erste Lösungsvorschlag hat den Nachteil, dass der Aufwand zur Reservierung bei verteilter Server – Anwendungen zu groß werden kann. Bei der zweiten Variante muss die Verarbeitung unterbrochen werden, um das Netz zu erweitern, was wiederum mit einer eventuell geforderten Antwortzeit kollidiert. Es sind daher ausführliche Untersuchungen zur Lösung der Probleme erforderlich, die nicht in dieser Arbeit durchgeführt werden können.

3.5.2.2 Ressourcen – Proxy

Ressourcen – Proxy

Nach Reservierung der Ressourcen müssen die Verträge mit der Ressourcen – Verwaltung durchgesetzt werden. Das ist die Aufgabe des *Ressourcen – Proxy*. Er muss garantieren, dass nur die reservierten Ressourcen verwendet werden.

Wird auf Betriebsmittel zugegriffen, aktiviert sich der Ressourcen – Proxy und leitete die Anfrage direkt an die Verwaltung dieser reservierten Ressourcen weiter. Die Einhaltung der Verträge (z. B. Größe des reservierten Speicherplatzes) überwacht die Ressourcen – Verwaltung.

Vertragsdurchsetzung

Im Rahmen dieser Arbeit wird die Reservierung und der Zugriff auf die Ressourcen ***CPU***, ***Speicher*** und ***Netzwerk*** betrachtet. Bei Verwendung der Ressource *CPU* sind keine Maßnahmen zur Durchsetzung der Verträge erforderlich. Konnte vom Vertragsmanager genügend Prozessorzeit reserviert werden, steht diese bei Bedarf zur Verfügung. Beim Zugriff auf die Ressourcen *Speicher* und *Netzwerk* treten einige Probleme auf. Die in dem Fall notwendigen Maßnahmen werden nachfolgend beschrieben.

Speicher und Netzwerk

Zugriff auf Ressourcen

Der Zugriff auf die reservierten Betriebsmittel erfolgt über die für die Ressource zuständigen Ressourcen – Manager. Beim Zugriff muss der bei der Vertragsaushandlung erstellte *Handle* für die Ressource übergeben werden. Desweiteren ist die benötigte Menge zu spezifizieren.

Speicher

Speicher: Um die Verträge mit dem Speicher – Ressourcen – Manager durchzusetzen, muss das Erstellen und Löschen von Objekten überwacht werden. Da die Komponenten – Verwaltung und die in ihr verarbeiteten Anwendungen in *Java* implementiert sind, ergeben sich Probleme bei der Überwachung des Zugriffs auf diese Ressourcen, da ein direkter Zugriff nicht zulässig ist.

'... the Java language has eliminated the programmer's ability to get behind the scenes and either forge or otherwise manufacture pointers to memory' [12]

Es muss demzufolge die Möglichkeit geschaffen werden, diese Beschränkung durch die *Java Virtual Machine* (*JVM*,[20]) zu umgehen. Bevor dazu Vorschläge unterbreitet werden, sollen die von der *JVM* umgesetzten Funktionen vorgestellt und die zu verändernden Bereiche beschrieben werden. Als Quellen dienten [12] und [23].

Objekterzeugung

In *Java* sind Objekte zu erzeugen, sobald eine der folgenden Funktionen ausgeführt wird:

- Instanzerzeugung (`new`),
- `newInstance` – Methode der Klasse `java.lang.Class` ausgeführt,
- Laden einer Klasse, die ein `String`-Literal enthält — Umwandlung in ein String-Objekt oder
- eventuell für Zwischenergebnisse bei der Auswertung des `String` – `'+'` – Operators in `String` – Objekten.

Die bei der Objekterzeugung auszuführenden Schritte sind:

1. ***Speicherallokation***,
2. *Default – Initialisierung* und
3. Konstruktion des Objektes.

Die Komponenten – Verwaltung hat in dem Fall die *Speicherallokation* vorzunehmen, und muss die bereits reservierten Speicherplätze verwenden. Für die daran anschließenden Schritte kann die ursprüngliche Funktionalität der *JVM* übernommen werden.

Speicherallokation

Bei der *Allokation* muss so viel Speichplatz reserviert werden, dass er groß genug ist für:

- alle Instanzvariablen der Klasse des Objektes und
- alle Instanzvariablen aller Oberklassen des Objektes.

Nachdem die Ressourcen reserviert werden konnten, werden die Instanzvariablen (auch die der Oberklassen) des neuen Objektes mit ihren *Default* – Werten initialisiert.

Objektzerstörung

Das Löschen der Objekte kann in *Java* ebenfalls nicht durch Anwendungsprogramme beeinflusst werden. Diese Funktion übernimmt ein ***Garbage Collector***. Er kennt alle existierenden Objekte und überwacht, auf welche nicht mehr verwiesen wird. Der *Garbage Collector* läuft als *Low – priority – thread* im Hintergrund. Deshalb wird er zum Beispiel nur aktiv, wenn auf Benutzereingaben gewartet wird. Die Änderung seiner Priorität erfolgt nur dann, wenn dem *Interpreter* kein *Speicher* mehr zur Verfügung steht. Obwohl der *Garbage Collector* unterstützt werden kann, indem die Referenzen auf `null` gesetzt werden, hat die Anwendung keinen Einfluss darauf, wann die Speicherplätze freigegeben werden. Da diese Bedingungen für die Komponenten – Verwaltung unzureichend sind, muss die Arbeitsweise des *Garbage Collectors* verändert oder umgangen werden.

Zielsetzung

In der umzusetzenden Komponenten – Verwaltung sind die reservierten Speicherplätze zu verwenden. Wird die Reservierung des Objektes aufgehoben, ist der Speicherplatz sofort freizugeben. Diese Funktionalität kann in *Java* nicht umgesetzt werden. Um diese Beschränkung zu umgehen, kann zum Beispiel der *Java Native Interface* (*JNI*, [26][19][5]) verwendet werden. Er bietet unter anderem die Möglichkeit *C* – oder *C++* – Quellcode in *Java* – Klassen zu integrieren. Diese Programmiersprachen besitzen Methoden, mit denen direkt auf Speicherplätze zugegriffen werden kann (`malloc` und `free` in *C* sowie `new` und `delete` in *C++*).

Um die Speicherzugriffe zu integrieren, müssen die oberhalb beschriebenen Funktionen, bei denen ein Zugriff auf den Speicher erfolgt, überschrieben werden. Desweiteren muss ein ***Destructor*** eingefügt werden, der eine sofortige Freigabe der Speicherplätze durchführt.

Netzwerk

Netzwerk: Bei der Arbeit mit der Ressource *Netzwerk* müssen ebenfalls Funktionen zum Zugriff und zur Freigabe umgesetzt werden.

Zur Aufgabe der *Netzwerk* – Ressource gehört einerseits die Überwachung, ob innerhalb einer Komponente auf das *Netzwerk* zugegriffen wird. Dieser Zugriff kann zum Beispiel mit *JDBC* oder *URL* erfolgen. Andererseits wird die Ressource *Netzwerk* verwendet, sobald mit einer im Komponenten – Netz reservierte Instanz kommuniziert wird.

Verwendung weiterer Ressourcen beim Zugriff auf Ressourcen: Werden bei der Verarbeitung des Zugriffs auf eine Ressource noch andere Ressourcen benötigt, müssen diese ebenfalls reserviert werden.

Diese Abhängigkeit tritt zum Beispiel auf, wenn für den Zugriff auf das *Netzwerk Speicherplatz* und *CPU* benötigt werden. Diese Anforderungen werden bereits bei der Reservierung der Ressource *Netzwerk* berücksichtigt. Die Netzwerk – Resourcen – Verwaltung reserviert bei jeder Reservierung zusätzlich die Ressourcen *Speicherplatz* und *CPU*.

Die Reservierung und Überwachung der Einhaltung dieser Verträge geschieht innerhalb der Ressourcen – Verwaltung. In der Komponenten – Verwaltung müssen diese Abhängigkeiten nicht berücksichtigt werden.

3.5.3 Freigabe der reservierten Ressourcen

Reservierung freigeben

Nach Beendigung seiner Anfragen gibt der Client die reservierten Komponenten wieder frei. Dazu kann er einerseits einzelne Reservierungen entlassen, andererseits werden *Sessions* über Sitzungen verwaltet, die mit einem Mal beendet werden können. In dem Moment werden alle in der *Session* reservierten Objekte freigegeben.

Caching

Hat ein Client eine Reservierung aufgehoben, ist im Sever das Komponenten – Netz aufzulösen und die Instanzen freizugeben. Die entlassenen Instanzen werden nicht gelöscht, sondern stehen für weitere Anfragen zur Verfügung. Wie im Abschnitt „Caching von Netzen“ (S. 56) beschrieben wird, besteht ebenfalls die Möglichkeit Netze zwischenzuspeichern.

Löschen

Komponenten – Netze und Instanzen werden endgültig gelöscht, wenn eine der folgenden Situationen eintritt:

- die Komponenten – Verwaltung beendet ihre Arbeit,
- die Komponenten – Implementierung wird gelöscht oder
- die Instanz erreicht ihren *Timeout*.

In jedem Fall wird die Instanz zerstört und die reservierten Ressourcen freigegeben. Die dabei zu beachtenden Bedingungen wurden bereits bei der Diskussion zur Verwaltung von Komponenten – Implementierungen beschrieben (S. 31 ff.).

3.6 Zusammenfassung

In diesem Kapitel wurde der Entwurf der Komponenten – Verwaltung und seiner Funktionseinheiten vorgestellt.

Für die Verwaltung der *QoS* – Eigenschaften ist das *QoS* – Repository zuständig. Die Informationen zu den Komponenten der Anwendungen werden in der Implementierungs – Verwaltung verarbeitet. Desweiteren ist sie für die Verwaltung der zur Laufzeit erstellten Instanzen zuständig. Die aufgrund der Ressourcenanforderungen erforderliche Aushandlung und Durchsetzung von Verträgen zwischen kommunizierenden Komponenten und mit der Ressourcen – Verwaltung ist die Aufgabe des Vertragsmanagers. Seine Teilbereich Kommunikations – Proxy und Ressourcen – Proxy sind für die Durchsetzung der Verträge zuständig.

Bei der Vertragsaushandlung werden die Beziehungen zwischen Implementierungen ausgewertet und auf deren Grundlage Netze aus kommunizierenden Instanzen erstellt. Zur fehlerfreien Reservierung der Netze müssen Beziehungen zwischen Instanzen von Komponenten – Implementierungen spezifiziert werden

können. Um den Aufwand für die Aushandlung der Verträge zu reduzieren werden Objekte und Netze zwischengespeichert.

Die entwickelten Konzepte zum Entwurf der Komponenten – Verwaltung bieten eine Grundlage für die im folgenden Kapitel vorgestellten Implementierungsdetails. Bei der Umsetzung sind die angesprochenen eventuell auftretenden Probleme zu berücksichtigen und in der folgenden Implementierung zu vermeiden.

Kapitel 4

Implementierung der Komponenten – Verwaltung

Im vorangegangenen Kapitel 3 wurden die Funktionen der Komponenten – Verwaltung analysiert und wichtige Entwurfsentscheidungen getroffen. An dieser Stelle wird nun der Entwurf verfeinert und Vorschläge zur Implementierung unterbreitet. Die prototypische Implementierung einiger Funktionsbereiche wird am Ende dieser Kapitels vorgestellt.

Bei der Realisierung der Komponenten – Verwaltung soll beachtet werden, dass die in ihr auszuführenden Anwendungen möglichst unabhängig von der Verarbeitung der *QoS* – Eigenschaften und den damit verbundenen Funktionen umgesetzt werden können. Zur Erfüllung dieser Anforderungen stehen die Konzepte der ***Interceptoren*** [25] oder der *Aspektorientierte Programmierung* zur Verfügung. Deren Eignung soll zu Beginn diskutiert werden.

4.1 Untersuchung der Einsatzmöglichkeiten von Interceptoren und AOP

Motivation

Während ein Client auf Objekte im Server zugreift, sind durch die Komponenten – Verwaltung zusätzliche Verarbeitungsschritte auszuführen, wie zum Beispiel:

- Reservierung von Instanzen und
- Zuweisung der reservierter Instanzen.

Zukünftig könnte die Komponenten – Verwaltung eventuell um die Funktionen:

- *Tracing* und *Logging*,
- *Kodierung* und *Dekodierung* und
- *Zugangskontrolle* mit *Authentication* und *Authorization*

Anforderungen

erweitert werden. Diese Funktionalität ist nicht Bestandteil der Geschäftsmethoden, muss aber beim Zugriff auf diese verarbeitet werden.

Eine wichtige Anforderung an die Komponenten – Verwaltung ist, die Erweiterung für den Anwendungsentwickler transparent umzusetzen. Die Anwendung soll unabhängig vom Aufruf dieser zusätzlichen Funktionen gestaltet werden können.

Zur Integration dieser zusätzlichen Funktionalität können *Interceptoren* und *Aspektorientierte Programmierung* (*AOP*) verwendet werden. Es werden die Eigenschaften dieser Konzepte vorgestellt und ihre Eignung zur Gestaltung der Komponenten – Verwaltung untersucht.

4.1.1 Interceptoren

Interceptor

Das *Interceptor* – Muster beschreibt eine Möglichkeit der Integration der zusätzlicher Funktionalität in einem ***Framework*** über eine einheitliche Schnittstelle. Es folgt der Vorstellung seiner Arbeitsweise, dessen Beschreibungen [25] und [29] entnommen wurden.

Integration

Funktionsweise: Für jede relevante Ereignisgruppe wird vom *Framework* ein ***Dispatcher*** angeboten. Im aktuellen Kontext sind zum Beispiel folgende Ereignisse von Bedeutung:

- für aufrufende Komponente oder den Client:
  - Anforderung oder Entlassung einer Referenz und Empfang der Antwort
  - Aufruf einer Geschäftsmethode und Empfang der Antwort
- für aufgerufene Komponente:
  - Empfang einer Anforderung oder Entlassung einer Referenz und Senden der Antwort
  - Empfang des Aufrufs einer Geschäftsmethode und Senden der Antwort

Funktionsweise

Tritt eines der beschriebenen Ereignisse ein, informiert das *Framework* den *Dispatcher*. Dieser leitet die Information an die bei ihm registrierten *Interceptoren* weiter, welche die zusätzliche Funktionalität implementieren. Um von einem *Dispatcher* über das Eintreten eines Ereignis informiert zu werden, muss sich ein *Interceptor* beim *Dispatcher* anmelden.

Die Beziehungen zwischen *Framework*, *Dispatcher* und *Interceptor* sind in der Abbildung 4.1 dargestellt. In den *Interceptoren* werden Methoden definiert, die die zusätzliche Funktionalität umsetzen. Für jede der bereits beschriebenen Framework – Erweiterungen können eigene *Interceptoren* definiert werden. Sie alle müssen die `Interceptor` Schnittstelle implementieren.

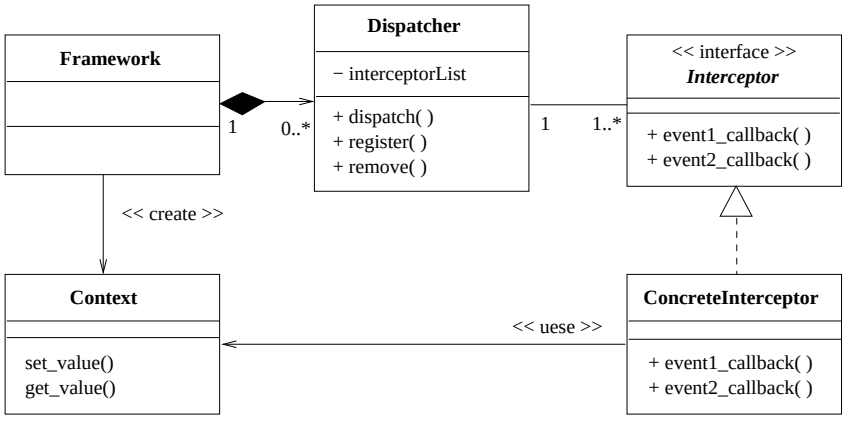


Abbildung 4.1: Umsetzung des *Interceptor* – Musters

Manipulation der Daten

In den *Interceptoren* kann die Anfrage bearbeitet werden, wie zum Beispiel die Verschlüsselung der Parameter. Im Server werden diese vor Ausführung der Methode wieder entschlüsselt. Das gleiche geschieht mit der Antwort.

Desweiteren besteht die Möglichkeit, dass die *Interceptoren* der Anfrage Informationen hinzufügen, so zum Beispiel eine Identifikation des Clients. Um den *Interceptoren* diese Verarbeitung zu ermöglichen, übergibt ihnen das *Framework* ein *Context* – Objekt. In diesem werden die erforderlichen Informationen zur Verfügung gestellt. Darüberhinaus werden Methoden zur Manipulation der Anfrage angeboten.

`Context` – Objekt

Das *Context* – Objekt kann den *Interceptoren* zu zwei Zeitpunkten übergeben werden – bei der Registrierung im *Dispatcher* oder bei jedem Eintreffen des beschriebenen Ereignisses. Die erste Variante ist weniger aufwendig, ermöglicht aber nur die Verarbeitung allgemeiner Informationen. Bei der zweiten Alternative stehen Informationen zum aktuellen Methodenaufruf zur Verfügung. Beim Entwurf muss außerdem entschieden werden, ob in einem `Context` – Objekt alle möglichen Informationen verarbeitet werden, oder ob das Framework mehrere ereignisspezifische anbietet.

Interceptoren im Server

Nach Manipulation des Anfragekontextes wird diese vom Framework an den Server weitergeleitet. Dort erfolgt ebenfalls die Ausführung der registrierten *Interceptoren*, bevor die eigentliche Methode aufgerufen wird.

Da für ein bestimmtes Ereignis mehrere *Interceptoren* registriert werden können, muss eine Reihenfolge der Abarbeitung festgelegt werden. Eine Möglichkeit bietet die Orientierung an der Reihenfolge der Registrierung. Dafür stehen zum Beispiel die Verfahren *FIFO* (*first in first out*) und *LIFO* (*last in first out*) zur Verfügung. Da für alle Maßnahmen in der Client – Umgebung entgegengesetzte Aktionen in der Server – Umgebung erforderlich sind (z. B. Ver – und Entschlüsselung), die in umgekehrter Reihenfolge abgearbeitet werden, steht einem *FIFO* in der Client – Umgebung ein *LIFO* in der Server – Umgebung gegenüber.

Proxy

Eine andere Option zur Verarbeitung der *Interceptoren* auf Server – Seite besteht in der Verwendung des *Proxy* – Musters [11]. In dem Fall wird von dem *Framework* eine Klasse instanziiert, die der gleichen Schnittstelle wie die Komponente genügt. In den Methoden werden als Erstes die *Interceptoren* ausgeführt und anschließend die assoziierte Instanz der Komponente aufgerufen. Der Client kommuniziert nur über das *Proxy* mit der Komponente.

Ein Problem stellt die Eintrittsüberwachung eines Ereignisses dar. Es besteht die Möglichkeit einen *Observer* umzusetzen. Als Alternative könnte *Aspektorientierte Programmierung* verwendet werden, auf die nun eingegangen wird.

4.1.2 Aspektorientierte Programmierung

AOP

Mit der *Aspektorientierten Programmierung* werden die selben Ziele wie mit *Interceptoren* verfolgt.

Vorteil

Ein großer Vorteil von *AOP* liegt darin, dass Anwendungen über einen Aspekt in eine existierende Laufzeitumgebung integriert werden können, ohne dass vom Anwendungsentwickler zum Beispiel umgebungsspezifische Schnittstellen zu implementieren sind. Um von einem Aspekt um die gesamte Funktionalität der Komponenten – Verwaltung erweitert zu werden, müssen die Klassen zum Beispiel nur bestimmte Namenskonventionenen (eventuell vorgegebene Endungen) einhalten. Da diese Anpassungen den Verlauf der Anwendung nicht beeinflussen, sind sie unproblematischer, falls die Komponenten zum Beispiel in einem anderen Aufgabenkontext verwendet werden sollen.

Integration

Mit Hilfe der *Aspekte* kann außerdem die Ausführung von Methoden überwacht und zusätzliche Funktionalität eingefügt werden. Es gibt die Möglichkeit an folgenden Punkten in den Programmverlauf einzugreifen:

- Aufruf einer Methode,
- Ausführung einer Methode,
- Beendigung der Ausführung einer Methode oder
- Rückkehr von einer Methode (erfolgreiche Rückkehr oder nach Auslösung einer *Exception*).

Diese Beispiele beziehen sich nur auf Methodenaufrufe. Es können aber auch Zugriffe auf Konstruktoren und Attribute überwacht werden.

Nachdem das Eintreten eines Ereignisses registriert wurde, kann in den *Advices* die zusätzliche Funktionalität ausgeführt werden. Diese entspricht der Implementierung in den *Interceptoren*. Da bei dieser Lösung der *Aspekt* immer erweitert werden muss, um die ergänzende Funktionalität einzufügen, bietet es sich an, ebenfalls eine einheitliche Schnittstelle zur Verfügung zu stellen, über die die Funktionen integriert werden können. Bei der Gestaltung der Schnittstelle kann auf das *Interceptor* – Muster zurückgegriffen werden.

4.1.3 Gestaltung der Laufzeitumgebung

Gestaltung Container

Aufgrund übersichtlicher Strukturierung und besseren Dokumentationsmöglichkeiten von Aufbau und Funktionsweise der Laufzeitumgebung wurde die Komponenten – Verwaltung in *Java* ohne Verwendung von *AspectJ* umgesetzt. Der modulare Aufbau wird im Anschluss an die Vorstellung der Implementierungsdetails im Abschnitt 4.5 beschrieben. An dieser Stelle werden zusätzlich Vorschläge unterbreitet, wie die Trennung der Funktionsbereiche in der Komponenten – Verwaltung durch *Aspekte* unterstützt werden kann.

Containerintegration

In der aktuellen Version der Komponenten – Verwaltung besteht die Aufgabe der *Aspekte* nur in der transparenten Integration der Funktionalität in der Laufzeitumgebung. Der Aspekt `Integration` setzt diese Anforderung um, indem er alle *Session Bean* – Komponenten von den dafür vorgesehenen Klassen und Schnittstellen der Komponenten – Verwaltung ableitet (Abschnitt 4.2.4). Die Integration in der Klassenstruktur der Komponenten – Verwaltung kann aufgrund der auf Seite 89 beschriebenen Namenskonventionen umgesetzt werden.

Containererweiterung

Soll die Funktionalität der Komponenten – Verwaltung zukünftig erweitert werden, besteht die Möglichkeit, den entwickelten *Container* direkt zu modifizieren. Wird dieser Eingriff in die Funktionsbereiche etwa aus Sicherheitsgründen ausgeschlossen, können Schnittstellen ähnlich dem *Interceptor* – Muster angeboten werden, um die Funktionalität zu integrieren.

Kombination AOP und Interceptoren

Es werden daher *Aspekte* erstellt, die die Ausführung bestimmter Methoden überwachen und diese Ereignisse an die *Dispatcher* melden. Ein Beispiel für einen solchen *Aspekt* `AccessController`, der in der Client – Laufzeitumgebung verwendet werden könnte, ist in der Abbildung 4.2 dargestellt.

Eine Auflistung der bei Verwendung dieser Funktionalität angebotenen *Dispatcher* ist dem Anhang B zu entnehmen. Bei der Spezifikation dieser ist zu beachten, dass alle Alternativen von Ereignissen vorhergesehen werden müssen. Sollten nach Erstellung der Komponenten – Verwaltung weitere benötigt werden, müssen die *Aspekte* ergänzt und zusätzliche *Dispatcher* geschaffen werden.

Es besteht ebenfalls die Möglichkeit ein *Context* – Objekt wie im *Interceptor* – Muster zu verwenden. Dieses wird in den *Aspekten* erstellt und ausgewertet – und an die *Interceptoren* übergeben.

Nachdem die Einsatzbereiche für *Aspekte* und *Interceptoren* geklärt wurden, folgt nun die Vorstellung der Implementierungsdetails zur Umsetzung der Komponenten – Verwaltung.

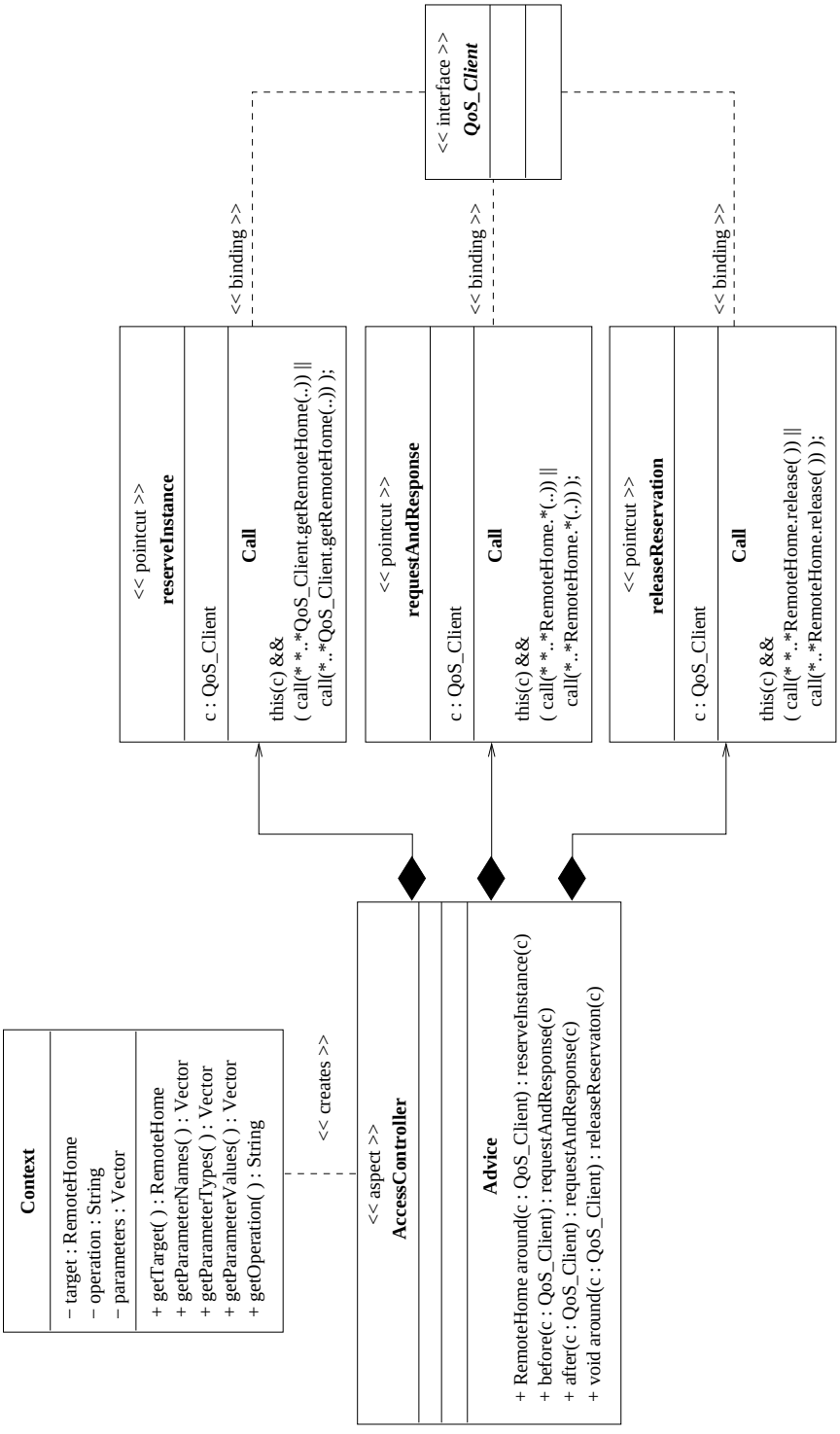


Abbildung 4.2: Aufbau eines Aspektes zur Zugriffsüberwachung auf die Komponenten – Verwaltung (nach [18])

4.2 Verfeinerter Entwurf der serverseitigen Komponenten – Verwaltung

In diesem Abschnitt wird die Arbeitsweise der Komponenten – Verwaltung auf dem Server detailliert beschrieben. Es werden Vorschläge zur Implementierung der Funktionalität unterbreitet.

4.2.1 Start der Komponenten – Verwaltung und Einlesen der Server – Anwendung

Start des Servers

Zum Start der Komponenten – Verwaltung wird die `main` – Methode der Klasse `ContainerImpl`, welche der `Container` – Schnittstelle genügt, ausgeführt. Da von der Client – Umgebung auf dieses Objekt zugegriffen wird, ist es beim ***Naming – Server*** anzumelden. Zum Startzeitpunkt müssen folgende Dateien zur Verfügung stehen:

Informationen für den Start

- eine *XML* – Datei, die in Anlehnung an die *CORBA IDL* für alle Komponenten – Implementierungen die angebotene (eine) und geforderten (mehrere) Schnittstellen spezifiziert (Beziehungs – Deskriptor),
- eine *XML* – Datei, die die *QoS* – Eigenschaften der Server – Anwendung spezifiziert und auf *CQML*$^+$ *basiert* (*QoS* – Deskriptor) und
- die Datei *container.conf* — sie folgende Informationen enthält:
  - Name, unter dem das `ContainerImpl` – Objekt beim *Naming – Server* anzumelden ist (Property `CONTAINER`),
  - Name, unter dem die Resourcen – Verwaltung beim *Naming – Server* angemeldet ist (Property `RESOURCE_MANAGER`),
  - Name und Verzeichnis der Datei, die die Beziehungen spezifiziert (Property `RELATION_XML`) und
  - Name und Verzeichnis der Datei, die die *QoS* – Eigenschaften spezifiziert (Property `CQML_XML`).

Die *container.conf* – Datei ist im Verzeichnis *qos/server/config/* in der Server – Umgebung abzulegen. Beim Start wird das `ContainerImpl` – Objekt als Erstes beim *Naming – Server* angemeldet. Im Anschluss daran folgt das Einlesen der Server – Anwendung. Dabei müssen die zu verwaltenden Schnittstellen, Implementierungen und *QoS* – Eigenschaften eingelesen werden. Im Container stehen dafür die zwei Parser `QoS_AppParser` und `QoS_Parser` zur Verfügung. Diesen werden bei der Initialisierung die Verzeichnisse und Namen der *XML* – Dateien übergeben.

Parser

Die beim Einlesen auszuführenden Arbeitsschritte werden nachfolgend beschrieben. Daran anschließend werden die durch die Komponenten – Verwaltung auszuführenden Funktionen erläutert.

Einlesen der Spezifikationen und Implementierungen

Verwaltungsklassen

Die Verwaltung der Schnittstellen und Implementierungen ist in der Abbildung 4.3 dargestellt.

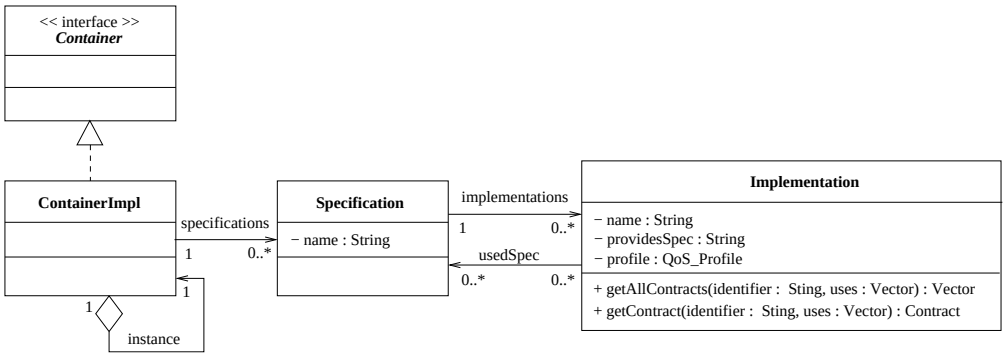


Abbildung 4.3: Verwaltung der Informationen zu den Komponenten – Implementierungen

QoS_AppParser

Die Informationen zu den Beziehungen werden vom `QoS_AppParser` – Parser eingelesen. Dazu ist seine `readApp` – Methode auszuführen. Es werden für alle Implementierungen folgende Arbeitsschritte ausgeführt:

```
begin Implementierung einlesen
  implName = EINLESEN(Name Implementierung);
  newImpl = new Implementation(implName);
  provInterfaceName = EINLESEN(Name angebotene Schnittstelle);
  Specification provSpec;
  provSpec = ContainerImpl.getSpec(provInterfaceName);
  if (ContainerImpl.getSpec(provInterfaceName) not exists) then
    provSpec = ContainerImpl.newSpec(provInterfaceName);
  end if
  provSpec.addImplementation(newImpl);
  while (Implementierung weitere Schnittstellen verwendet)
  do
    usesInterfaceName = EINLESEN(Name angebotene Schnittstelle);
    usesSpec = ContainerImpl.getSpec(usesInterfaceName);
    if (ContainerImpl.getSpec(usesInterfaceName) not exists) then
      usesSpec = ContainerImpl.newSpec(usesInterfaceName);
    end if
    newImpl.addUsesSpecification(usesSpec);
  end do
end
```

Profil einbinden

Nachdem eine Implementierung eingelesen wurde, wird in der `QoS_Parser` – Klasse die Methode `parseProfileForComponent` ausgeführt. Sie liest die *QoS* – Eigenschaften für die Implementierung ein und gibt den Verweis auf das `QoS_Profile` – Objekt zurück, welches daraufhin an die Klasse zur Beschreibung der Implementierung (`Implementation`) gebunden wird. Die Funktionsweise der `QoS_Parser` – Klasse wird im nächsten Abschnitt vorgestellt.

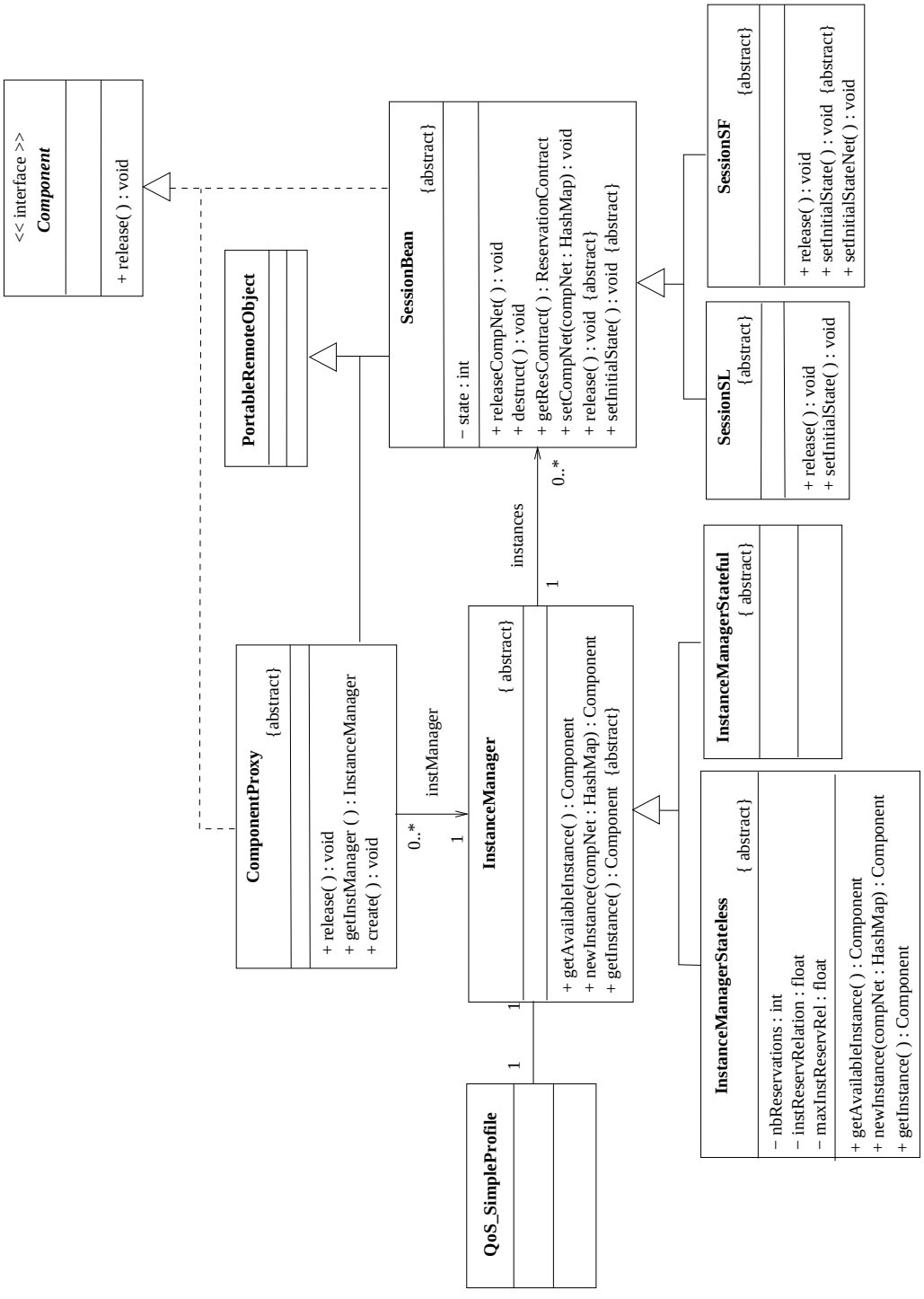


Abbildung 4.4: Verwaltung der Instanzen eines einfachen Profils

Einlesen der *QoS* – Eigenschaften

QoS_Parser

Sobald eine Implementierungen registriert wurde, werden durch Ausführung der `parseProfileForComponent` – Methode im `QoS_Parser` die *QoS* – Eigenschaften zu der registrierten Implementierung eingelesen. Die Abbildung der Eigenschaften erfolgt auf die im Kapitel 3 ab Seite 20 dargestellten Klassenstrukturen. Dabei werden für angebotene und geforderte Eigenschaften die statistischen Werte vom Typ *Range* in *Minimum* und *Maximum* zerlegt, um den zur Laufzeit erfolgenden Vergleich zu erleichtern.

Wurden alle Eigenschaften eingelesen, ist der Container für Zugriffe bereit. Die Verwaltung der dabei erstellten Instanzen wird im nächsten Abschnitt beschrieben.

4.2.2 Verwaltung der Instanzen

Instanzen Manager

Die zur Laufzeit erstellten Instanzen werden durch einen Instanzen – Manager (`InstanceManager`) verwaltet. Da jedes einfache Profil unterschiedliche Ressourcenanforderungen stellt, wird jedem `QoS_SimpleProfile` – Objekt eine eigene Instanz von `InstanceManager` zugewiesen (Abb. 4.4, S. 77).

Zu den Aufgaben des Instanzen – Managers gehören das Erstellen und Löschen der Objekte sowie ihre Reservierung und Freigabe.

Erstellung

Bei der Erstellung der Instanzen (`newInstance` – Methode) werden die benötigten Betriebsmittel reserviert. Dazu wird die `negotiate` – Methode des *QoS* – Managers aufgerufen. Da dabei die Resourcenanforderungen übergeben werden müssen, speichert der `InstanceManager` eine Referenz auf das zugehörige Profil (`QoS_SimpleProfile`). Konnten die Ressourcen reserviert werden, wird eine neue Instanz erstellt. Dazu wird ihr das beim Aufruf der `newInstance` – Methode übergebene Netz aus abhängigen Instanzen (`compNet` – `HashMap`) zugewiesen.

Verwaltung

Alle Objekte eines Profils einer Komponenten – Implementierung werden in der `HashMap instances` verwaltet. Zur eindeutigen Identifizierung einer Instanz wird vom Instanzen – Manager eine Zahl zugewiesen, die in `instances` als Schlüssel dient.

Löschen

Das Löschen einer Instanz kann nur durch den Instanzen – Manager erfolgen. Dazu werden folgende Schritte ausgeführt:

1. Durch Aufruf der `releaseCompNet` – Methode in `SessionBean` werden die Reservierungen von abhängigen Instanzen freigegeben. Es wird auf alle Einträge im Komponenten – Netz (`compNet`) die `release` – Methoden ausgeführt und schließlich `compNet` auf `null` gesetzt.

2. Es wird der *Handle* zur Beschreibung der reservierten Ressourcen ausgelesen (`getResContract`).

3. Der Eintrag für das Objekt im `instances` – *HashMap* wird gelöscht.

4. Das Objekt wird durch Ausführung der `destruct` – Methode in `SessionBean` gelöscht und die Referenz darauf wird verworfen (`null`).

5. Durch Ausführung der `release` – Methode im *QoS* – Manager werden die reservierten Ressourcen freigeben. Bei diesem Aufruf wird der Reservierungs – Handle übergeben (S. 90 ff.).

Reservierung

Für die Reservierung von Objekten stellt der Instanzen – Manager zwei Methoden zur Verfügung. Bei Ausführung der `newInstance` – Methode wird wie bereits beschrieben eine neue Instanz erstellt.

Zwischenspeichern

Die `getAvailableInstance` – Methode hingegen versucht eine zwischengespeicherte Instanz zu reservieren. Dazu wird die Verfügbarkeit der bereits erstellten Instanz überprüft. Diese Eigenschaft ist anhand des Attributes `state` erkennbar, welches jedes *Session Bean* von der Oberklasse `SessionBean` erbt. Die Zustände des Attributes `state` haben in den beiden Typen von *Session Beans* unterschiedliche Bedeutung (Tabelle 4.1).

Zustand	***Stateful Session Beans***	***Stateful Session Beans***
0	Das Objekt wurde erstellt. Es ist von keinem Client reserviert worden.	
1	Das Objekt wurde von einem Client reserviert. Die `create` – Methode wurde noch nicht ausgeführt.	Ein Client führt eine Geschäftsmethode aus und reserviert währenddessen das Objekt.
2	Die `create` – Methode wurde ausgeführt.	Dieser Zustand tritt nie ein.

Tabelle 4.1: Spezifizierung des Zustands der Instanzen durch das Attribut `state`

Ist keine Instanz verfügbar, ist die Antwort der `getAvailableInstance` – Methode `null`. Andernfalls wird eine Instanz reserviert. Dieser Vorgang unterscheidet sich bei *Stateful* und *Stateless Session Beans*.

Stateful

Stateful Session Beans: Da jede aufrufende Instanz (aufrufender Client) ein eigenes Objekt des *Stateful Session Beans* erhält, gibt der Instanzen – Manager eine Referenz auf ein verfügbares Objekt zurück. Das Attribut `state` wird auf `1` gesetzt. Für den Fall, dass eine Reservierung nicht aufgehoben wird, wenn zum Beispiel die Client – Anwendung vergisst die Reservierung freizugeben oder ausfällt, wird ein *Timeout* verwendet, der nach jeder Beendigung einer Methode zu starten ist.

Stateless

Stateless Session Beans: Bei *Stateless Session Beans* teilen sich viele Clients die existierenden Instanzen. Daher gliedert sich der Reservierungsprozess in zwei Schritte:

- Registrierung der Reservierung im Instanzen – Manager und
- Zuweisung einer Instanz beim Zugriff auf eine Geschäftsmethode.

Zur Registrierung der Reservierungen wird im Instanzen – Manager die Anzahl aller Reservierungen gespeichert (Attribut `nbReservations`).

Untergrenze

Zur Überprüfung der Verfügbarkeit wird bei jeder Reservierung das Verhältnis der Anzahl der erstellten Instanzen zu der Anzahl der Reservierungen ermittelt. Dieses Verhältnis wird mit dem Attribut `instReservRelation` verglichen, welches das minimale Verhältnis der beiden Werte beschreibt. Diesem Attribut wird beim Start des Servers ein Wert zugewiesen, der im folgenden Bereich liegen kann:

$$0 < \texttt{instReservRelation} \leq 1$$

Der Wert `1` beschreibt, dass je Reservierung mindestens eine Instanz existieren muss. Die Höhe des Wertes kann sich an den *QoS* – Eigenschaften orientieren. Wird zum Beispiel eine sehr gute Antwortzeit verlangt, sollte der Wert von `instReservRelation` möglichst groß sein.

Trifft eine Reservierungsanfrage ein, und es existiert keine zwischengespeicherte Instanz oder das neue Verhältnis aus Instanzen und Reservierungen unterschreitet den minimalen Wert, muss eine neue Instanz erstellt werden.

Obergrenze

Das beschriebene Verfahren kann erweitert werden, indem das Attribut `maxInstReservRel` eingeführt wird.

$$\texttt{instReservRelation} \leq \texttt{maxInstReservRel}$$

Es spezifiziert das maximale Verhältnis aus zwischengespeicherten Instanzen und Reservierungen. Mit dessen Hilfe kann festgestellt werden, ob viel mehr Objekte existieren als Reservierungen vorliegen. Da diese begrenzte Ressourcen blockieren, müssen sie von dem Instanzen – Manager gelöscht werden. Das Verhältnis aus Instanzen und Reservierungen wird nach jeder Freigabe einer Reservierung überprüft. Wurde die letzte Reservierung aufgehoben, bleibt die Anzahl der Instanzen erhalten.

Verweis auf Verwaltung

Bei der Verwendung von *Stateless Session Beans* wird erst beim Zugriff auf die Geschäftsmethoden eine eigene Instanz benötigt. Deshalb wird bei der Reservierung eine Referenz auf den Instanzen – Manager zurückgegeben. Da dieser nicht der Schnittstelle genügt, muss durch Anwendung des *Proxy* – Musters ein Objekt vorgelagert werden, das diese Anforderung erfüllt.

Proxy

Das von der `ComponentProxy` – Klasse abgeleitete *Proxy* – Objekt implementiert darum die Spezifikation der verwalteten Implementierung. Desweiteren beinhaltet es einen Verweis auf den Instanzen – Manager, um während der Ausführung einer Methode eine Instanz anfordern zu können. Die *Proxy* – Klassen werden vor dem Start der Komponenten – Verwaltung aufgrund der Informationen im Beziehungs – Deskriptor und unter Verwendung von *XSLT* [32][31] generiert. Der Aufbau und die Funktionsweise dieses Objektes wird anhand eines Beispiels im Anhang A beschrieben.

Greift ein Client auf eine Geschäftsmethode zu, fordert das *Proxy – Objekt* eine Instanz beim Instanzen – Manager an (`getInstance` – Methode). In dieser Situation werden die gleichen Funktionen wie bei der Reservierung eines *Stateful Session Beans* ausgeführt. Nach Beendigung der Geschäftsmethode ruft das *Proxy – Objekt* die `release` – Methode der reservierten Instanz auf, um sie für andere Zugriffe freizugeben. Bei Ausführung dieser Methode wird in *Stateless Session Beans* nur das Attribut `state` auf `0` gesetzt.

Reservierung freigeben

Zur Freigabe einer Reservierung ruft der Client die `release` – Methode auf. Empfängt eine Instanz diesen Aufruf, wird er in den beiden Typen der *Session Beans* unterschiedlich behandelt.

Stateful

Stateful Session Beans: Da in *Stateful Session Beans* ein Konversationszustand verwaltet wird, müssen diese in einer solchen Situation einen definierten Anfangszustand annehmen. Es besteht die Möglichkeit, die Instanz zu löschen und eine neue, die die gleichen Ressourcen verwendet, zu erstellen. Als eine alternative Lösung kann in der Komponenten – Verwaltung die `setInitalState` – Methode aufgerufen werden, die durch den Anwendungsentwickler zu implementieren ist. Da auch das Netz aus abhängigen Instanzen zwischengespeichert wird, muss dieser Aufruf an alle Einträge weitergereicht werden.

Zu diesem Zweck wurde die `setInitalStateNet` – Methode eingeführt. In *Stateful Session Beans* ruft sie die `setInitialState` – sowie in allen Einträgen in `compNet` die `setInitalStateNet` – Methode auf. In *Stateless Session Beans* muss diese Methode nicht implementiert werden, da diese *Session Beans* keinen Konversationszustand verwalten und keine *Stateful Session Beans* als Nachfolger haben können. Die Methode besitzt daher in `SessionSL` einen leeren Methodenkörper.

Timeout

Nachdem alle Instanzen den Anfangszustand angenommen haben, wird im freigegebenen Objekt das Attribut `state` auf `0` gesetzt und die *Timeout* – Überwachung gestartet. Dieses *Timeout* überwacht die Zeit bis zur nächsten Reservierung eines Netzes. Da aber auch ein *Timeout* eingeführt wurde, das die Zeiträume zwischen zwei Methodenaufrufen in reservierten *Stateful Session Beans* überwacht, kann das zu Problemen bei zwischengespeicherten Netzen führen. Damit die gespeicherten Netze nicht unbrauchbar werden, sollte in abhängigen Instanzen die *Timeout* – Funktionalität deaktiviert werden, da diese sonst gelöscht werden könnten.

Stateless

Stateless Session Beans: Empfängt ein *Stateless Session Bean* die `release` – Methode, ruft das *Proxy – Objekt* die `release` – Methode im `InstanceManager` auf. Dieser registriert, dass eine Reservierung aufgehoben wurde, indem er den Wert von `nbReservations` um eins verringert.

Freigabe Komponenten – Netz

In beiden Fällen wird erst bei endgültiger Löschung einer Instanz und Freigabe der reservierten Betriebsmittel durch den Instanzen – Manager die Reservierung der abhängigen Objekte durch Aufruf der `release` – Methode aufgehoben.

4.2.3 Vertragsmanager

Auslösung Netzerstellung

Fordert eine Client – Anwendung eine Referenz auf eine Komponente der Server – Anwendung an, führt sie die `getRemoteHome` – Methode aus und übergibt zur Spezifikation der geforderten Schnittstelle den im Beziehungs – Descriptor verwendeten Bezeichner. Die Methode wird von der Client – Umgebung realisiert und leitet den Aufruf an die `reserveComponent` – Methode in der Container – Implementierung weiter. Dabei wird der Name der geforderten Schnittstelle, der Bezeichner aus dem Beziehungs – Deskriptor und die von der Client – Anwendung geforderten *QoS* – Eigenschaften übergeben.

Netz – Lebenszyklus

Bei der Ausführung der Methode erstellt der Vertragsmanager das Netz aus kommunizierenden Komponenten, reserviert die Betriebsmittel und liefert eine Referenz auf die Wurzel des Netzes zurück. Daraufhin können in der Client – Anwendung die Geschäftsmethoden dieses Objekts ausgeführt werden. Beendet ein Client seine Anfrage, wird die Reservierung entlassen. Auf die Implementierung dieser Vertragsmanagerfunktionen wird in den folgenden Abschnitten ausführlich eingegangen.

4.2.3.1 Aushandlung der Verträge und Reservierung der Ressourcen

Voraussetzungen

Für die Vertragsaushandlung werden folgende Informationen benötigt:

- Name der geforderten Schnittstelle,
- den im Beziehungs – Descriptor verwendeten Bezeichner, da zu einer Schnittstelle mehrere Instanzen mit unterschiedlichen *QoS* – Anforderungen reserviert werden können und
- die geforderten *QoS* – Eigenschaften

In den anschließenden Betrachtungen wurde der Name der Schnittstelle bereits ausgewertet und die zugehörige Beschreibung (`Specification`) ausgewählt. Desweiteren wird die Prüfung der einfachen Profile abgebrochen, sobald ein geeignetes gefunden wurde (`getContract` – Methode). Sollen alle zulässigen Profile ermittelt werden, ist die `getAllContracts` – Methode auszuführen.

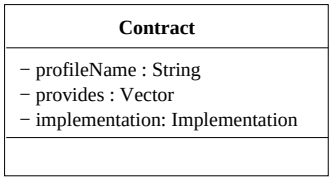


Abbildung 4.5: Vertrag – erstellt bei Prüfung von `Implementation`

Vertragsaushandlung

Wird zu einer Spezifikation eine Instanz gesucht, ist in allen Implementierungen dieser Schnittstelle die Methode `getContract` aufzurufen. Dabei wird der Bezeichner aus dem Beziehungs – Deskriptor und ein `Vector` mit den geforderten Anweisungen übergeben. Während der Ausführung der Methode werden bei erfolgreicher Suche nach einem geeigneten Profil die im Sequenzdiagramm 4.6 dargestellten Funktionen ausgeführt.

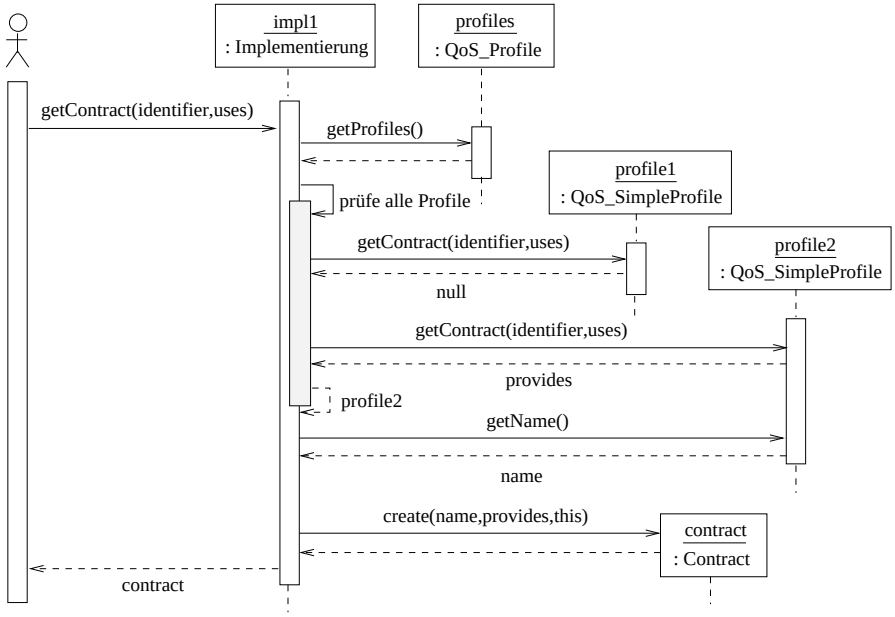


Abbildung 4.6: Sequenzdiagramm: `getContract` in `Implementation`

Es erfolgt die Untersuchung aller einfachen Profile der Implementierung. Sobald eines die Anforderungen erfüllt, wird ein Vertrag (`Contract`, Abb. 4.5) erstellt und an die aufrufende Implementierung (oder `ContainerImpl` anstelle des Clients) übergeben. Diese sucht daraufhin in allen Verträgen, die sie zu der Spezifikation erhalten hat, nach der optimalen Implementierung. Dabei können zum Beispiel die `best-effort` – Forderungen ausgewertet werden.

Reservierung

Nachdem ein Vertrag ausgewählt wurde, wird die zugehörige Implementierung reserviert. Dazu wird die `reserveInstance` – Methode im `Implementation` – Objekt ausgeführt. Zwei alternative Funktionsweisen dieser Methode sind in den folgenden Sequenzdiagrammen 4.7 und 4.8 dargestellt. Im Diagramm 4.7 ist eine zwischengespeicherte Instanz verfügbar.

Komponenten – Netze

Im zweiten Beispiel (Abb. 4.8, S. 85) müssen die abhängigen Instanzen und Ressourcen neu reserviert werden. Das erstellte abhängige Teilnetz (`compNet`) wird zur Reservierung der neuen Instanz dem Instanzen – Manager übergeben. Dieser weist der Instanz die vollständige `HashMap` durch Aufruf der `setCompNet` – Methode zu. Die neue Instanz wird ins Komponenten – Netz der aufrufenden Implementierung aufgenommen. Als Schlüssel dient der Bezeichner aus dem Beziehungs – Deskriptor. Auf diese Weise können für die selbe Spezifikation mehrere Instanzen mit unterschiedlichen *QoS* – Anforderungen reserviert werden.

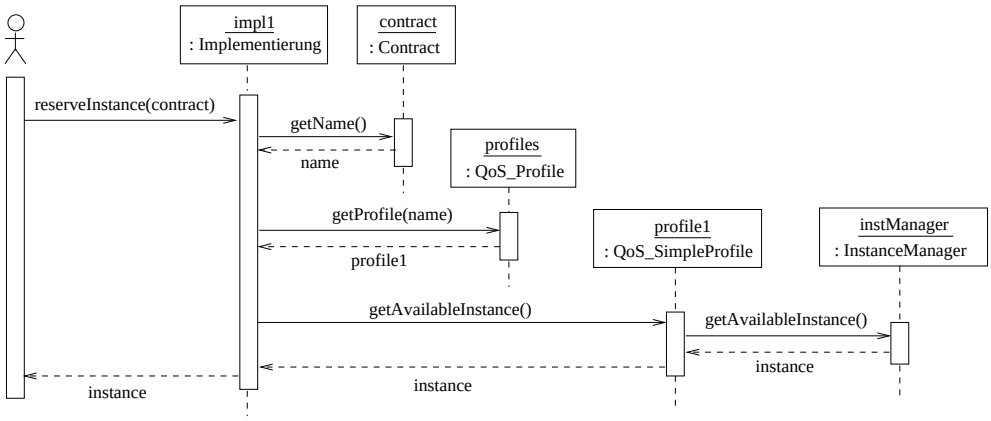


Abbildung 4.7: Sequenzdiagramm: Reservierung zwischengespeicherter Instanz

Scheitern der Reservierung

Kann zu einer geforderten Spezifikation keine Instanz verfügbar, müssen die eventuell schon reservierten abhängigen Instanzen durch Aufruf der `release` – Methode freigegeben werden. An das aufrufende Objekt wird in dem Fall `null` zurückgegeben. Dieses versucht daraufhin, eine Instanz einer anderen geeigneten Implementierung zu reservieren.

Die Funktionsweise der Methode zur Eignungsprüfung eines einfachen Profils (`getContract` in `QoS_SimpleProfile`) wird in den folgenden Abschnitten beschrieben.

Untersuchung der Eignung eines Profils

Profilprüfung

Im Kapitel 3 wurden die Konzepte für die Untersuchung der Eignung eines Profils erarbeitet (S. 44 ff.). In diesem Abschnitt wird auf die Implementierungsdetails eingegangen.

Bei der Aushandlung der Verträge wurde für eine geforderten Spezifikation eine Implementierung und ein einfaches Profil dieser Implementierung ausgewählt. Es sollen die durch dieses Profil angebotenen *QoS* – Eigenschaften mit den geforderten verglichen werden.

Für der Überprüfung eines einfachen Profils, bietet `QoS_SimpleProfile` ebenfalls eine `getContract` – Methode an. Bei erfolgreicher Untersuchung wird als Antwort ein Vertrag, der den angebotenen Eigenschaften entspricht, zurückgegeben. Die bei Ausführung dieser Methode folgende Untersuchung von Charakteristiken wird nun ausführlich erläutert.

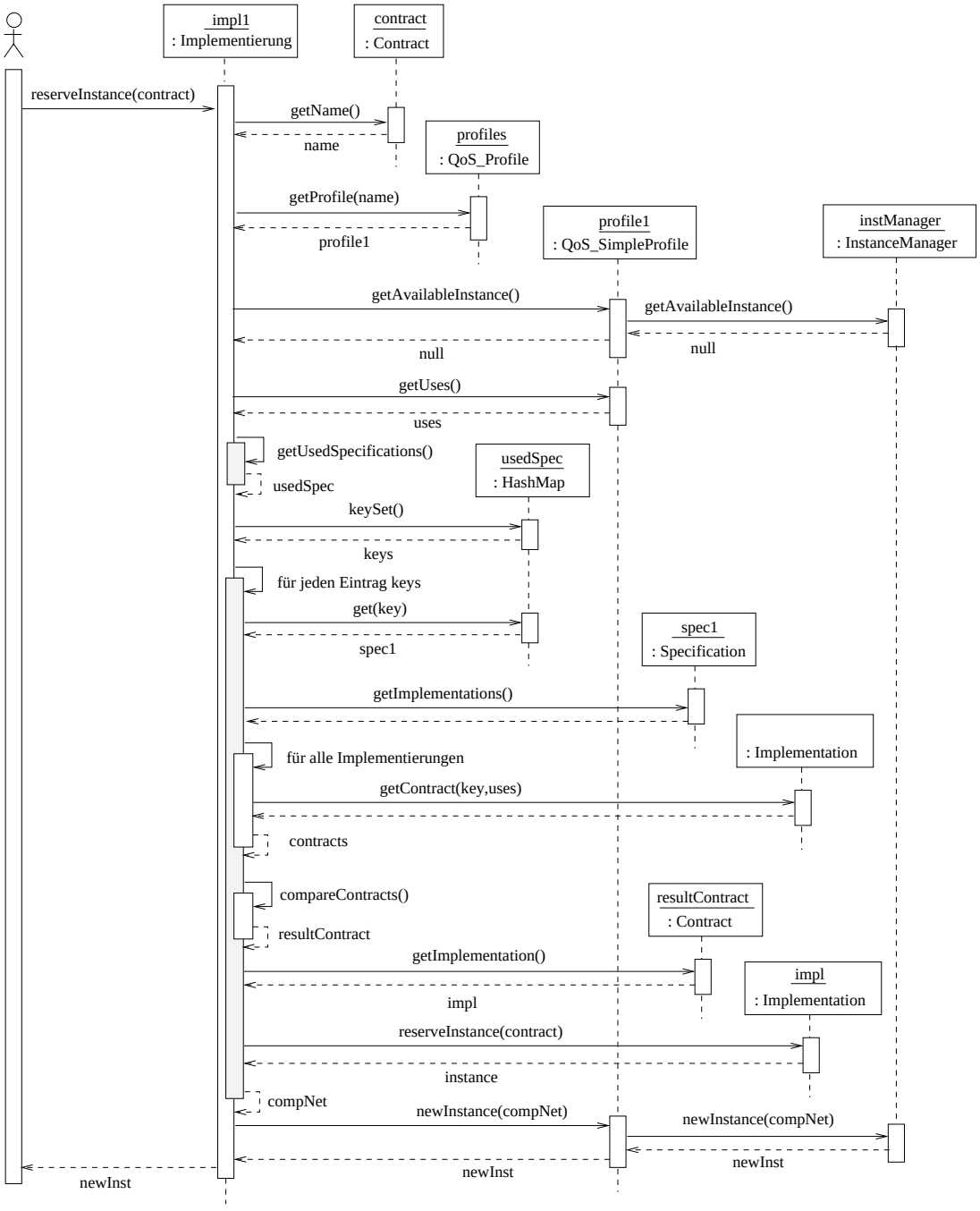


Abbildung 4.8: Sequenzdiagramm: Reservierung neuer Instanz

Bezeichner prüfen

Überprüfung von Charakteristiken: Für jede geforderte Charakteristik wird als erstes geprüft, ob sie von der Implementierung zu erfüllen ist. Dazu wird der im Beziehungs – Descriptor verwendete Bezeichner (`identifier`) ausgewertet. Eine geforderte Charakteristik muss nur erfüllt werden, wenn einer ihrer Parameter diesen `identifier` besitzt. Ist die Charakteristik für die Untersuchung relevant, muss überprüft werden, ob die angebotenen Eigenschaften ihren Anforderungen genügen.

Zur Überprüfung einer bewerteten Charakteristik wird die Methode

```
isEqualOrRestrictive(Comparison comp)
```

Rückgabewerte

umgesetzt. Sie wird von Implementierungen der Schnittstellen `QoS_Statement` und `Constraint` angeboten. Die Methode liefert `'true'`, wenn der angebotene Wert den Anforderungen der als Parameter übergebenen bewerteten Charakteristik (`comp`) genügt. Widerspricht die Charakteristik den Anforderungen wird `'false'` zurückgegeben. Konnte keine eindeutige Aussage über die Eignung oder den Verstoß gegen Anforderungen getätigt werden, wird die `ClassCastException` ausgelöst. Wenn unterschiedliche Charakteristiken miteinander verglichen wurden, löst das ebenfalls die `ClassCastException` aus. Die bei der Untersuchung zu berücksichtigenden Bedingungen wurden detailliert ab Seite 44 beschrieben. An dieser Stelle wird nur die Implementierung des Vergleichs der Beträge vorgestellt.

Vergleich der Wertebereiche

Vergleich der Beträge (`DomainInstance`): Die Werte werden durch einen Vergleichsoperator an die Charakteristiken gebunden. Diese Vergleichsoperatoren beziehen sich nicht auf die Ordnungsrichtung der Implementierungen. Die Ordnungsrichtung wird erst bei der Suche nach dem besseren Wert verwendet. Für Vergleiche werden daher von den Implementierungen der `DomainDefinition` – Schnittstelle zwei verschiedene Methoden angeboten:

`compare` - ohne Betrachtung der Ordnungsrichtung und
`isBetter` - Beachtung der Ordnungsrichtung.

Ein Element (`DomainInstance`) kann mit einem anderen verglichen werden, indem eine der Methoden `compareTo` oder `isBetterThan` aufgerufen wird. Die Methode leitet die Anfrage an die zugehörige Klasse, die der `DomainDefinition` – Schnittstelle genügt, weiter.

Die Methoden ermitteln folgende Werte:

`= 0` - die Beträge sind gleich,
`< 0` - der erste Betrag ist kleiner oder
`> 0` - der erste Betrag ist größer.

Der Vergleich zweier Beträge aus den verschiedenen Wertebereichen ist [6] entnommen. Es wird nachfolgend die Funktionsweise der Methode unter Verwendung der Ordnungsrichtung vorgestellt.

Numeric: Mit `Numeric` werden die Datentypen `Integer`, `Real` and `Natural` zusammengefasst. Der Wert wird in der `NumericInstance` – Klasse in einer `Float` – Variable gespeichert. Beim Vergleich zweier Werte wird dieser Vergleich an die `compareTo` – Funktion im Datentyp `Float` weitergeleitet. *Numeric*

Beim Vergleich von *Decreasing* – Werten muss dagegen nur das Vorzeichen geändert werden.

Enumeration: Die Ordnung der Einträge in einer `Enumeration` erfolgt in *CQML*$^+$ jeweils durch Definition der Beziehungen zwischen zwei Einträgen, wie zum Beispiel: *Enum*

```
a < b, b < c, c < d, c < e
```

Die dadurch definierte Ordnung (total oder partial) ist *transitiv*. Die Paare werden durch Felder mit zwei Einträgen beschrieben und in einem `Vector` zusammengefasst.

Zur Verwaltung einer sortierten `Enumemeration` muss die Ordnungsrichtung spezifiziert werden. Diese Information wird in der `int` – Variable `direction` hinterlegt. Sie kann folgende Werte annehmen:

```
 0  - unsortiert,
 1  - increasing oder
-1  - decreasing.
```

BeimVergleich zweier Beträge einer unsortierten `Enumeration` kann nur die Gleichheit der Einträge festgestellt werden. Andernfalls wird die `java.lang.ClassCastException` ausgelöst.

Für einen Betragsvergleich einer sortierten Menge muss die Beziehung der Werte definiert sein (direkt oder transitiv). Wurde die Beziehung nicht beschrieben, wird die `java.lang.ClassCastException` ausgelöst.

In einem *Decreasing* – Wertebereich muss beim Vergleich das Vorzeichen geändert werden.

Set: Die Einträge eines `Set` – Wertebereichs werden in *CQML*$^+$ auf die gleiche Weise wie in der `Enumeration` sortiert. Daher erfolgt die Abbildung der Beziehungen ebenfalls auf die gleiche Art. *Mengen*

Beim Mengenvergleich wird untersucht, ob die eine in der anderen enthalten ist. Ist eine Menge vollständig enthalten, ist sie kleiner. Bestehen beide Mengen aus den gleichen Einträgen, gelten sie als gleich gross. Enthalten die beiden Teilmengen nach Abzug der ermittelten Schnittmenge noch Einträge, ist ein Vergleich der unsortierten Mengen nicht möglich. In dem Fall wird die `java.lang.ClassCastException` ausgelöst.

Sollen zwei Mengen eines sortierten Wertebereichs verglichen werden, können die unabhängigen Teilmengen ausgewertet werden. Dies geschieht durch Gegenüberstellung der jeweils größten Einträge. Es gilt die Menge als die größere, deren unabhängige Teilmenge den größten Eintrag enthält. Der größte Wert

einer Teilmenge lässt sich nur ermitteln, wenn für alle Einträge die Beziehung definiert wurde. Andernfalls wird die `java.lang.ClassCastException` ausgelöst.

Werden zwei Mengen eines *Decreasing* – Wertebereichs verglichen, ist die Änderung des Vorzeichens unzureichend. Es muss stattdessen bei den unabhängigen Teilmengen nach dem kleinsten Eintrag gesucht werden. Die Menge mit dem 'kleinsten' Eintrag gilt als die größere.

Nach dem Vergleich der Beträge muss durch Auswertung der Vergleichoperatoren festgestellt werden, ob das Werteverhältnis die Anforderungen genügt.

In dem abgeschlossenen Abschnitt wurden die Implementierungsdetails zur Umsetzung der im vorangegangenen Kapitel erarbeiteten Konzepte zur Vertragsaushandlung vorgestellt. Nun soll auf die Vertragsdurchsetzung eingegangen werden.

4.2.3.2 Durchsetzung der Verträge

Nachdem die Verträge ausgehandelt und die Ressourcen sowie das Komponenten – Netz reserviert wurden, kann die Anfrage erfolgen. Die Implementierung der Vertragsdurchsetzung über den Komponten – Proxy wird in diesem Abschnitt beschrieben. Aufgrund der Komplexität des Ressourcen – Proxies kann dessen Umsetzung im Rahmen dieser Arbeit nicht behandelt werden. Grundlegende Konzepte und Problemstellungen wurden bereits erwähnt (S. 64 ff.).

Kommunikations – Proxy: Zuweisung der reservierten Instanzen

Auswahl reservierter Instanzen

Die reservierten Instanzen werden zugewiesen, wenn in der Server – Anwendung die `getLocalHome` – Methode aufgerufen wird. Dabei wird der im Beziehungs – Descriptor verwendete Bezeichner übergeben. Dieser steht dem Anwendungsentwickler zur Verfügung, da er ebenfalls die Datei zur Spezifizierung der Beziehungen erstellt hat.

Bei Ausführung der Methode wird der Eintrag zu diesem Bezeichner ausgewählt und an die aufrufende Instanz übergeben. Im Anschluss kann diese unabhängig von der Komponenten – Verwaltung auf die Geschäftsmethoden zugreifen. Ist eine zusätzliche Überwachung erwünscht, kann diese über die *Interceptoren* eingefügt werden (Abschnitt 4.1).

In den vorangegangenen Abschnitten wurden die Methoden zur Umsetzung der Funktionalität der Komponenten – Verwaltung im Server vorgestellt. Um die Verarbeitung der *QoS* – Eigenschaften in Anwendungen zu integrieren, müssen diese angepasst werden. Auf die erforderliche Gestaltung der Komponenten wird nun eingegangen.

4.2.4 Gestaltung einer Server – Anwendung

Bei der Entwicklung von Anwendungen, die in der Komponenten – Verwaltung verarbeitet werden sollen, sind einige Anpassungen notwendig. Durch den Einsatz von *Aspektorientierter Programmierung* zur Integration müssen die nachfolgend beschriebenen Namenskonventionen eingehalten werden. Anschließend folgt die Vorstellung der funktionalen Erweiterungen der Komponenten – Implementierungen.

Namenskonventionen

Session Bean Endungen

Für die Kennzeichnung von *Session Beans* wurden wie für *Entity Beans* (S. 15) Namenskonventionen eingeführt. Die *Session Beans* müssen folgende Endungen annehmen:

Stateful Session Beans - Endung: `_SF`
Stateless Session Beans - Endung: `_SL`

Schnittstellen Endungen

Die Endungen für die Schnittstellen bleiben wie bei den *Entity Beans* erhalten.

***Home* – Schnittstelle** - Endung: `RemoteHome`
***Remote* – Schnittstelle** - Endung: `Home`

Integration

Durch Einhaltung der beschriebenen Konventionen können die *Session Beans* und ihre Schnittstellen von dem Aspekt `Integration` in die Laufzeitumgebung eingefügt werden.

Stateful Session Bean - Ableitung von `SessionSF`
Stateless Session Bean - Ableitung von `SessionSL`
Home – Schnittstelle - Ableitung von `RemoteHomeQoS`
Remote – Schnittstelle - Ableitung von `RemoteQoS`

Da alle Komponenten - Implementierung einer Spezifikation die gleichen *Home* – und *Remote* – Schnittstellen anbieten, können diese mit einem einheitlichen Namen zusammengefasst werden. Sie unterscheiden sich dann nur durch die Endungen. Da mehrere Implementierungen den Schnittstellen hinterlegt werden, können sie nicht durch den gleichen Namen eindeutig gekennzeichnet werden.

Es besteht jetzt die Möglichkeit über die `SubscriptionManagerRemote` – und `SubscriptionManagerRemoteHome` – Schnittstellen auf `SubscriptionManager1_SL` und `SubscriptionManagerSpecial_SL` zuzugreifen. Da die Implementierungen und Schnittstellen nicht über die Namenskonventionen zusammengefasst werden können, muss bei jeder Schnittstelle registriert werden, von welcher Komponenten – Implementierung sie implementiert wird. Diese Aufgabe erfüllt ein `Specification` – Objekt.

Anforderungen an die Implementierung

Implementierungserweiterungen

Zusätzlich zu den Namenskonventionen müssen die Komponenten – Implementierungen folgenden Anforderungen genügen:

- Der Default – Konstruktor ist umzusetzen.
- Desweiteren müssen alle Konstruktoren eine `throws` – Klausel mit der `java.rmi.RemoteException` aufnehmen, da diese Klausel nicht durch einen Aspekt eingefügt werden kann. Die *Exception* kann auftreten, da die Komponenten vom `java.rmi.PortableRemoteObject` abgeleitet werden, um auf sie via *RMI* zugreifen zu können.
- Aus den selben Gründen müssen alle Methoden der Schnittstellen die `java.rmi.RemoteException` in der `throws` – Klausel aufnehmen.
- In *Stateful Session Beans* muss zusätzlich die `setInitialState` – Methode umgesetzt werden. Sie setzt alle Attribute auf die Werte ihre Defaultinitialisierung. Diese Methode wird nicht über eine Schnittstelle der Komponente angeboten.

Durch Verwendung des *AspectJ* – Compilers werden die Komponenten mit dem Quellcode der Laufzeitumgebung verwoben. Dadurch können sie in der Komponenten – Verwaltung verarbeitet werden.

4.2.5 Zusammenfassung

Die serverseitige Komponenten – Verwaltung bietet eine Laufzeitumgebung zur Verarbeitung von *QoS* – Anforderungen. Bei Einhaltung von Konventionen ist die Möglichkeit gegeben, die Komponenten in der Laufzeitumgebung zu integrieren. Bei der Reservierung der Betriebsmittel muss auf eine Ressourcen – Verwaltung zugegriffen werden. Die im Abschnitt 2.1.1 vorgestellte Schnittstelle wurde für die Arbeit mit der Komponenten – Verwaltung angepasst. Deren Funktionalität wird anschließend vorgestellt.

4.3 Anpassung der Schnittstelle zur Ressourcen – Verwaltung

Im zweiten Kapitel wurde die Schnittstelle zum Zugriff auf die Ressourcen – Verwaltung bereits vorgestellt (S. 4 ff.). Zur prototypischen Umsetzung der Funktionalität wurde der `QoS_Manager` implementiert. Um die Zusammenarbeit mit der Komponenten – Verwaltung zu ermöglichen, waren einige Anpassungen notwendig. Der Aufbau der `client_id` wurde an die Verarbeitung in einem in *Java* implementierten Container angepasst. Desweiteren wurden die Rückgabewerte der Methodenaufrufe zusammengefasst. Diese Modifikationen werden auf den folgenden Seiten detailliert beschrieben.

Identifizierung des Clients

ID Vergabe

Bei der Reservierung benötigt die Ressourcen – Verwaltung eine eindeutige Identifikationsnummer des 'Clients'. Die Zuweisung dieser *ID* kann auf verschiedenen Ebenen erfolgen:

1. eine *ID* je Client – Anwendung,
2. eine *ID* für die gesamte Server – Anwendung,
3. eine *ID* je Implementierung,
4. eine *ID* je Profil (b z w. Instanzen – Manager) oder
5. eine *ID* je Instanz.

Da die Instanzen zwischengespeichert und damit von vielen Client – Anwendungen reserviert werden, stellt die Variante 1 keine sinnvolle Lösung dar.

Die Varianten 2 bis 4 sind ebenfalls unzureichend. Beim Eintreffen einer Information über Änderungen der Verfügbarkeit von Ressourcen kann die Instanz, die auf diese zugreift, nicht zugeordnet werden. Es ist daher erforderlich direkt auf die Instanzen zu verweisen (Vorschlag 5).

`ClientId`

Da die Reservierung der Ressourcen von den Instanzen – Managern durchgeführt wird, und eine Freigabe ebenfalls durch sie zu erfolgen hat, müssen sie ebenfalls über die Änderung der Verfügbarkeit von Ressourcen informiert werden. Um die Komponenten – Verwaltung und den Instanzen – Manager in der Ressourcen – Verwaltung eindeutig identifizieren zu können, besteht die `ClientId` aus drei Einträgen (Abb. 4.9). Der dritte Eintrag dient zum Verweis auf den Container, in dem sich die Instanz und ihr Instanzen – Manager befindet.

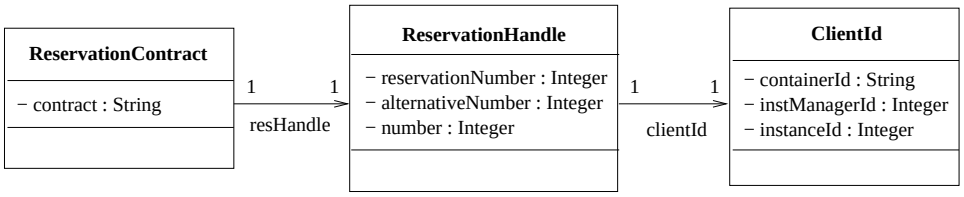


Abbildung 4.9: Abbildung des Vertrages mit der Ressourcen – Verwaltung

Instancen – Manager – ID

ID Erstellung: Bei der Erstellung eines `InstanceManager` – Objektes wird von der `ContainerImpl` eine Zahl, zur eindeutigen Kennzeichnung des Instanzen – Manager in dieser Komponenten – Verwaltung, angefordert. Die Container – Implementierung generiert beim Eintreffen der Anfrage eine `instManagerId`. Mit dieser Zahl wird der Instanzen – Manager im Container eindeutig beschrieben. Die Referenz auf das `InstanceManager` – Objekt muss erhalten bleiben, um später auf es zugreifen zu können. Zur Verwaltung dieser Abbildung wird eine `HashMap` verwendet.

Instanzen – ID

Soll eine neue Instanz erstellt werden, generiert der Instanzen – Manager eine Zahl (`instId`), die die neue Instanz innerhalb des Instanzen – Managers eindeutig beschreibt. Aus den beiden zuvor beschriebenen *ID*'s und dem Namen unter dem die Container – Implementierung am *Naming – Server* angemeldet ist, wird eine `ClientId` – Objekt erstellt und bei der Reservierungsanfrage übergeben.

Container – ID

Vertragsänderung: Mit den Informationen im `ClientId` – Objekt kann auf die Instanz verwiesen werden, um sie über Änderungen der Verträge zu informieren. Dazu wird als Erstes der `containerId` – Eintrag ausgewertet. Mit dieser *ID* kann vom *Naming – Server* eine Referenz auf die Container – Implementierung angefordert werden. Da diese die `Notification` – Schnittstelle implementiert, empfängt das `ContainerImpl` – Objekt die Änderungsnachricht. Nach Auswertung des Eintrags `instManagerId` in `ClientId` kann die Container – Implementierung die Referenz auf den Instanzen – Manager auslesen. Daraufhin leitet sie die Nachricht an den Instanzen – Manager weiter, der prüft, ob die Änderungen zulässig sind, oder ob die Instanz vollständig zu löschen ist.

Verträge mit der Ressourcen – Verwaltung

Reservierungs – Vertrag

Der Komponenten – Verwaltung wird nach erfolgreicher Reservierung ein `ReservationHandle` und ein *XML* – `String`, der der Definition *contract.dtd* entspricht, übergeben. Der `ReservationHandle` dient zur Kennzeichnung der Reservierung im *QoS* – Manager, während in dem *XML* – `String` die Verträge mit den einzelnen Ressourcen – Managern beschrieben werden. Diese Informationen werden in einem `ReservationContract` zusammengefasst (Abb. 4.9).

Vertrags-auswertung

Während das `ReservationHandle` – Objekt der neuen Instanz direkt zugewiesen werden kann, muss der `String` ausgewertet werden. Die Informationen werden in `ResourceContract` – Objekten gespeichert. Diese enthalten die Referenzen der zugehörigen, im *QoS* – Repository gespeicherten Ressourcenanforderungen. Die den Instanzen zugewiesenen Informationen werden im `ResourcesContract` – Objekt zusammengefasst (Abb. 4.10).

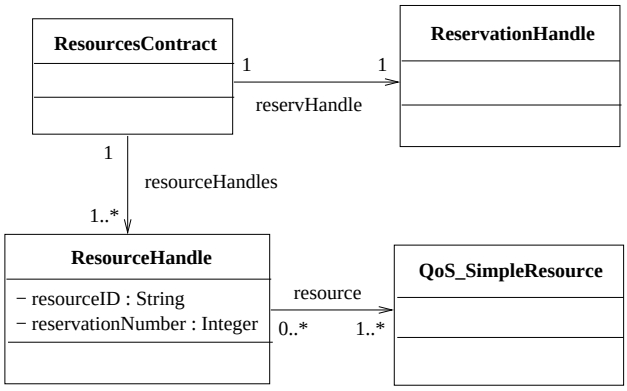


Abbildung 4.10: Abbildung des Vertrages zur Verwaltung in der Instanzen

Zusammenfassung

Über den *QoS* – Manager können Ressourcen reserviert werden. Dazu müssen in der Komponenten – Verwaltung die Ressourcenanfragen und eine *ID* zur eindeutigen Identifizierung der Instanz, für die die Ressourcen zu reservieren sind, generiert werden. Die Ressourcen – Verwaltung erstellt einen Vertrag, mit dem auf die reservierten Ressourcen zugegriffen werden kann. In der Komponenten – Verwaltung werden diese Verträge ausgewertet und ihre Durchsetzung überwacht.

4.4 Verfeinerter Entwurf der clientseitigen Komponenten – Verwaltung

Client – Umgebung

Für den Zugriff auf die Komponenten – Verwaltung muss für Client – Anwendungen ebenfalls eine Laufzeitumgebung geschaffen werden. Bei deren Gestaltung kann ein Großteil der Funktionalität der Komponenten – Verwaltung übernommen werden. Die wiederzuverwendenen Funktionen werden nachfolgend zusammengefasst.

- *QoS* – Repository
  - ohne Definition der Charakteristik und der Wertebereiche
  - ohne Ressourcenanforderungen (Da in dieser Version keine Ressourcen für den Client reserviert werden, müssen die `resources` – Einträge nicht ausgewertet werden.)

  Da die *QoS* – Eigenschaften übers Netzwerk transportiert werden müssen, implementieren alle Einträge im *QoS* – Repository die Schnittstelle `java.io.Serializable`.
- Implementierungsverwaltung
  - ohne Instanzen – Verwaltung
  - ohne angebotene Schnittstelle

Zusätzlich wird für die Client – Anwendungen eine *Session* – Verwaltung eingeführt. In der Client – Umgebung kann auf die Funktionalität der Vertragsaushandlung des Vertragsmanagers vollständig verzichtet werden.

Die Funktionsweise der Aufgabenbereiche der Client – Umgebung werden nachfolgend beschrieben.

4.4.1 Aufgaben der Client – Umgebung

Zur Integration der Client – Anwendungen müssen sie von der `QoS_Client` – Klasse abgeleitet werden. Diese Aufgabe erfüllt ein `Integration` – *Aspekt*. Alle Klassen, die mit ihm zusammen kompiliert werden, werden um die Ableitung und somit um die Funktionalität der Client – Umgebung erweitert.

Informationsverwaltung

Die Verwaltung der *QoS* – Eigenschaften für einen Client beinhaltet nur die geforderten Eigenschaften (*uses*).

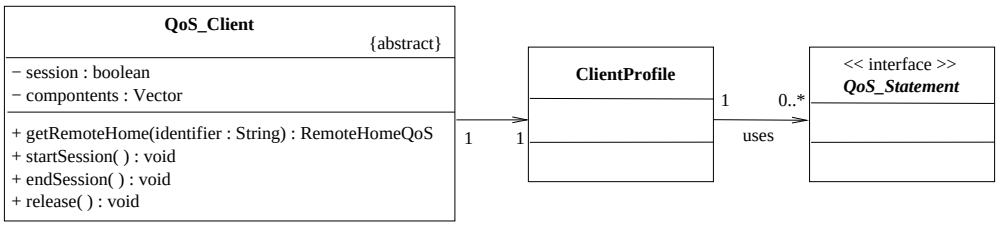


Abbildung 4.11: Verwaltung der Profile von Client – Anwendungen

Diese Eigenschaften müssen beim Start der Client – Umgebung eingelesen werden. Es ist daher erforderlich, dass für jede Client – Anwendung die *XML* – Dateien zur Spezifikation der Beziehungen und der *QoS* – Eigenschaften erstellt werden.

Da nicht alle Einträge der $CQML^+$ – Spezifikation benötigt werden, liest ein modifizierter Parser die Eigenschaften ein. Nach Beendigung des Einlesevorgangs kann die Client – Anwendungen ausgeführt werden.

Methoden

Damit auf die Komponenten – Verwaltung zugegriffen werden kann, müssen von der Client – Umgebung folgende Funktionen angeboten werden:

- die `getRemoteHome` – Methode, zur Reservierung von Instanzen,
- eine *Session* – Verwaltung und
- die `release` – Methode, zur Freigabe einer Reservierung.

Reservierung

Reservierung einer Instanz: Will ein Client auf ein Objekt im Server zugreifen, muss er die `getRemoteHome` – Methode aufrufen und den im Beziehungs – Deskriptor verwendeten Bezeichner übergeben. Die Client – Umgebung liest die für die Client – Anwendung registrieren *QoS* – Eigenschaften aus und ruft die `reserveComponent` – Methode des Objektes, das der `Container` – Schnittstelle genügt, im Server auf.

`reserveComponent(`*`Bezeichner, SpezifikationsName, uses-Eigenschaften`*`)`

Der Container erstellt bei Ausführung der `reserveComponent` – Methode das Netz aus kommunizierenden Instanzen. An die Client – Umgebung wird eine Referenz auf das erste Objekt im Netz (Wurzel) übertragen. Diese Referenz wird von der Client – Umgebung an die Client – Anwendung weitergeleitet.

Zugriff

Zugriff auf Geschäftsmethoden: Bei der Ausführung der `create` – und der Geschäftsmethoden ist keine Überwachung durch die Client – Umgebung erforderlich. Die Anfragen werden direkt an den Server übergeben. Dort führt die Komponenten – Verwaltung die gleichen Funktionen wie bei der Kommunikation zwischen Komponenten – Implementierungen aus.

Freigabe

Freigabe einer Reservierung: Beendet eine Client – Anwendung ihre Anfrage an ein einzelnes Objekt, ruft sie die `release` – Methode auf. Diese wird direkt auf die Wurzel des reservierten Netzes ausgeführt.

Da jede Reservierung die Ressourcen vor weiteren Zugriffen blockiert, sollten alle Objekt, die nicht mehr benötigt werden, sofort nach Beendigung der Anfragen freigegeben werden.

Die Client – Umgebung muss die Ausführung der `release` – Methode nicht überwachen. Ihre Aufgabe ist es, sicherstellen, dass nach Beendigung der Methode, die Referenz aufgehoben wird, damit innerhalb der Client – Anwendung nicht noch einmal auf das zwischengespeicherte Objekt zugegriffen werden kann.

Session – Verwaltung

Session – Verwaltung: Ein Client kann auf mehrere Objekte zugreifen. Jedem dieser Objekte ist ein Netz aus kommunizierenden Instanzen hinterlegt. Die reservierten Objekte können innerhalb einer ***Session*** verarbeitet werden. Nach Beendigung dieser werden alle während der *Session* reservierten Objekte freigegeben. In einer einzelnen Client – Anwendung kann immer nur eine *Session* verwaltet werden. Erste nach Beendigung einer *Session* kann eine neue gestartet werden.

Die Verwaltung der *Sessions* erfolgt durch die Client – Umgebung. Um diese Funktionalität den Client – Anwendungen zugänglich zu machen, müssen diesen die Methoden `startSession` und `endSession` bereitgestellt werden. Dies geschieht auf die gleiche Weise wie bei der `getRemoteHome` – Methode. Die Funktionsweise der `startSession` – und `endSession` – Methoden wird im Folgendem beschrieben.

Start Session

Zum Start einer *Session* ruft die Client – Anwendung die `startSession` – Methode auf. Alternativ können die *Sessions* beim Start der Client – Anwendung oder beim ersten Zugriff auf die Komponenten – Verwaltung durch die Client – Umgebung gestartet werden.

Verwaltung

Nach Start einer *Session* müssen alle darauf folgenden Reservierungen innerhalb der *Session* verwaltet werden. Diese von der Client – Umgebung zusätzlich zu realisierende Funktionalität wird durch das `boolean` – Attribut `session` signalisiert, indem es den Wert `true` annimmt.

Bei jeder Anforderung eines Objektes im Server (Aufruf der `reserveComponent` – Methode), wird die Referenz auf die Wurzel des reservierten Netzes in einer `components` – Menge verwaltet.

Ende Session

Um eine *Session* zu beenden muss die `endSession` – Methode ausgeführt werden. Erfolgt dies nicht durch den Anwendungsentwickler, kann der Aufruf zum

Beispiel am Ende der `main` – Methode von einem *Aspekt* eingeführt werden.

Die Client – Umgebung setzt die `endSession` – Methode um, indem sie alle innerhalb der *Session* reservierten Komponenten – Netze freigibt. Dazu führt sie auf jedem Eintrag in der `components` – Menge die `release` – Methode aus. Nach Abschluss dieser Freigabe, werden die Referenzen verworfen und das Attribut `session` auf `false` gesetzt.

Probleme bei mehrfacher Ausführung der `create` – Methode: Da dem Client bereits bei der Suche nach einem Objekt, das einer vorgegebenen *Home* – Schnittstelle genügt (`getRemoteHome` – Methode), eine konkrete Instanz einer Implementierung zugewiesen werden muss, erfolgt die Reservierung schon zu diesem Zeitpunkt. Auf das reservierte Objekt kann die `create` – Methode und mehrere Aufrufe der Geschäftsmethoden ausgeführt werden. Die Reservierung bleibt erhalten, bis das Objekt freigegeben wird (`release` – Methode), oder der Client die *Session* beendet. Hierbei muss beachtet werden, dass nach Ausführung der ersten `create` – Methode immer noch auf das Objekt verwiesen wird, das beim Aufruf der `getRemoteHome` – Methode zurückgegeben wurde. Wird das Komponenten – Netz freigegeben, wird die Referenz gelöscht, d. h. bei einem erneuten Aufruf der `create` – Methode muss das Netz vollkommen neu erstellt werden. Da in dem Fall die Information, welche Schnittstelle angesprochen wird, nicht mehr verfügbar ist, kann die Anfrage nicht an die `reserveComponent` – Methode weitergeleitet werden. Es kann demzufolge nur eine Fehlermeldung ausgelöst werden. Um diese Situation zu vermeiden, muss durch den Anwendungsentwickler beachtet werden, dass er sich immer eine neue Referenz durch Aufruf der `getRemoteHome` – Methode reserviert.

4.4.2 Zusammenfassung

Es wurde eine Laufzeitumgebung zur Verarbeitung von *QoS* – Eigenschaften für Client – Anwendungen geschaffen und beschrieben. Diese greift auf die Funktionen einer Server – Anwendung zu, für die ebenfalls *QoS* – Eigenschaften verwaltet werden.

4.5 Modulare Gestaltung des Containers

Eine Zielvorgabe für den Entwurf war die modulare Gestaltung der Komponenten – Verwaltung und eine dadurch zu erreichende hohen Austauschbarkeit der Funktionsbereiche.

Die entwickelte Laufzeitumgebung unterteilt ihre Funktionalität in folgende Einheiten:

- *QoS* – Repository
- Implementierungs – Verwaltung

  - Verwaltung der Informationen zu den Komponenten
  - Instanzen – Verwaltung

- Vertragsmanager

Für die Realisierung dieser Aufgaben war die Verwendung von *Aspektorientierter Programmierung* nicht zwingend notwendig. Es wurde die klassischen Konzepte der Objektorientierten Programmierung angewandt.

Obwohl *QoS* – Repository und Implementierungs – Verwaltung eng zusammenarbeiten, ist es gelungen ihre Funktionsbereiche klar zu trennen. Über die öffentlichen Methoden der Klassen `QoS_Profile` und `QoS_SimpleProfile` des *QoS* – Repository sowie `Implementation` und `InstanceManager` der Implementierungs – Verwaltung kommunizieren sie miteinander. Um die Kapselung der Funktionalität zu Verstärken werden Schnittstellen definiert, die von den Klassen zu implementieren sind.

Da der Vertragsmanager die Informationen der anderen Einheiten auswertet, ist seine Funktionalität direkt in ihnen integriert. Dies erscheint sinnvoll, da zum Beispiel bei Änderung der Verwaltung der Parameter, der Vergleich ebenfalls angepasst werden muss.

Ist ein eigenes Modul für den Vertragsmanager gefordert, kann dies durch einen *Aspekt* (`ContractManager`) realisiert werden. Über diesen werden alle Funktionen zur Vertragsaushandlung und – durchsetzung in den Klassen der anderen Module eingeführt. Nachfolgend sind die zu manipulierenden Klassen mit den eingefügten Methoden aufgelistet:

- `Implementation:`
  - `getContract(String identifier,Vector uses)`
  - `getAllContracts(String identifier,Vector uses)`
- `QoS_SimpleProfile:`
  - `getContract(String identifier,Vector uses)`
- `QoS_Statement, Constraint, ...:`
  - `isEqualOrRestrictive(Comparison comp)`
- `Parameter, ...:`
  - `compareTo(Parameter parameter)`
- `Discriminator, ...:`
  - `compareTo(Discriminator descr)`
- `SessionBean:`
  - `compNet`
  - `getLocalHome(String identifier)`

Um die Funktionalität der Komponenten – Verwaltung zu erweitern, kann wie bereits beschrieben das *Interceptor* – Muster in Kombination mit Aspekten angewandt werden (S. 73 f.). Zur Änderung der bereits entworfenen Funktionseinheiten der Komponenten – Verwaltung können die Module vollständig ausgetauscht werden. Die neuen Einheiten müssen aber weiterhin die in den Abschnitten zuvor beschriebenen Methoden umsetzen.

Zum Abschluss der Betrachtung der Implementierungsdetails wird die Implementierung des Prototypen vorgestellt.

4.6 Implementierung des Prototypen

Funktionalität

Für die Implementierung des Prototypen wurde die Kernfunktionalität der Komponenten – Verwaltung umgesetzt.[1] Zu den realisierten Funktionsbereichen gehören:

- Einlesen der *QoS* – Eigenschaften und Abbildung auf Objektstruktur,
- Überprüfung der Eignung eines einfachen Profils zur Erfüllung der *QoS* – Anforderungen einer aufrufenden Implementierung,
- Reservierung der Ressourcen und Erstellung von Instanzen (ohne Verwendung der reservierten Ressourcen!)

Zusätzlich wurde eine *Dummy* – Implementierung des *QoS* – Managers erstellt, um die Reservierungsanfragen bearbeiten zu können.

Testanwendung

Da bei der Entwicklung der Testanwendung auf die Funktionen der fehlenden Einheiten nicht zurückgegriffen werden konnte, weist die Implementierung folgende Einschränkungen auf:

- Vereinfachte Implementierung des `QoS_AppParser`: Dieser liest keine *XML* – Datei zu den Beziehungen aus, sondern speichert direkt die Information zu `StockQuotationDistributor_SL`, `SubscriptionManager_SL` und `ClientController_SL`.
- Die Suche nach einem einfachen Profil wird abgebrochen, sobald ein geeignetes gefunden wird.
- Es werden Ressourcen angefordert, welche aber bei der Erstellung der Instanz nicht ausgewertet werden.
- Obwohl *Stateless Session Beans* umgesetzt werden, wird der Verweis auf die Instanz und nicht auf das Proxy – Objekt zurückgegeben.

[1] Der Quellcode ist auf der beigefügten CD zu finden.

Start Anwendung

Nachdem die Ersatzimplementierung für die Ressourcen – Verwaltung gestartet wurde, kann die Komponenten – Verwaltung (`ContainerImpl`) ausgeführt werden. Diese liest die *QoS* – Eigenschaften ein und startet die Test – Anwendung (`test` in `ContainerImpl`).

Funktionalität

Bei Ausführung der Methode werden die Verträge zwischen `ClientController_SL` und `SubscriptionManager_SL` ausgehandelt und nach erfolgreichem Abschluss versucht Instanzen zu erstellen. Dazu müssen die eingelesenen Ressourcenanforderungen ausgewertet und eine Anfrage an die Ressourcen – Verwaltung gesandt werden. Können die Ressourcen reserviert werden, wird eine Instanz erstellt, andernfalls wird das Scheitern gemeldet.

4.7 Zusammenfassung

Bei der Gestaltung der Komponenten – Verwaltung wird auf die klassischen Konzepte der objektorientierten Programmierung zurückgegriffen. Die in *Java* implementierte Laufzeitumgebung nutzt zusätzlich die Vorteile von *AOP* und *Interceptoren* zur Integration der Komponenten und zur späteren Erweiterung der Funktionalität.

Das Einlesen der *QoS* – Eigenschaften und den Beziehungen erfolgt durch die Parser `QoS_Parser` und `QoS_AppParser`.

Zur Bearbeitung von Clientanfragen werden Methoden für die Prüfung von Profilen, für den Zugriff auf die Ressourcen – Verwaltung sowie zur Reservierung und Freigabe von Instanzen angeboten. Die Durchsetzung der Verträge zwischen kommunizierenden Instanzen wird garantiert, indem von der Laufzeitumgebung die Methode zum Zugriff auf abhängige Instanzen zur Verfügung gestellt wird.

Bei der Gestaltung der Server – Anwendungen ist das Einhalten von Konventionen erforderlich.

Desweiteren muss eine Laufzeitumgebung für Client – Anwendungen gestaltet werden.

Der im Laufe dieser Arbeit entwickelte Prototyp setzt die Kernfunktionalität der Komponenten – Verwaltung mit ‘Einlesen’, ‘Profilprüfung’ und ‘Ressourcenreservierung’ um und ist auf der beigefügten CD zu finden.

Kapitel 5

Zusammenfassung und Ausblick

Ziel der Arbeit war der Entwurf einer Laufzeitumgebung für Komponenten mit *QoS*–Eigenschaften. Mit Hilfe von *CQML*$^+$ wurden die benötigten Betriebsmittel und die Anforderungen, die eine Komponente an andere stellt, spezifiziert. Die Verwaltung der Komponenten und ihrer *QoS*–Eigenschaften, sowie die Gewährleistung der Erfüllung ihrer Anforderungen sind die Aufgaben der zu entwickelnden Komponenten–Verwaltung. Zur Reservierung und Verwendung von Betriebsmitteln muss die Komponenten–Verwaltung auf eine Ressourcen–Verwaltung zugreifen. Das Ergebnis der Arbeit wird in diesem Kapitel vorgestellt.

Nach einer Zusammenfassung und Bewertung der Resultate werden Probleme, die bei der Bearbeitung der Aufgabenstellung auftraten, beschrieben. Abschließend soll auf spätere Erweiterungsmöglichkeiten bezüglich der Funktionalität hingewiesen werden, die während der Bearbeitung als sinnvolle Ergänzung erkannt wurden.

5.1 Realisierung der Anforderungen

Funktionsbereiche

Durch Analyse der Aufgabenstellung konnten die notwendigen Funktionsbereiche der Komponenten–Verwaltung ermittelt werden: *QoS–Repository*, *Implementierungs–Verwaltung* und *Vertragsmanager* mit *Kommunikations–Proxy* und *Ressourcen–Proxy*.

QoS–Repository

QoS–Repository: Die Verwaltung der *QoS*–Eigenschaften der Komponenten erfolgt im *QoS*–Repository. Es bindet die Eigenschaften an die Implementierungen. Desweiteren stellt es Methoden zur Prüfung der Eignung von Profilen zur Verfügung. Bei diesen Untersuchungen wird festgestellt, ob die angebotenen Eigenschaften den Anforderungen der aufrufenden Komponenten–Implementierung entsprechen.

Implementierungs – Verwaltung

Implementierungs – Verwaltung: Zu den Aufgaben der Implementierungs – Verwaltung gehört die Informationsverwaltung der Komponenten – Implementierungen, einschließlich ihrer Beziehungen untereinander. Jeder Implementierung können mehrere Profile zugewiesen werden, die jeweils unterschiedliche *QoS* – Eigenschaften repräsentieren. Aufgrund der verschiedenen Ressourcenanforderungen werden zu jedem Profil eigenständige Instanzen – Verwaltungen eingeführt. Zu deren Aufgaben gehört die Organisation des Zwischenspeicherns von Instanzen.

Vertragsmanager

Vertragsmanager: Für die Durchsetzung der *QoS* – Anforderungen ist der Vertragsmanager zuständig. Zu seinen Funktionen gehört die Auswertung der Beziehungen und *QoS* – Eigenschaften und die daraus resultierende Aushandlung der Verträge zwischen kommunizierenden Implementierungen und mit der Ressourcen – Verwaltung. Nach Abschluss aller Verträge ist es ebenfalls die Aufgabe des Vertragsmanagers, für deren Einhaltung zu sorgen. Seine Funktionsbereiche Kommunikations – Proxy und Ressourcen – Proxy setzen die Verträge zwischen kommunizierenden Instanzen und mit der Ressourcen – Verwaltung durch.

Gestaltung

Zur Gestaltung der Komponenten – Verwaltung wird auf die klassischen Konzepte der Objektorientierten Programmierung zurückgegriffen und zusätzlich die Vorteile der *Aspektorientierten Programmierung* zur Integration der Komponenten genutzt. Für spätere Erweiterungen werden unter Verwendung des *Interceptor* – Musters und *AOP* Möglichkeiten der Integration dieser Funktionalität geschaffen.

Um die Anwendungen mit Hilfe von *Aspekten* integrieren zu können, müssen Konventionen eingehalten werden.

Auf Basis der gewonnenen Erkenntnisse wurde ein Prototyp implementiert, der die Kernfunktionalität umsetzt. Bei der Erprobung bestätigten sich die theoretischen Ansätze. Eine Erweiterung des Prototyps sollte alle in der Arbeit beschriebenen Entwurfkonzepte berücksichtigen und umsetzen.

5.2 Auswertung der Ergebnisse

Im Laufe der Arbeit wurden die Funktionsbreiche der Komponenten – Verwaltung ermittelt und detailliert beschrieben. Es erfolgte eine Untersuchung ihrer Aufgaben und der Realisierungsmöglichkeiten. Nach Erarbeitung verschiedener Lösungsalternativen wurden die getroffenen Entwurfsentscheidungen begründet.

Die entworfene Laufzeitumgebung setzt die gesamte Funktionalität zur Verarbeitung der *QoS* – Eigenschaften um. Zur Veranschaulichung der Konzepte wurde ein Prototyp der Komponenten – Verwaltung implementiert.

Probleme $CQML^+$

Während der Arbeit wurden folgende Ungenauigkeiten in der $CQML^+$ – Spezifikation festgestellt:

- Keine Erklärung zur Verwendung einiger Vergleichsoperatoren mit den statistischen Werten *Minimum*, *Maximum* und *Range*, z. B.:

  - der Unterschied zwischen $\geq$ `Minimum` und `= Minimum` ist unklar,
  - keine Bedeutung für $\geq$ `Range` definiert,
- kein Vergleichsoperator für *Limit* definiert und
- ungenaue Spezifikation der *Diskriminatoren* für den Vergleich von zwei bewerteten Charakteristiken.

Es mussten daher Entscheidungen zur Verarbeitung dieser Sonderfälle getroffen werden (S. 47 ff.).

Zusätzlich wurden Möglichkeiten zur Erweiterung der entwickelten Komponenten – Verwaltung herausgestellt, und im folgenden Abschnitt zusammengefasst.

5.3 Erweiterungen

Während der Analyse und Umsetzung der Funktionalität der Komponenten – Verwaltung wurden Erweiterungsmöglichkeiten ermittelt. Diese sollen nachfolgend beschrieben und gegebenenfalls Ansätze zur Umsetzung vorgestellt werden. Einzelne Funktionsbereiche konnten aber aufgrund ihrer Komplexität im Rahmen dieser Arbeit nicht umgesetzt werden. Sie werden als Hinweis für zukünftige Erweiterungen abschließend zusammengefasst.

Ressourcen für Laufzeitumgebung

Ressourcenanforderungen der Komponenten – Verwaltung: Eine wichtige noch umzusetzende Aufgabe ist die Reservierung der Ressourcen, die die Komponenten – Verwaltung zur Erfüllung ihrer Aufgaben benötigt. Es ist zu ermitteln, welche und wie viele Betriebsmittel zur Verwaltung der Komponenten und ihrer *QoS* – Eigenschaften sowie für Vertragsaushandlung und – durchsetzung zur Verfügung stehen müssen.

Neben den Anforderungen des Standardbetriebes müssen zusätzlich Ressourcen zur Verfügung stehen, um zum Beispiel erforderliche Netzerweiterungen schnell bearbeiten zu können.

Erweiterung Ressourcen – Verwaltung

Erweiterung der Ressourcen – Verwaltung: Momentan sind die Beschreibungsmöglichkeiten der Ressourcenanforderungen in der *CQML*$^+$ – Spezifikation umfangreicher, als sie in der Ressourcen – Verwaltung verarbeitet werden können (S. 21). Ein Ausbau dieser ist anzustreben.

Ressourcen – Proxy

Realisierung des Ressourcen – Proxy: Im Zeitrahmen dieser Arbeit war es nicht möglich eine ausführliche Analyse der Möglichkeiten des Zugriffs auf die *JVM* durchzuführen, um so die Verträge mit der Ressourcen – Verwaltung durchsetzen zu können. Diese Aufgaben müssen in einer Erweiterung der Laufzeitumgbung realisiert werden.

Einhaltung der Eigenschaften

Überwachung der Einhaltung der QoS – Eigenschaften: Nachdem die Verträge ausgehandelt wurden, und die Anfragen erfolgen, ist nicht nur für die Durchsetzung der Verträge zu sorgen, sondern auch die Einhaltung der zugesicherten *QoS* – Eigenschaften ist zu überwachen. Da dieser Funktionsbereich Informationen der Charakteristiken und Verträge auswerten muss, ist es sinnvoll ihn

direkt durch Eingriff in die Implementierung der Komponenten – Verwaltung einzufügen. Andernfalls müsste bei Anwendung des *Interceptor* – Musters ein sehr umfangreiches `Context` – Objekt erstellt werden, so dass alle relevanten Informationen zur Verfügung stehen.

Erweiterung bereits umgesetzter Konzepte: Es folgt eine Auflistung von Einschränkungen, die aufgrund ihrer Komplexität für diese Arbeit getroffen wurden und in einer erweiterten Version der Komponenten – Verwaltung umgesetzt werden sollten:

- Untersuchung von Charakteristiken erfolgt nur auf einer Ableitungsebene (S. 46),
- Einbeziehung nur von statistischen Werten wie *Minimum*, *Maximum* und *Range* (S. 50),
- keine Wertefunktion über Profile (Ermittlung bester Profile nicht möglich)
- Annahme von ausfallsicheren Komponenten (S. 31 f.),
- Vergleich der Parameter bewerteter Charakteristiken ermöglicht keine Auswertung der Beeinflussung der Flüsse durch die Übertragungsstrecke (S. 46 f.),
- keine Auswertung der Operationen auf *Event – Sequenzen* (S. 46) und
- Annahme, dass sich Charakteristiken nur auf eine Komponente beziehen (S. 44 f.).

Änderungsvorschläge: Aufgrund der Erfahrungen konnten einige Schwachstellen der ursprünglichen Konzepte ermittelt werden, für die nachfolgenden Änderungsvorschläge vorgestellt werden.

Home – Schnittstelle

Änderungsvorschlag für das Komponentenmodell: In der Komponenten – Verwaltung wird den aufrufenden Komponenten bereits bei der Anforderung einer Referenz auf ein Objekt, das der *Home* – Schnittstelle genügt, die endgültig zu verwendende Instanz zugewiesen. Die Aufgabe der `create` – Methode besteht nur noch darin, die Attributwerte der *Stateful Session Beans* zu setzen und nicht die Erstellung einer neuen Instanz wie in *EJB*. Da zusätzliche Probleme bei mehrfachem Aufruf der `create` – Methode auftreten (Netz – Erweiterung), sollte die Trennung der Schnittstellen aufgehoben werden. Für jede benötigte Instanz muss eine neue Referenz angefordert werden (`getLocalHome`).

Reduzierung der Objektabbildung der QoS – Eigenschaften: Auf die Verwaltung der *QoS* – Anweisungen kann verzichtet werden, da der Vergleich nur auf Ebene der Charakteristiken erfolgt. Schon beim Einlesen werden die Parameter und Werte ausgewertet und den Charakteristiken zugewiesen.

Bei der Umsetzung dieser Reduzierung muss zusätzlich beachtet werden, dass die geschachtelten *Konjunktiven Normalformen* — der Anweisungen in den einfachen Profilen und der bewerteten Charakteristiken in den einfachen Anweisungen — beim Einlesen aufzulösen sind (S. 21 ff.).

Anhang A

Aufbau des Proxy - Objektes bei Stateless Session Beans

Bei der Reservierung von *Stateless Session Beans* wird nicht auf die *Session Bean* – Instanz sondern auf den Instanzen – Manager verwiesen. Dabei ergeben sich Probleme, da dieser nicht der Spezifikation entspricht. Es wird demzufolge für den Zugriff auf `InstanceManager` ein Objekt (*Proxy*) benötigt, dass der Komponenten – Spezifikation entspricht.

Die Erstellung des *Proxy* soll nicht die Aufgabe des Anwendungsentwicklers sein, da er auf diese Weise die Funktionalität des Containers unkontrolliert beeinflussen könnte. Aus Sicherheitsgründen wird dies ausgeschlossen, und die *Proxy* – Klasse muss von der Laufzeitumgebung generiert werden. Eine Möglichkeit bietet `XSLT` [32][31].

Die notwendigen Informationen werden dem Beziehungs – Descriptor entnommen:

- Name der Spezifikation (einschließlich Package – Struktur)
- Auflistung der Methoden, mit
  - Methodenname
  - Datentyp des Rückgabewertes
  - Auflistung der Parameter, mit
    - Parametername
    - Datentyp des Parameters

Aus einer Beispielanwendung

- examples.ExampleSpecification
  - doThis
  - void
    - test
    - String
  - doThat
  - String
    - test
    - String

wird folgende Klasse generiert:

```
package examples;

import qos.server.instancepool.ComponentProxy;
import qos.server.instancepool.InstanceManager;

public class ExampleSpecificationProxy extends ComponentProxy
        implements ExampleSpecification {

  private ExampleSpecification impl;

  public ExampleSpecificationProxy(InstanceManager instManager)
           throws java.rmi.RemoteException {
    super(instManager);
  }

  public void doThis(String test)
           throws java.rmi.RemoteException {
    impl = (ExampleSpecification)getInstManager.getInstance();
    impl.doThis(test);
    impl.release();
  }

  public String doThat(String test)
           throws java.rmi.RemoteException {
    impl = (ExampleSpecification)getInstManager.getInstance();
    String result = impl.doThat(test);
    impl.release();
    return result;
  }

}
```

Die aus dem Beziehungs – Descriptor entnommenen Informationen sind im Folgenden durch Schrägstellung gekennzeichnet. Die restlichen Eigenschaften werden für alle *Proxy* – Objekte einheitlich generiert.

- Name der Klasse: *`SpecificationName`*`Proxy`
- Die Klasse wird von `ComponentProxy` (Abb. A.1) abgeleitet und implementiert die beschriebene Spezifikation.
- Sie erbt das Attribut vom Typ `InstanceManager` (`instManager`).
- In allen Geschäftsmethoden wird vom `InstanceManager` eine verfügbare Instanz angefordert und anschließend der Aufruf daran weitergeleitet. Nach Beendigung der Methode muss die Reservierung wieder freigegeben werden. Dazu wird die `release` – Methode aufgerufen, die das Attribut `state` auf `0` setzt.
- Der Aufruf der `release` – Methode wird in `ComponentProxy` an die `release` – Methode in `InstanceManager` weitergeleitet, der die Anzahl der Reservierungen reduziert.
- Da in *Stateless Session Beans* keine Verarbeitungsschritte in der *create* Methode ausgeführt werden, beinhaltet die in `ComponentProxy` implementierte Methode nur:
  `return this;`

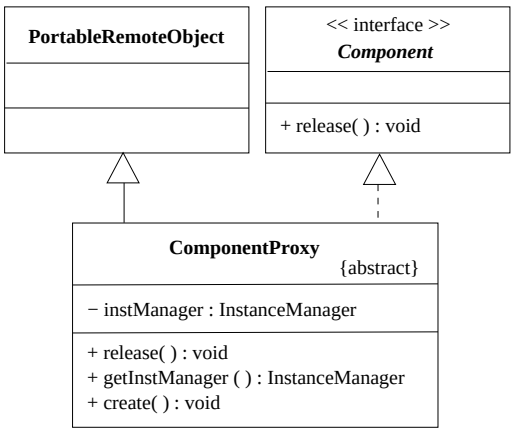


Abbildung A.1: *Proxy* – Objekt (`ComponentProxy`)

Anhang B

Dispatcher der Komponenten – Verwaltung

Ziel ist es, eine existierende Komponenten – Verwaltung um Funktionalität zu erweitern, ohne in die existierenden Funktionsbereiche eingreifen zu müssen beziehungsweise zu dürfen. Eine Möglichkeit zur Lösung diese Aufgabe bietet das *Interceptor* – Muster [25] (Abschnitt 4.1.1). In diesem Fall wird an definierten Punkten im Programmverlauf die zusätzliche Funktionalität, die in *Interceptoren* umgesetzt wurde, eingefügt. Ein *Interceptor* meldet sich bei einem *Dispatcher* an, der ihn beim Eintreffen des zugehörigen Ereignisses informiert. Zur Überwachung der Ereignisse wird ein *Aspekt* definiert.

Bei der Definition der Eingriffspunkte muss sichergestellt werden, dass alle möglichen Ereignisse berücksichtigt wurden. Die für die Komponenten – Verwaltung ermittelten *Dispatcher* sind nachfolgend aufgelistet. Werden weitere benötigt, muss der *Aspekt* angepasst und die *Dispatcher* hinzugefügt werden.

Die beschriebenen Ereignisse treten ein, wenn:

- eine Instanz angefordert (`Instance`),
- eine Geschäftsmethoden ausgeführt (`Business`) oder
- eine Reservierung aufgehoben (`Release`) wird.

Auflistung der möglichen *Dispatcher*:

- in Client – Umgebung:
  - `InstanceRequestDispatcher`
  - `BusinessRequestDispatcher`
  - `ReleaseRequestDispatcher`
  - `InstanceResponseDispatcher`
  - `BusinessResponseDispatcher`

  - `ReleaseResponseDispatcher`

- in Server – Umgebung:

  - `LocalStatelessInstanceRequestDispatcher`
  - `LocalStatelessBusinessRequestDispatcher`
  - `LocalStatelessReleaseRequestDispatcher`
  - `LocalStatelessInstanceResponseDispatcher`
  - `LocalStatelessBusinessResponseDispatcher`
  - `LocalStatelessReleaseResponseDispatcher`
  - `RemoteStatelessInstanceResponseDispatcher`
  - `RemoteStatelessBusinessResponseDispatcher`
  - `RemoteStatelessReleaseResponseDispatcher`

  - `LocalStatefulInstanceRequestDispatcher`
  - `LocalStatefulBusinessRequestDispatcher`
  - `LocalStatefulReleaseRequestDispatcher`
  - `LocalStatefulInstanceResponseDispatcher`
  - `LocalStatefulBusinessResponseDispatcher`
  - `LocalStatefulReleaseResponseDispatcher`
  - `RemoteStatefulInstanceResponseDispatcher`
  - `RemoteStatefulBusinessResponseDispatcher`
  - `RemoteStatefulReleaseResponseDispatcher`

Anhang C

Inhalt der CD - ROM

Glossar

Advice	Konstrukt der Aspektorientierten Programmierung. Innerhalb des Advices ist der zusätzlich auszuführende Code implementiert.
AOP	Kurz für *Aspektorientierte Programmierung*. Dient der Umsetzung der Crosscutting – Eigenschaft. Sie stellt eine Erweiterung herkömmlicher Programmiersprachen dar.
AspectJ	Von Xerox Parc entwickelte Erweiterung des Java Compilers, welche AOP umsetzt. Nach dem Aspect Weaving wird der Standard Java Compiler ausgeführt.
Aspekt	Konstrukt der Aspektorientierten Programmierung. Der Aspekt kapselt die Crosscutting – Funktionalität.
COMQUAD	Kurz für *COMponents with QUantitative properties and ADaptivity*. Mehr Informationen sind der Homepage *www.comquad.org* zu entnehmen.
CORBA	Kurz für *Common Object Request Broker Architecture*. Standard, der die Kommunikation zwischen Objekten und Programmen regelt.
CORBA IDL	CORBA *Interface Definition Language*. Deklarative Sprache zur Beschreibung von Schnittstellen für CORBA Objekte. Sie ermöglicht die Definition einer sprachunabhängigen Schnittstellen Spezifikation.
CQML	Kurz für *Component Quality Modelling Language*. Deklarative Sprache zur Spezifikation der *QoS* – Eigenschaften von Komponenten. Basiert auf *QML*.
Dispatcher	Konstrukt des Interceptor – Musters. Er registriert Interceptoren und informiert sie über das Eintreten von Ereignissen.

EJB	Kurz für *Enterprise JavaBeans*. Architektur für die Entwicklung und den Einsatz komponentenbasierter verteilter Anwendungen.
EJB Container	Gesamtheit der EJB – Funktionalität.
Entity Bean	Konstrukt der EJB – Spezifikation. Objektabbildung eines persistenten Datenbankeintrages.
DROPS	Kurz für *Dresden Real – Time Operating System*. Projekt zur Entwicklung eines verteilten Echtzeit – Betriebssystems zur Unterstützung von Anwendungen mit QoS – Anforderungen.
Interceptor	Konstrukt des Interceptor – Musters. Setzt die zu integrierende Funktionalität um.
Join Point	Konstrukt der Aspektorientierten Programmierung. Der Join Point ist ein fester Punkt in der Programmausführung. An diesen Stellen wird die Crosscutting – Funktionalität in die Klassen eingefügt, indem die Pointcut auf die Join Points zugreift.
JNI	Kurz für *Java Native Interface*. Schnittstelle zum Verbinden von Java – Code mit Code anderer Programmiersprachen.
Komponenten – Verwaltung	Laufzeitumgebung für Komponenten und Verarbeitung von QoS – Eigenschaften.
Naming-Server	Server, für Zuordnung von Namen und Internetadressen.
Pointcut	Konstrukt der Aspektorientierten Programmierung. Innerhalb des Pointcuts werden die Join Points definiert, an denen die Crosscutting – Funktionalität in ein Programm eingefügt wird.
QML	Kurz *Quality Modelling Language*. Deklarative Sprache zur Spezifikation von *QoS* – Eigenschaften.
QoS	Kurz für *Quality of Service*. Beschreibt Dienstgütemerkmale die überwacht und deren Einhaltung garantiert werden soll.

QoS – Eigenschaften	Spezifikation der nichtfunktionalen Eigenschaften. Strukturierung: - *QoS* – Anforderungen: von anderen Komponenten geforderte Eigenschaften; - *QoS* – Angebote: von einer Komponente angebotene Eigenschaften und - Ressourcenanforderungen: Anforderungen an Betriebsmitteln.
Ressourcen – Verwaltung	Verwaltung der Betriebsmittel. Definierte Schnittstelle zur Reservierung und zum Zugriff auf Ressourcen.
RMI IIOP	Kurz für *Remote Method Invocation*. Sie ist mit der CORBA – Technologie kombiniert und entspricht daher dem *Internet Inter-Orb Protocol*. RMI erlaubt einer Anwendung mit Objekten, enthalten in einem Programm auf einer entfernten Maschine, zu kommunizieren.
Stateful Session Bean	Konstrukt der EJB – Spezifikation. Realisierung von Geschäftsprozessen mit Verwaltung eines Konversationszustandes.
Stateless Session Bean	Konstrukt der EJB – Spezifikation. Realisierung von Geschäftsprozessen ohne Verwaltung eines Konversationszustandes.
Vertrag	Bei der Verarbeitung von *QoS* – Eigenschaften werden Verträge über Zusicherungen von nichtfunktionalen Eigenschaften abgeschlossen: - zwischen Komponenten – Implementierungen und - mit der Ressourcen – Verwaltung.
XML	Kurz für *Extensible Markup Language*. Standardisierte Metasprache zum Definieren von Dokumenttypen.
XSLT	Kurz für *eXtensible Stylesheet Language for Transformations*. Sprache zur Transformation von XML Dokumenten zwischen verschiedenen strukturellen Modellen.

Literaturverzeichnis

[1] Special issue on aspect-oriented programming. *ACM Press New York, USA*, pages 5–168, Oktober 2001.

[2] Sun Microsystems, EJB-Spezifikation, Version 2.0, Final Release, August 2001.

[3] AspectJ, Version 1.0.5, 28. Juni 2002. http://www.eclipse.org/aspectj/.

[4] COMQUAD an der Technischen Universität Dresden, April 2003. http://www.comquad.org.

[5] Java Native Interface – Documentation, März 2003. http://java.sun.com/products/jdk/1.2/docs/guide/jni/.

[6] Jan Øyvind Aagedal. *Quality of Servic Support in Development of Distributed Systems*. Department of Informatics, University of Oslo, M 2001.

[7] Helmut Balzert. *Lehrbuch der Softwaretechnik: Software – Entwicklung*. Spektrum Akademischer Verlag GmbH, 1996.

[8] Antoine Beugnard, Jean-Marc Jézéquel, Noël Plouzeau, and Damien Watkins. Making Components Contract Aware. 1999.

[9] Lutz Dominick. 'AOP: Aspektorientierte Programmierung' im Überblick. *JavaSPEKTRUM*, (4):43ff, April 2000.

[10] Brit Engel. *Aspektorientierte Programmierung und EJB im Vergleich*. Technische Universität Dresden, Fakultät Informatik, Institut für Software- und Multimediatechnik, 12. Juli 2002.

[11] Richard Helm Ralph Johnson John Vlissides Erich Gamma. *Entwurfsmuster Elemente wiederverwendbarer Software*. Addison-Wesley Verlag, München, 1996.

[12] James Gosling and Henry McGilton. The Java Language Environment, Mai 1996. http://java.sun.com/docs/white/langenv/Security.doc1.html.

[13] Object Management Group. CORBA 3.0 new component chapters, OMG Document, Oktober 1999. http:// www.omg.org/cgi-bin/doc?ptc/99-10-04.

[14] Object Management Group. Unified Modelling Language Specification Version 1.4. *OMG Document*, September 2001.

[15] XML Schema Working Group. *XML Schema w3c recomendation*. http://www.w3.org/TR/xmlschema-0/, http://www.w3.org/TR/xmlschema-1/, http://www.w3.org/TR/xmlschema-2/, Mai 2001.

[16] XML Working Group. *Extensible Markup Language (XML) 1.0 W3C Recommendation*. http://www.w3.org/TR/REC-xml, 6. Oktober 2000.

[17] H. Härtig, R. Baumgartl, M. Borriss, Cl.-J. Hamann, M. Hohmuth, F. Mehnert, L. Reuther, S. Schönberg, and J. Wolter. *DROPS: OS support for distributed multimedia applications*. In Proc. 8th ACM SIGOPS European Workshop: Support for Composing Distributed Applications (Sintra, Portugal, Sept. 1998), Sintra, Portugal, September 1998.

[18] Mohamed Mancona Kandé, Joerg Kienzle, and Alfred Strohmeier. *From AOP to UML – A Bottom - Up Approach*. Mai 2002. http://lglwww.epfl.ch/workshops/aosd-uml/Allsubs/kande.pdf.

[19] Sheng Liang. *The Java Native Interface*. Addison-Wesley Verlag, 1999.

[20] Tim Lindholm and Frank Yellin. *The JavaTM Virtual Machine Specification*. April 1999.

[21] Richard Monson-Haefel. *Enterprise Java Beans*. O'Reilly & Associates, Inc., 3rd edition, September 2001.

[22] Jörg Nothnagel. *Ressourcenverwaltung in DROPS*. Technische Universität Dresden, Fakultät Informatik, Institut für Systemarchitektur, Professur Betriebssysteme, 4. Juli 2002.

[23] Peter Pfahler. Programmieren in Java, 1997/98. http://www.uni-paderborn.de/fachbereich/AG/agkastens/lehre/index.htm.

[24] Simone Röttger and Steffen Zschaler. *CQML$^+$: Enhancements to CQML*. Technische Universität Dresden, unveröffentlicht.

[25] Douglas C. Smidt, Michael Stal, Hans Rohnert, and Frank Buschmann. *Pattern-Oriented Software Architecture Volume 2 – Networked and Concurrent Objects*. John Wiley and Sons, 2000.

[26] Beth Stearns. Java Native Interface – Tutorial, Februar 2003. http://java.sun.com/docs/books/tutorial/native1.1/index.html.

[27] AspectJ Team. *The AspectJ Programming Guide*. April 2003. http://www.eclipse.org/aspectj/.

[28] Marcus Voelter. Aspect - Oriented Programming in Java, 26. Oktober 2001. http://www.voelter.de/data/articles/aop/aop.html.

[29] Dapeng Wang. Mehr als Kaffeefilter, Java Servlet-Filter in der Webapplikation. *Java Magazin Online*, Februar 2003. www.javamagazin.de/itr/online_artikel/psecom,id,291,nodeid,11.html.

[30] Ute Wappler. *CQML-Schma - Entwurfsdokumentation.* Technische Universität Dresden, Fakultät Informatik, Institut für Software- und Multimediatechnik, unveröffentlicht, 8. April 2003.

[31] Sabine Winkler and Thomas Bayer. Code Generierung mit XSLT, März 2003. http://www.oio.de/m/codegenerator/.

[32] Ralf Wirdemann. Generative Programmierung von Enterprise JavaBeans mit XML und XSLT. *OBJEKTspektrum*, Juli 2002.

Abbildungsverzeichnis

Tabellenverzeichnis

Erklärung

Ich erkläre, dass ich die vorliegende Arbeit selbstständig, unter Angabe aller Zitate und nur unter Verwendung der angegebenen Literatur und Hilfsmittel angefertigt habe.

Brit Engel Dresden, den 30.04.2003