Komplexpraktikum Softwaretechnologie – Umsetzung eines Containers für Komponenten mit nichtfunktionalen Eigenschaften

Aufgabenstellung

Inhaltsverzeichnis

Kapitel 1

Einleitung

Komponenten

In der modernen Softwareentwicklung beginnen sich komponentenbasierte Technologien immer mehr durchzusetzen. Beispiele für diese Technologien sind *CORBA* [4], *COM* [5] und *Enterprise Java Beans* (EJB, [3]). Komponentenbasierte Entwicklung bietet Vorteile insbesonder im Hinblick auf Wiederverwendung, da einzelne Komponenten nach Möglichkeit so ausgelegt sein sollen, daß sie sich in verschiedenen Kontexten wiederverwenden lassen.

Was ist eigentlich eine Komponente? Dazu gibt es in der Literatur einige Diskussion, doch gibt die Definition von Szyperski in [6] einen recht guten Einstieg:

> 'A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties.'

Container

Eine Softwarekomponente ist also ein Stück Software, das explizit angibt, welche Dienste es anbietet und welche Dienste oder Eigenschaften es in der Umgebung voraussetzt. Komponenten werden auf einem Server (auch Container genannt) installiert, welcher die „Lebensvoraussetzungen" für die Komponente schafft. Zum Beispiel sorgt er dafür, daß die Komponenten die benötigten Dienste finden, miteinander kommunizieren können oder Zugriff auf die Ressourcen des Systems erhalten. Container bieten typischerweise auch noch andere Dienste an, etwa Transaktions- und Sicherheitsdienste.

Funktionale vs. nichtfunktionale Eigenschaften

An Software werden verschiedene Anforderungen gestellt. Ein wichtiger Teil der Anforderungen sind die sogenannten *funktionalen Anforderungen*, die beschreiben, was die Software tun soll und was sie auf keinen Fall tun darf. Mindestens genau so wichtig, wenngleich schwieriger zu erfassen und durchzusetzen, sind die *nichtfunktionalen Eigenschaften*. Diese beschreiben wie schnell etwas geschehen soll, wieviel Speicher dafür verwendet werden darf oder welches Sicherheitslevel eingehalten werden muß, wenn Daten über ein Netzwerk versendet werden. Neben diesen Eigenschaften, die im allgemeinen als *Dienstgüte* oder *Quality of Service* (QoS) bezeichnet wird, beschreiben nichtfunktionale Eigenschaften aber auch die Ausfallsicherheit eines Systems, den Prozeß, der zur Erstellung

Quality of Service

der Software benutzt wurde und ähnliches. Im Rahmen dieses Komplexpraktikums schränken wir uns auf QoS ein.

Einordnung in COMQUAD

Das Projekt COMQUAD [1] an der TU Dresden und der Friedrich-Alexander-Universität Erlangen-Nürnberg beschäftigt sich mit QoS-Support für komponentenbasierte Software. Im Rahmen dieses Projekts sind einige Konzepte für einen Komponentencontainer mit QoS-Unterstützung entstanden, die im Rahmen des Komplexpraktikums prototypisch umgesetzt werden sollen.

Struktur des Dokuments

Der Rest dieses Dokuments gliedert sich wie folgt. In Kapitel 2 wird ein Überblick über den zu entwickelnden Container und eine Einordnung in das Gesamtbild gegeben. Dabei wird näher auf das Zusammenspiel mit anderen Technologien und auf Schnittstellen eingegangen. In Kapitel 3 schließlich erfolgt eine Präzisierung der eigentlichen Aufgabenstellung, das heißt, eine genauere Beschreibung der zu entwickelnden Komponentenverwaltung und ihrer Bestandteile. Dabei wird auch näher auf einige der Konzepte eingegangen, die in COMQUAD bereits dafür entwickelt wurden.

Kapitel 2

Gesamtkonzept

In diesem Kapitel soll ein Überblick über die gesamte COMQUAD-Architektur gegeben werden, in die sich der zu entwickelnde Container eingliedert. Dazu wird zunächst die grobe Schichtenarchitektur vorgestellt, woran sich eine Vorstellung des Komponentenmodells anschließt. Zum Abschluß werden die verschiedenen Spezifikationssprachen und Spezifikationsdokumente, die vom Container ausgewertet werden mssen, kurz beschrieben.

2.1 Container Architektur

Die Aufgabe besteht in der Gestaltung einer Laufzeitumgebung für Komponenten mit Ressourcenanforderungen. Dabei soll die Aushandlung der Verträge zur Erfüllung der Anforderungen transparent für die Komponenten erfolgen. Diese Aufgabe soll durch eine Komponenten – Verwaltung erfüllt werden, die Bestandteil eines Containers ist, dessen Aufbau in der Abbildung 2.1 dargestellt ist.

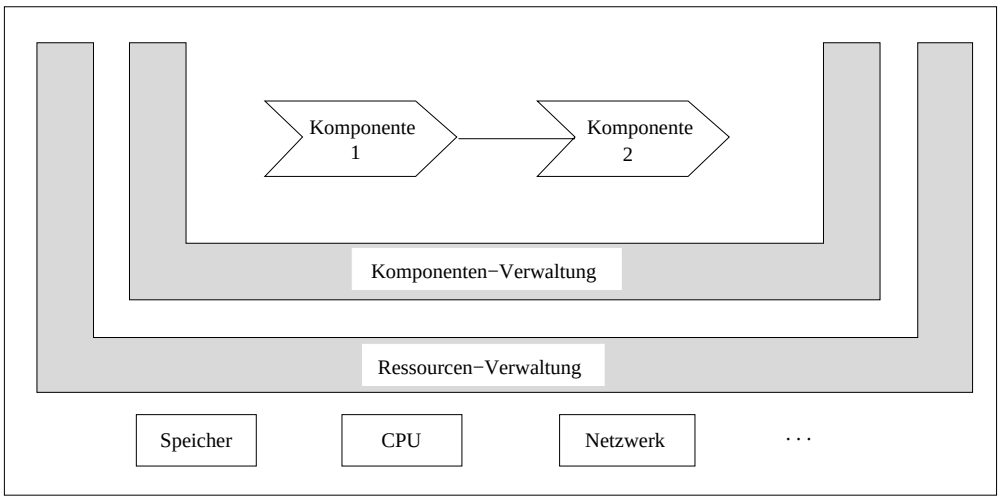

Abbildung 2.1: Container zur Verarbeitung der QoS – Anforderungen.

Der Container ist in mehrere Schichten unterteilt. Neben der Komponenten – Verwaltung, für die im nächsten Kapitel Anforderungen beschrieben werden, ist die *Ressourcen – Verwaltung* enthalten. Diese ist unabhängig von der Komponentenverwaltung und wird von dieser nur über eine definierte Schnittstelle angesprochen.

Neben der Komponentenverwaltung liegen im Container auch noch andere Dienste, so ist es geplant, einen Adaptionsdienst einzubauen, der es erlaubt, Anwendungen zur Laufzeit zu restrukturieren, um ein bestimmtes QoS-Level zu erreichen.

2.1.1 Ressourcen – Verwaltung

Die Organisation des Zugriffs auf die Ressourcen (z. B. Speicher, Netzwerk) erfolgt über die Ressourcen – Verwaltung. Auf deren Funktionalität greift die zu erstellende Komponenten – Verwaltung über eine definierte Schnittstelle zu. Die genaue Beschreibung der Schnittstelle befindet sich in einem separaten Dokument.

2.1.2 Aufgaben der Ressourcen – Verwaltung

Verwaltung verschiedener Ressourcen: Die Verwaltung einzelner Ressourcen wird von Ressourcen – Managern übernommen. Diese befinden sich in einer hierarchischen Anordnung. Die oberste Schicht der Ressourcen – Verwaltung registriert alle untergeordneten Ressourcen – Manager und kommuniziert mit der Komponenten – Verwaltung.

Verarbeitung der Ressourcen – Anforderungen: Der Ressourcen – Verwaltung wird die Ressourcenanfrage der Anwendung übergeben. Anhand dieser stellt sie verschiedene Kombinationen der zur Verfügung stehenden Ressourcen zusammen.

Zusammenstellung möglicher Ressourcen – Kombinationen: Bei der Bearbeitung einer Anfrage stehen mehrere Möglichkeiten der Zusammenstellung der Ressourcen zur Auswahl. Die Ressourcen – Verwaltung erfasst die möglichen Alternativen.

Auswahl der optimalen Alternative: Nachdem alle Ressourcen – Kombinationen zusammengestellt wurden, wählt die Ressourcen – Verwaltung die den Anforderungen der Anwendung optimal entsprechende Alternative aus.

Beschränkung zu belegender Ressourcen: 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.

Sicherstellung der Einhaltung der Zusicherungen: Die Ressourcen – Verwaltung muss sicher stellen, dass die einmal zugesicherten Ressourcen auch zur Verfügung stehen.

Änderung von QoS – Verträgen: Durch Angabe einer Prioritätsstufe sollen in Situationen zu starker Auslastung einige Zusicherungen zurückgesetzt werden können, damit wichtigere Prozesse ausgeführt werden können. Dies wird der Komponenten – Verwaltung über ein `notification`-Interface signalisiert.

Freigabe der Ressourcen: Die Komponenten – Verwaltung soll sicher stellen, dass nicht mehr benötigte Ressourcen freigegeben werden.

2.2 Komponentenmodell

Basierend auf EJB

Das verwendete Komponentenmodell basiert auf den *Stateless Session Beans* von *Enterprise Java Beans* [3]. Der Schwerpunkt liegt jedoch auf der Bereitstellung von Quality of Service (QoS). Daher wurden einige Anpassungen vorgenommen:

Vereinfachungen

Das *Session Bean* Konzept wurde vereinfacht. Es wurden Sicherheits-, Transaktions-, und Persistenzdienste entfernt. Weiterhin wird nur die Grundfunktionalität von *Stateless Session Beans* unterstützt. Es werden zunächst weder *Stateful Session Beans* noch *Entity Beans* umgesetzt.

Erweiterungen

Zusätzlich zu den Kommunikationsmechanismen der *Session Beans* wurde eine Unterstützung zur strombasierten Kommunikation geschaffen. Diese ist in einem separaten Dokument beschrieben.

Weiterhin müssen Komponenten zusätzlich zur Spezifikation der angebotenen Schnittstellen auch die Schnittstellen spezifizieren, die sie selbst benutzen. In einem weiteren Spezifikationsdokument werden diese Schnittstellen miteinander verknüpft.

Begriffe

Zum Verständnis der in diesem Text im Zusammenhang mit Komponenten verwendeten Begriffe werden die wesentlichsten in der folgenden Tabelle 2.1 kurz definiert.

Begriff	**weitere Bezeichnung**	**Beschreibung**
Komponenten – Spezifikation	- Schnittstelle	Schnittstellenbeschreibung einer Gruppe von Komponenten – Implementierungen, die die gleich Semantik umsetzen. Die Implementierungen setzen die Funktionalität unterschiedlich um und bieten verschiedene QoS – Eigenschaften an.
Komponenten – Implementierung	- Implementierung - Komponenten – Realisierung - Realisierung	Konkrete Implementierung einer Komponente, die einer Schnittstelle (Komponenten – Spezifikation) genügt.
Instanz	- Objekt	Eine zur Laufzeit gebildet Instanz einer konkreten Komponenten – Implementierung.

Fortsetzung auf der nächsten Seite

Fortsetzung von vorheriger Seite		
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.
Vertrag		Bei der Erstellung der Komponenten – Netze werden zwischen kommunizierenden Komponenten – Implementierungen Verträge ausgehandelt.
Komponente		Zusammenfassung einer Komponenten – Implementierung und und ihre Spezifikation.

Tabelle 2.1: Begriffsklärung

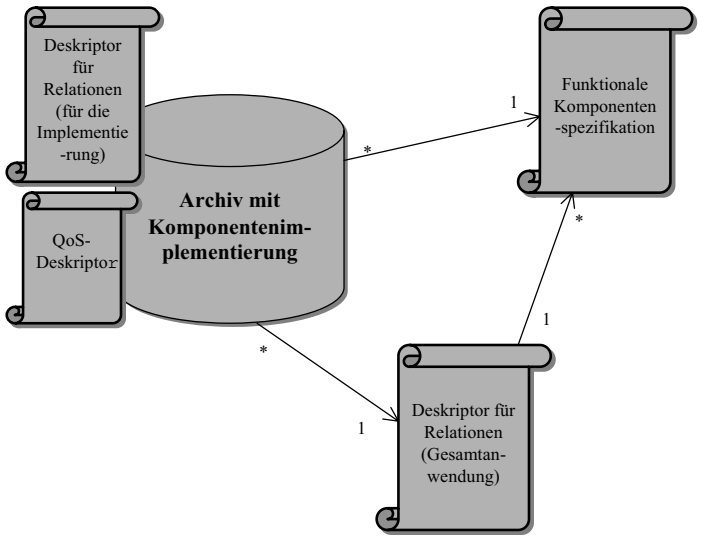

Abbildung 2.2: Verschiedene Deskriptorentypen.

Deskriptoren

Der Entwickler muß zu einer komponentenbasierten Anwendung verschiedenen Deskriptoren liefern. Diese sind in Abbildung 2.2 dargestellt. Zunächst muß für jede Komponente eine funktionale Spezifikation angegeben werden. Diese beschränkt sich im Kontext des Komplexpraktikums im wesentlichen auf die Definition von Schnittstellen. Sie liegt somit auf der gleichen Ebene wie eine Spezifikation mit IDL. Zu einer solchen Spezifikation können mehrere Implementationen bereitgestellt werden. Diese werden in Archiven verpackt, in denen sich auch zwei weitere – implementationsspezifische – Deskriptoren befinden: der QoS-Deskriptor und ein Relationendeskriptor. Der QoS-Deskriptor

beschreibt die QoS-Anforderungen und -Angebote der Implementierung. Konzeptionell basiert die verwendete Sprache auf CQML, s. Abschnitt 2.3.

Der Relationendeskriptor beschreibt, wie die Implementation mit anderen Komponenten zusammenarbeitet. Einen ähnlichen Deskriptor gibt es noch einmal für die gesamte Anwendung, dieser beschreibt auf Ebene der Spezifikationen, wie die verschiedenen Komponenten zusammen die Funktionalität der Anwendung ergeben. Die verwendete Sprache lehnt sich an die CORBA Component Assembly Descriptoren an und wird in einem separaten Dokument beschrieben.

2.3 Component Quality Modelling Language (CQML)

Die Sprache, in der der QoS-Deskriptor formuliert wird, basiert auf *CQML*$^+$, einer Erweiterung der Sprache *CQML*. Diese soll hier kurz beschrieben werden.

CQML [2] besteht aus vier Konstrukten zur Spezifikation der *QoS*–Eigenschaften:

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

In Abbildung 2.3 ist der Zusammenhang zwischen diesen 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. Die Zuordnung der Spezifikationen zu konkreten Komponenten eines Systems erfolgt in *QoS*–Profilen (*QoS Profile*). Die so definierten *QoS*–Eigenschaften werden in *QoS*–Kategorien (*QoS Category*) gruppiert.

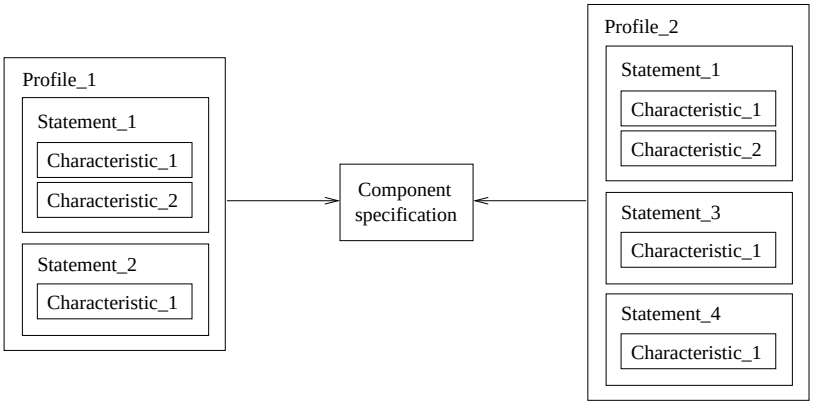

Abbildung 2.3: CQML Überblick [2]

QoS Characteristic (QoS – Charakteristiken) Die Charakteristik stellt den Grundbaustein für eine *QoS* – Spezifikation dar und ist durch einen eindeutigen Bezeichner gekennzeichnet und durch einen Wertebereich beschrieben. Sie definiert ein Maß, über dessen Meßwerten dann Einschränkungen formuliert werden können. CQML definiert die Typen `numeric`, `set` und `enum`. Im folgenden Beispiel ist eine Charakteristik zur Beschreibung der Antwortzeit 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.

Bei Verwendung von `numeric` kann der Wertebereich durch die Intervalnotation (`(`,`)` oder `[`, `]`) beschränkt werden. Des Weiteren 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.

Der Typ `enum` steht für Enumeration. In dem Fall kann ein Eintrag von einer Nutzer – definierten Menge ausgewählt werden. Es besteht zudem die Möglichkeit für die Einträge ein Ordnung anzugeben. In dem Fall muss die semantische Ordnungsrichtung angegeben werden (`increasing` oder `decreasing`).

Bei Verwendung des Types `set` wird immer eine Teilmenge von Nutzer – definierten Mengen von Namen ausgewählt. Wie in `enum` kann zu dem Wertebereich eine Ordnung angegeben werden.

Erweiterungen

Eine Charakteristik kann von einer anderen abgeleitet werden. In diesem Fall definiert die abgeleitete Charakteristik eine Einschränkung der ursprünglichen Domäne.

Es besteht ebenfalls die Möglichkeit, der Charakteristik Parameter zuzuweisen. Ein Parameter kann zum Beispiel ein Objekt, eine Methode, einen Fluss oder eine Signal beschreiben. Aus den Parametern wird der aktuelle Wert der Charakteristik errechnet (siehe Ausführungen zu Schlüsselwort *value*).

Des Weiteren können statistische Aspekte angegeben werden, z. B. maximum, percentile und distribution. Diese werde aus einer Folge von Werten über einen bestimmten Zeitraum ermittelt.

Neben der Spezifizierung durch statistische Aspekte, kann eine Charakteristik auch durch andere *QoS* – Charakteristiken beschrieben werden. Der Wert der Charakteristik hängt dabei von den Werten der anderen Charakteristiken ab.

Semantik von Charakteristiken

In der Charakteristik können Invarianten (`invariant`) festgelegt und die Semantik der Werte (`values`)beschrieben werden. Die nach dem Schlüsselwort `invariant` beschriebenen Eigenschaften müssen sich immer zu einer wahren Aussage auswerten lassen.

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

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

Wertzuweisung

Neben fest zugesicherten Werten (Schlüsselwort `guaranteed`), können innerhalb einer Anweisung auch best – mögliche Werte (Schlüsselwort `best-effort`) definiert werden. Die Einhaltung dieser Zusagen kann nicht garantiert werden, da der Wert nur unter besten Bedingungen erreicht werden kann. Bei einer einfachen Auflistung der Charakteristiken müssen alle Werte eingehalten werden. Es besteht aber auch die Möglichkeit, mehrere Charakteristiken mit `OR` zu verbinden. Die Gesamtheit der *QoS* – Charakteristiken in einem Statement kann somit als *Konjunktive Normalform* (*KNF*) beschrieben werden.

Anweisungen können abgeleitet werden. In diesem Fall werden die Wertebereiche der Charakteristiken weiter beschränkt oder neue Charakteristiken aufgenommen.

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

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

`provides` - *QoS* – Eigenschaften, die die Komponente anbietet

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 z.B. in *OMG IDL* spezifiziert werden.

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

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

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 kann das Schlüsselwort `any` oder der Name eines Profils verwendet werden. `Any` steht für jedes beliebige Profil.

QoS – Verträge

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

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 Profil nur `uses` – und `provides` – Klauseln enthält, wurde um das Schlüsselwort `resources` erweitert. Es werden die Betriebsmittel beschrieben, die die Komponente benötigt. Die dort verwendeten Parameter verweisen auf die Ressourcen.

Die Arbeit von Aagedal zu CQML sowie die EBNF-Grammatik zu CQML$^+$ werden zur Verfügung gestellt.

2.4 Definition der CQML – Spezifikation in XML

Zur Nutzung in der Laufzeitumgebung wird eine XML-Repräsentation der Spezifikation erzeugt. In dieser sind einige Konstrukte von CQML an die Laufzeitumgebung angepasst.

Das XML-Schema mit der zugehöhrigen Dokumentation wird ebenfalls zur Verfügung gestellt.

Kapitel 3

Anforderungen an eine Komponenten - Verwaltung

In diesem Kapitel wird die zu entwickelnde Laufzeitumgebung beschrieben. Diese Laufzeitumgebung wird als Komponenten – Verwaltung bezeichnet. Sie ist in die in Abschnitt 2.1 vorgestellte Infrastruktur eingebettet. Bei den Betrachtungen werden einige Lösungsalternativen vorgestellt.

3.1 Architektur der Komponenten – Verwaltung

Der Aufbau der Komponenten – Verwaltung ist der Abbildung 3.1 zu entnehmen.

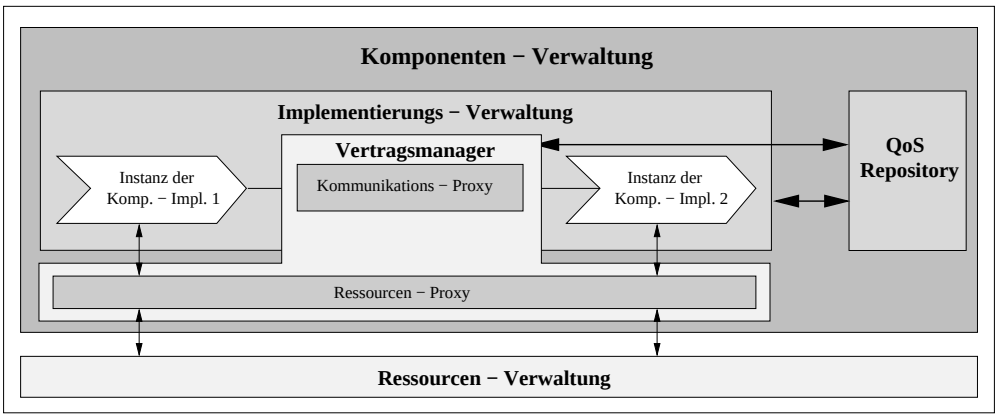

Abbildung 3.1: Komponenten – Verwaltung

In den folgenden Abschnitten wird ausführlicher auf die Verwaltungseinheiten eingegangen.

3.2 *QoS* – Repository

Die Aufgabe des *QoS* – *Repository* besteht im Einlesen und Verwalten der *QoS* – Eigenschaften. Es muss diese Informationen zur Laufzeit zur Verfügung stellen, damit sie von den anderen Funktionseinheiten der Komponenten – Verwaltung verarbeitet werden können. Die QoS-Eigenschaften liegen als XML-Datei (siehe XML Schema für CQML$^+$) vor. Beim Einlesen der XML-Datei kann davon ausgegangen werden, dass bereits eine Typüberprüfung durchgeführt wurde.

Zur Erfüllung seiner Aufgabe weist das *QoS* – Repository die *QoS* – Eigenschaften den Komponenten – Implementierungen zu. Dafür wird an jede Komponente (`Component`) ein Profil (`QoS_Profile`), das die *QoS* – Eigenschaften beschreibt, gebunden (siehe Abb. 3.2).

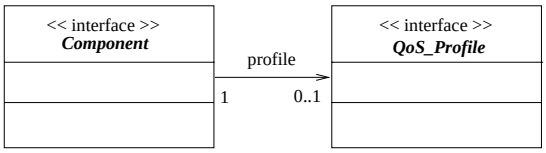

Abbildung 3.2: Definition der Datentypen

Während der Aushandlung der Verträge müssen die *QoS* – Eigenschaften ausgewertet und miteinander verglichen werden können. Das *QoS* – *Repository* ist entsprechend zu strukturieren.

3.3 Implementierungs – Verwaltung

Die Aufgaben der Implementierungs – Verwaltungen bestehen in der:

- Verwaltung der Komponenten – Implementierungen mit
  - Einlesen und Entfernen von Komponenten – Implementierungen,
  - Verwaltung der zu einer Komponenten – Spezifikation gehörenden Implementierungen sowie
  - Verwaltung der Abhängigkeiten einer Komponenten – Implementierung
- und der Verwaltung der zur Laufzeit erstellten Instanzen.

Diese Aufgaben werden in den folgenden Abschnitten beschrieben.

3.3.1 Verwaltung der Komponenten – Implementierungen

In der Implementierungs – Verwaltung werden Schnittstellen (Komponenten – Spezifikationen) verwaltet (siehe Abbildung 3.3). Für jede Schnittstelle existie-

ren mehrere Komponenten – Implementierungen, die die funktionale Spezifikation dieser Schnittstelle unterschiedlich umsetzen. Insbesondere unterscheiden sich verschiedene Implementierungen durch unterschiedliche *QoS* – Eigenschaften.

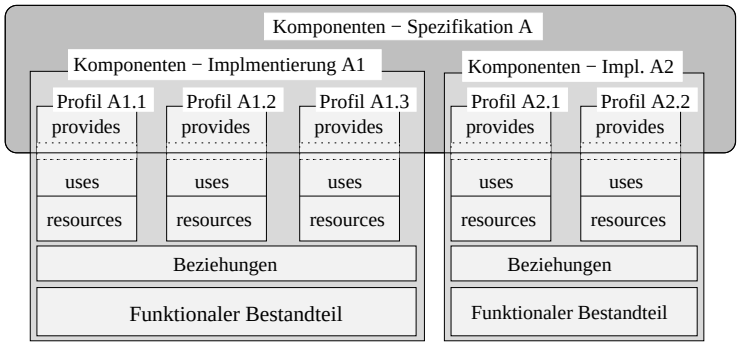

Abbildung 3.3: Zusammenfassung von Komponenten – Implementierungen zu einer Komponenten – Spezifikation

Jede Implementierung kann über mehrere alternative Profile verfügen, die unterschiedliche Ressourcenanforderungen beschreiben. Ein Profil besteht aus den bereits auf Seite 9 beschriebenen Bereichen *uses*, *provides* und *resources*.

Des Weiteren kann in einer Implementierung auf andere Komponenten zugegriffen werden, das heißt es existieren Abhängigkeiten mit anderen Komponenten – Spezifikationen. Diese sind daher in Abb. 3.3 in einem separaten Bereich *Beziehungen* dargestellt. Die Beziehungen zwischen Komponenten und zwischen Komponenteninstanzen werden in separaten XML-Dateien beschrieben. Es existiert eine Datei, die in Anlehnung an CORBA Components IDL die Komponentenspezifikationen enthält, sowie eine, die in Anlehnung an CORBA Component Assembly Descriptor eine Spezifikation der Beziehungen auf Instanzenebene enthält. Die Details zum Aufbau dieser Dateien finden sich in separaten Dokumentationen.

Einlesen der Komponenten – Implementierungen: Das Einlesen soll zum Start des Servers aber auch während der Laufzeit ermöglicht werden.

Beim Start des Servers werden alle angegebenen Komponenten – Beschreibungen eingelesen, die sich in einem dem Server bekannten Verzeichnis befinden. Dieses Verzeichnis muss regelmässig auf neu eintreffende Komponenten überprüft werden.

Entfernen der Komponenten – Implementierungen: Das Entfernen einer Komponenten – Implementierung während der Laufzeit ist möglich, indem die Referenz auf dieses Objekt verworfen wird. Dabei muss überprüft werden, ob diese Komponenten – Implementierung gerade in Verwendung ist, oder innerhalb einer Anfrage noch verwendet werden soll.

Sollte eine zu entfernende Komponente noch an die Anwendung gebunden sein, kann die Implementierung erst entgültig aus dem Container entfernt werden, wenn alle Anfragen an Objekte dieser Implementierung beendet wurden. Gleichzeitig wird die Implementierung als nicht mehr verwendbar markiert, damit neu ankommende Requests nicht mehr durch Instanzen dieser Implementierung berabeitet werden.

Nach Abschluss aller beschriebenen Schritte, werden die Referenzen auf die Objekte dieser Komponenten – Implementierungen gelöscht.

3.3.2 Instanzen – Verwaltung

Eine weitere wichtige Funktion der Implementierungs – Verwaltung ist die Verwaltung der zur Laufzeit erstellten Instanzen. Jede Komponenten – Implementierung bietet eine *Home* – Schnittstelle an. Außerdem besitzt sie eine Schnittstelle für den Zugriff auf die Geschäftsmethoden. Mit Hilfe von *CQML* können den Geschäftsmethoden einer Komponenten – Implementierung unterschiedliche *QoS* – Eigenschaften zugewiesen werden.

Obwohl alle Clients ein und dieselbe Komponenteninstanz verwenden können, kann der Zugriff zu einem Zeitpunkt nur durch einen Client erfolgen. Wird gerade eine Geschäftsmethode im Objekt ausgeführt, müssen die anderen Clients warten, bis die Verarbeitung abgeschlossen ist. Um dieses Blockieren zu verringern, können mehrere Objekte je Profil erstellt und so die Last verteilt werden.

Im folgenden Abschnitt soll der Lebenszyklus einer Komponente vorgestellt werden.

Lebenszyklus

Da Komponenten keinen Konversationszustand verwalten, können alle Clients auf ein und dasselbe Objekt zugreifen. Beendet ein Client seinen Zugriff auf ein Objekt muss dieses nicht entfernt werden, da es noch von anderen Clients wiederverwendet werden kann. Allerdings ist zu beachten, dass nicht mehrere Clients zur gleichen Zeit auf die Methoden einer Instanz zugreifen können.

Obwohl sich die Clients in der Komponenten – Verwaltung eine Instanz teilen können, muss zu jedem Profil eine eigene Instanz erstellt werden, da jeweils andere Ressourcenanforderungen erfüllt werden müssen.

Eine Instanz wird nur zur Verarbeitung einer Methodenanfrage direkt einem Client zugewiesen. Die Reservierung einer Instanz erfolgt auf Ebene eines Verwaltungsobjekts.

Beim ersten Zugriff auf eine Komponente wird eine neue Instanz erzeugt und die Ressourcen werden reserviert. Anschließend geht die Komponenteninstanz in den Zustand „Bereit für Zugriff“ über. Beim Aufruf einer Geschäftsmethode geht die Instanz in den Zustand „Zugriff erfolgt“ über und die Geschäftsmethode wird ausgeführt. In diesem Zustand kann kein anderer Client auf Geschäftsme-

thoden dieser Instanz zugreifen. Nach dem Beenden der Geschäftsmethode geht die Instanz zurück in den Zustand „Bereit für Zugriff“ und steht nun wieder anderen Clients zur Verfügung. Eine Instanz, die sich im Zustand „Bereit für Zugriff“ befindet, kann auch gelöscht werden. Dabei werden die reservierten Ressourcen wieder freigegeben. Abb. 3.4 stellt den Lebenszyklus als Zustandsdiagramm dar.

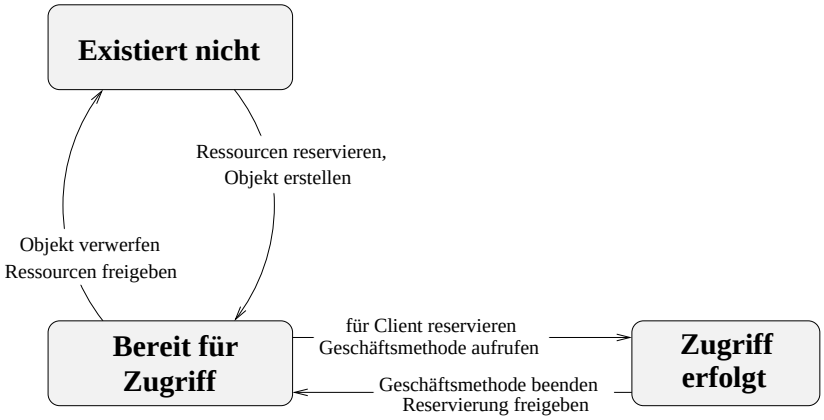

Abbildung 3.4: Lebenszyklus einer Komponente

Die Erstellung der Instanzen könnte bereits erfolgen, wenn die Komponenten – Implementierungen eingelesen werden. Ein Objekt kann aber auch erstellt werden, wenn es für einen Client reserviert werden soll. In dieser Situation wird von der Komponenten – Verwaltung überprüft, wie viele Clients für dieses Profil eine Instanz reserviert haben, und wie viele Instanzen tatsächlich zur Verfügung stehen. Ergibt die Überprüfung, dass ein Grenzwert[1] überschritten wurde, kann eine neue Instanz erstellt werden.

3.4 Vertragsmanager

Die Aufgabe des Vertragsmanagers besteht in der Aushandlung und Durchsetzung der Verträge zur Verarbeitung einer Anfrage.

Er wird aktiv sobald ein Client eine Komponente anfordert. Da eine Komponenten – Implementierung Betriebsmittel benötigt und eventuell noch andere Komponenten – Implementierungen bei ihrer Ausführung verwendet, muss vor Gewährung einer Client – Anfrage überprüft werden, ob die benötigten Ressourcen verfügbar sind. Dieser Vorgang der Überprüfung wird als Vertragsaushandlung bezeichnet. Er unterteilt sich in folgende Arbeitsschritte:

1. Vergleich der *QoS* – Eigenschaften zwischen kommunizierenden Komponenten – Implementierungen,

[1] Der Grenzwert kann zum Beispiel das Verhältnis von 'existierende Instanzen' zu 'Reservierungen durch Clients' beschreiben. Der Wert wird durch Anwendungsentwickler spezifiziert.

2. Reservierung der Ressourcen über die Ressourcen – Verwaltung und

3. Erstellung der Instanzen sowie Reservierung dieser für die Client – Anfrage.

Nach Aushandlung aller Verträge entsteht ein Netz aus kommunizierenden Instanzen. Ein Beispiel für dieses Netz ist in der Abbildung dargestellt.

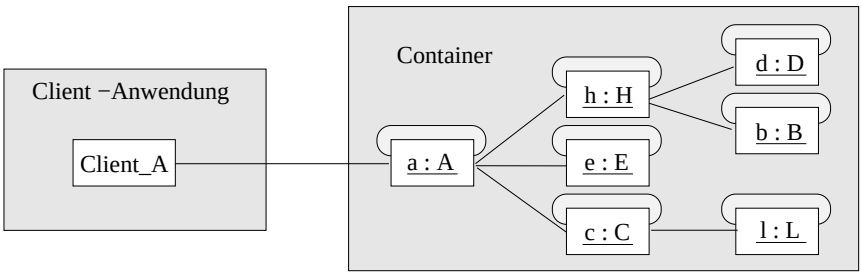

Abbildung 3.5: Komponenten – Netz

Nach Reservierung aller Ressourcen und Erstellung des Komponenten – Netzes kann die Anfrage des Clients ausgeführt werden. Dabei müssen durch den Vertragsmanager die reservierten Instanzen zugewiesen und die reservierten Ressourcen verwendet werden. Diese Aufgaben übernehmen die Funktionsuntergruppen *Kommunikations – Proxy* und *Ressourcen – Proxy*.

Beendet ein Client seine Verarbeitung müssen durch den Vertragsmanager alle Verträge aufgelöst werden. Dabei werden die reservierten Ressourcen freigegeben.

Auf die zuvor angesprochenen Funktionen soll in den nächsten Abschnitten ausführlich eingegangen werden.

3.4.1 Vertragsaushandlung

Die Vertragsaushandlung beginnt, nachdem eine Client – Anwendung eine Referenz auf eine Komponente angefordert hat. Clients rufen dazu eine Methode `getRemoteHome` auf und übergeben ihr den Namen der gewünschten Schnittstelle sowie die QoS-Anforderungen des Clients.

Nachdem vom Server zu der geforderten Schnittstelle eine passende Implementierung gefunden wurde, für die auch zu allen abhängigen Komponenenten – Spezifikationen passende Implementierungen reserviert werden konnten, wird die Referenz auf diese Instanz an den Client übergeben.

Auf die einzelnen Schritte der Vertragsaushandlung wird in den nächsten Abschnitten eingegangen. Sie beinhaltet die Suche nach geeigneten Implementierungen für angeforderte Spezifikationen. Des Weiteren müssen für die gefundenen Implementierungen Ressourcen reserviert werden. Erst nach Abschluss beider Arbeitsschritte gilt die Vertragsaushandlung als erfolgreich.

Suche nach den geeigneten Implementierungen

Bei der Suche nach geeigneten Implementierungen spielt der Suchalgorithmus sowie der Vergleich von QoS-Spezifikationen eine wichtige Rolle.

Ansätze für Suchalgorithmen: Zu jeder Komponenten – Implementierung wird registriert, welche Komponenten – Spezifikationen sie in ihrer Verarbeitung verwendet. Für alle diese Spezifikationen muss eine passende Implementierung gefunden und eine Instanz davon reserviert werden.

Beim Vergleich der Profile wird innerhalb der Komponenten – Implementierungen immer in der durch den Anwendungsentwickler vorgegebenen Ordnung vorgegangen.

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

- 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.

- Eine bessere Lösung wäre die Ermittlung aller möglichen Implementierungen und anschließender Auswahl der besten Alternative. Dieses Verfahren kann in zwei Varianten umgesetzt werden:

  1. Bei Anfrage eines Clients werden alle Lösungsvarianten für das gesamte Komponenten – Netz erstellt. Im Anschluss kann aufgrund unterschiedlicher Kriterien das beste Netz ausgewählt werden.

  2. Der Vergleich der Alternativen erfolgt für jede bereits ausgewählte Komponenten – Implementierung separat. Es werden in dem Fall für die benötigten Schnittstellen alle geeigneten Implementierungen heraus gesucht und im Anschluss die Beste ausgewählt.

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

Profile gelten als passend, wenn die von den `uses`-Klauseln geforderten *QoS* – Eigenschaften durch die `provides`-Klauseln vollständig abgedeckt werden. Das heißt, was durch die `provides`-Klauseln angeboten wird, darf höchstens besser sein, als was durch die `uses`-Klauseln gefordert wird.

Reservierung der Ressourcen

Nachdem eine Implementierung gefunden wurde, die die geforderten *QoS* – Anforderungen erfüllt, wird eine Instanz benötigt, die für den Client reserviert werden kann. Steht keine Instanz zur Verfügung muss eine neue erstellt und die nötigen Ressourcen reserviert werden. Dazu werden die *resources* – Einträge des ausgewählten Profils ausgewertet. Die daraus abgeleitete Ressourcen – Anfrage wird an die Ressourcen – Verwaltung weitergeleitet.

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 einen *Handle*, der an die Instanzen gebunden wird (siehe Abb. 3.6). Eine ausführliche Beschreibung der Einträge im *Handle* ist der Beschreibung der Schnittstelle der Ressourcen – Verwaltung zu entnehmen.

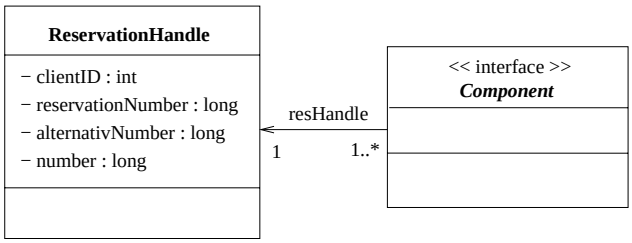

Abbildung 3.6: Komponenten – Netz

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

Stehen die benötigten Ressourcen nicht zur Verfügung liefert die Ressourcen – Verwaltung eine Fehlermeldung zurück. Da die ausgewählte Implementierung nicht verwendet werden kann, muss die Suche fortgesetzt werden.

Die Funktionen die während der Reservierung der Ressourcen auszuführen sind, werden in der folgenden Liste noch einmal zusammengefasst.

- Auslesen der Ressourcenanforderungen
- Reservierung der Ressourcen (Zugriff auf *Ressourcen – Verwaltung*)
- Auswertung der Rückgabe der *Ressourcen – Verwaltung*
- Übergabe des *Handles* zum Zugriff auf die Ressourcen an die Instanzen

3.4.2 Durchsetzung der Verträge

Nachdem die Verträge für eine Client – Anfrage ausgehandelt wurden, wird dem Client die Referenz auf die Basis des reservierten Netzes übergeben. Im Anschluss daran kann er Geschäftsmethoden auf dieser Instanz ausführen. Der Vertragsmanager übernimmt die Aufgabe der Durchsetzung der Verträge, sobald auf die Geschäftsmethoden der reservierten Instanzen zugegriffen wird. Diese Überwachung unterteil sich in zwei Funktionsbereiche.

Der *Kommunikations – Proxy* hat die Aufgabe, die reservierten Instanzen zuzuweisen, während der *Ressourcen – Proxy* die Verwendung der reservierten Betriebsmittel durchsetzt. Die Funktionen dieser Bestandteile des Vertragsmanagers werden im Folgenden vorgestellt.

Kommunikations – Proxy

Wenn eine Komponente mit einer anderen kommunizieren will, ruft sie die `getRemoteHome`-Methode auf und übergibt den Namen der Schnittstelle, über die sie kommunizieren möchte. Bei der Vertragsaushandlung wurden vom Container Kommunikationsproxies installiert, die diese Aufrufe abfangen und die ausgehandelte Instanz zurückliefern.

Weiterhin wird jeder Aufruf von Operationen in einer anderen Komponente durch den Kommunikations – Proxy geleitet. Dadurch können auch diese Aufrufe überwacht und ggf. *QoS*-Eigenschaften (z.B. Verschlüsselung) durchgesetzt werden.

Ressourcen – Proxy

Nach Reservierung der Ressourcen müssen diese Verträge durchgesetzt werden. Das ist die Aufgabe des *Ressourcen – Proxy*.

Er muss aktiv werden, sobald auf Betriebsmittel zugegriffen wird. In dem Moment leitet er die Anfrage direkt an die reservierten Ressourcen weiter. Die Überwachung der Einhaltung der Verträge (z. B. Größe des reservierten Speicherplatzes) übernimmt die Ressourcen – Verwaltung.

3.4.3 Freigabe der reservierten Ressourcen

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 (und daran gebundene Instanzen) freigegeben.

Im Sever sind die Komponenten – Netze aufzulösen und die Instanzen und reservierten Ressourcen freizugeben.

Literaturverzeichnis

[1] Comquad webseite. http://www.comquad.org.

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

[3] Sun Microsystems. Enterprise JavaBeans Specification, version 2.0. Final Release, August 2001.

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

[5] Dale Rogerson. *Inside COM: Microsoft's Component Object Model*. Microsoft Press, 1997.

[6] Clemens Szyperski. *Component Software : Beyond Object-Oriented Programming*. Addison-Wesley Publishing Company, November 1997.