TUD-FI03-10-August 2003

Ronald Aigner, Elke Franz, Steffen Göbel, Hermann Härtig, Heinrich Hußmann, Klaus Meißner, Klaus Meyer-Wegener, Marcus Meyerhöfer, Andreas Pfitzmann, Christoph Pohl, Martin Pohlack, Simone Röttger, Alexander Schill, Frank Wehner, Steffen Zschaler

Zwischenbericht der DFG-Forschergruppe 428 „Components with Quantitative Properties and Adaptivity (Comquad)“

Zwischenbericht der DFG-Forschergruppe 428 „Components with Quantitative Properties and Adaptivity (COMQUAD)“

Ronald Aigner, Elke Franz, Steffen Göbel, Hermann Härtig, Heinrich Hußmann, Klaus Meißner, Klaus Meyer-Wegener, Marcus Meyerhöfer, Andreas Pfitzmann, Christoph Pohl, Martin Pohlack, Simone Röttger, Alexander Schill, Frank Wehner, Steffen Zschaler

Technische Universität Dresden,
Fakultät Informatik,
01062 Dresden,
schill@rn.tu-dresden.de

Friedrich-Alexander-Universität Erlangen-Nürnberg,
Institut für Informatik,
Martensstr. 3,
91058 Erlangen,
kmw@informatik.uni-erlangen.de

http://www.comquad.org/

1. Einleitung ... 3
1.1. Ziele des Projekts ... 3
1.2. Zusammenarbeit der Teilprojekte ... 3
2. Das Anwendungsszenario ... 4
2.1. Funktionalität des Börsentickers ... 5
2.2. Architektur der Systemumgebung ... 7
3. Entwicklungsmethodik ... 8
3.1. Entwicklung serverseitiger Komponenten ... 10
3.2. Entwicklung anwendungsseitiger Komponenten ... 12
4. Spezifikationssprache ... 13
4.1. Architekturbeschreibung ... 13
4.2. Nichtfunktionale Eigenschaften ... 14
4.3. Schutzziele ... 15
4.3.1. Besondere Problemstellung bei Komponentensystemen ... 15
4.3.2. Sicherheitsanforderungen am Anwendungsbeispiel ... 16
4.3.3. Spezifikation der Schutzziele ... 17
4.4. Systemseitige Adaption ... 17
4.5. Anwendungsseitige Adaption ... 18
4.5.1. Dokumentkomponenten ... 18
4.5.2. Adaptation auf den einzelnen Abstraktionsebenen: ... 19
5. Abbildungsmechanismen ... 20
6. Laufzeitumgebung ... 23
6.1. Konzept ... 23
6.1.1. Komponentenverwaltung ... 24
6.1.2. Sicherheitsmechanismen ... 25
6.1.3. Ressourcenverwaltung ... 26
6.1.4. Systemseitige Adaption ... 28
6.1.5. Anwendungsseitige Adaption ... 28
6.2. Prototypen ... 29
7. Zusammenfassung ... 31
8. Literatur ... 31
Anhang A: IDL des Börsenticker-Beipiels ... 33

1. Einleitung

1.1. Ziele des Projekts

Ziel des COMQUAD Projektes ist die Unterstützung von quantitativen Eigenschaften und Adaption bei Komponenten. Dazu gehören folgende Teilziele:

- Quantitative Eigenschaften müssen geeignet spezifiziert und mit den entsprechenden Komponenten in Verbindung gebracht werden. Dazu gehören Mechanismen um quantitative Anforderungen an eine Komponente auf Anforderungen dieser Komponente an andere Komponenten abzubilden.
- Der Entwicklungszyklus einer Komponente mit quantitativen Eigenschaften soll einer genau beschriebenen Entwicklungsmethodik folgen.
- Erstellte Komponenten bedürfen dann der Unterstützung ihrer Laufzeitumgebung, um die zugesagten quantitativen Eigenschaften zu gewährleisten und gegebenenfalls eine Adaption durchzuführen.

Eine Komponente bietet einerseits quantitative Eigenschaften anderen Komponenten an, andererseits kann sie quantitative Eigenschaften anderer Komponenten nutzen. Zur Laufzeit müssen sich zwei Komponenten auf konkrete Eigenschaften dieser Beziehung festlegen und darüber einen Vertrag schließen. Dieser Vertrag beinhaltet, dass die nutzende Komponente sich auf Zusagen der anbietenden Komponente verlassen kann und diese ihre Leistungen mit den garantierten quantitativen Eigenschaften erfüllt. Dies gilt unter der Voraussetzung, dass wiederum Verträge, welche sie mit von ihr genutzten Komponenten geschlossen hat, eingehalten werden.

Kann zu irgendeinem Zeitpunkt die anbietende Komponente den Vertrag nicht mehr erfüllen, besteht eine mögliche Reaktion auf die Vertragsverletzung in der Adaption des Vertrages. Dazu müssen beide Komponenten mit den neuen Bedingungen einverstanden sein. Dies kann bereits bei Vertragsabschluss vereinbart worden sein, oder es muss ein neuer Vertrag ausgehandelt werden. Adaption findet natürlich auch statt, wenn sich die Anforderungen der nutzenden Komponente ändern.

Als Laufzeitumgebung für die Komponenten dient ein Container, ähnlich dem Container von Enterprise Java Beans Komponenten. Der Container bietet den Komponenten verschiedenste Dienste an. Diese können von der einfachen Instanziierung einer neuen Komponente bis hin zur Garantie der Persistenz von Daten reichen. Ein COMQUAD Container muss zudem Dienste zur Vertragsaushandlung und –überwachung anbieten. Für alle Komponenten bietet der Container eine Abstraktion der Ressourcen an.

1.2. Zusammenarbeit der Teilprojekte

Wie in Abbildung 1 dargestellt, entsprechen den einzelnen Teilzielen des COMQUAD Projektes verschiedene Teilbereiche. Dabei hat für jedes Teilziel eine Gruppe des Projektes die Leitung und erhält Zuarbeiten von anderen Gruppen für einzelne Teilbereiche.

Die Entwicklung der Spezifikationssprache wird von der Softwaretechnologie (ST) Gruppe geleitet. Die anderen Gruppen steuern ihre Expertise bei den entsprechenden Teilbereichen bei. So liefert die Gruppe Rechnernetze (RN) Zuarbeiten zur Spezifikation von Komponentenarchitekturen – zusammengesetzten Komponenten. Die Gruppe Datenschutz und Datensicherheit (IK) steuert ihr Fachwissen zum Thema Schutzziele bei und die Gruppen Multimediatechnik (MM) und RN steuern Fachwissen zur anwendungsseitigen Adaption bei. RN und die Gruppe Betriebssysteme (BS) beschäftigen sich mit systemseitiger Adaption. Alle Gruppen sind bei der Spezifikation von quantitativen Eigenschaften beteiligt.

Das Teilziel Abbildungsmechanismen wird von der Gruppe Datenbanken (DB) geleitet. Da diese Arbeit sowohl in die Spezifikationssprache einfließt, als auch die Gestaltung der Laufzeitumgebung beeinflusst, liefern alle Gruppen Beiträge zu Teilbereichen dieses Zieles.

Die Gruppe ST entwickelt eine Methodik für den Entwurf und die Bereitstellung von Komponenten mit quantitativen Eigenschaften. Die Methodik basiert auf den Ergebnissen und Erfahrungen der andern Gruppen

Die Laufzeitumgebung wird in verschiedenen Teilbereichen realisiert. Die zugrunde liegende Ressourcenverwaltung wird von der Gruppe Betriebssysteme entwickelt. Die darauf aufbauende Komponentenverwaltung wird von der Gruppe RN entwickelt. Die Gruppe IK steuert Arbeiten zu Sicherheitsmechanismen bei. Die Gruppen RN und MM integrieren Adaptionsmechanismen in die Laufzeitumgebung. Die Schnittstelle zwischen Ressourcenverwaltung und Komponentenverwaltung ist wohl definiert, was eine spätere Integration beider erlaubt.

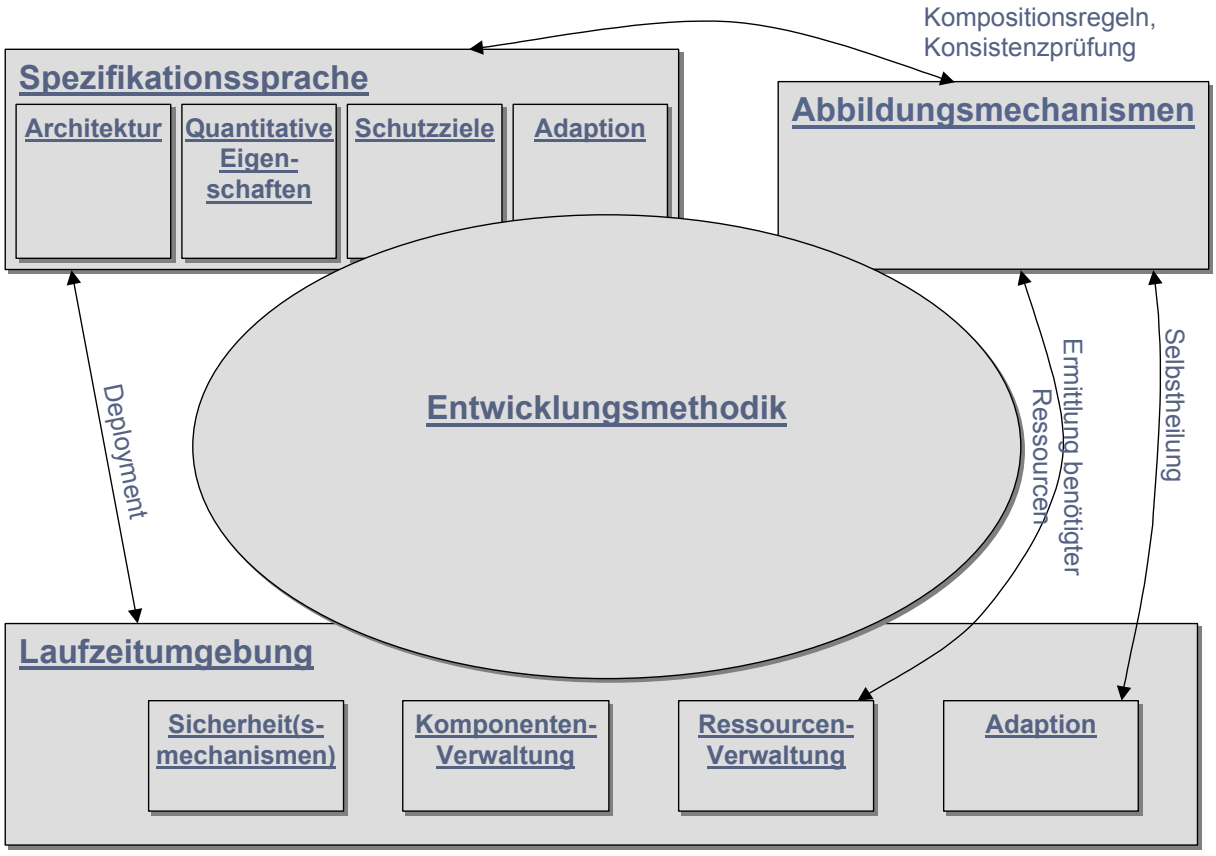

Abbildung 1: Zusammenarbeit der Teilprojekte

Nachdem im folgenden Abschnitt das für COMQUAD ausgewählte Anwendungsszenario vorgestellt wurde, gibt Kapitel 3 einen Überblick über die dem Projekt zugrunde liegende Entwicklungsmethodik. Danach wird in Kapitel 4 CQML$^+$ als Sprache zur Spezifikation nichtfunktionaler Eigenschaften näher erläutert. Im Anschluss daran folgt in Kapitel 5 eine Beschreibung möglicher Abbildungsmechanismen. Kapitel 6 stellt das Konzept von Laufzeitumgebung und Prototypen vor, Kapitel 7 schließlich gibt eine Zusammenfassung.

2. Das Anwendungsszenario

In diesem Abschnitt soll das gemeinsame Beispielszenario vorgestellt werden, das die Basis für die folgenden Kapitel darstellt. Ziel dieses Szenarios ist die Entwicklung einer Client-/Serverarchitektur zur dynamischen Generierung multimedialer Börseninformationsdokumente. Kern der Präsentation soll dabei ein durch den Anwender konfigurierbarer Börsenticker sein, der sich auch an Änderungen der client- und serverseitigen Systemumgebung anpassen

kann. Bevor auf die Systemumgebung dieses Beispiels eingegangen wird, soll es zunächst aus funktionaler Sicht näher erläutert werden.

2.1. Funktionalität des Börsentickers

Der Börsenticker erlaubt es Benutzern, sich für verschiedene Aktien zu registrieren. Anschließend werden die Benutzer ständig über den aktuellen Stand dieser Aktienkurse informiert. Bei der Registrierung geben sie an, wie aktuell die Information sein muss, d.h. wie viele Millisekunden nach einer Aktualisierung sie informiert werden möchten. Es handelt sich um eine verteilte Anwendung mit drei Knoten: Client-Rechner, Börsenticker-Rechner und Börsenserver. Der Börsenserver ist ein fiktiver Dienst der Deutschen Börse, der in regelmäßigen Abständen die aktuellen Kursstände aller angefragten Aktien an interessierte Systeme verschickt. Der Börsenticker-Rechner ist der Knoten, auf dem die eigentliche Anwendung läuft, d.h. das Verteilen der Aktienkurse und die Gewährleistung der angeforderten QoS.

Abbildung 2 stellt die Strukturierung des Börsentickers in Komponenten in einem UML-Diagramm dar. Komponenten wurden dabei als Subsysteme modelliert. Die senkrechten gestrichelten Linien stellen zusätzlich die Knotengrenzen dar.

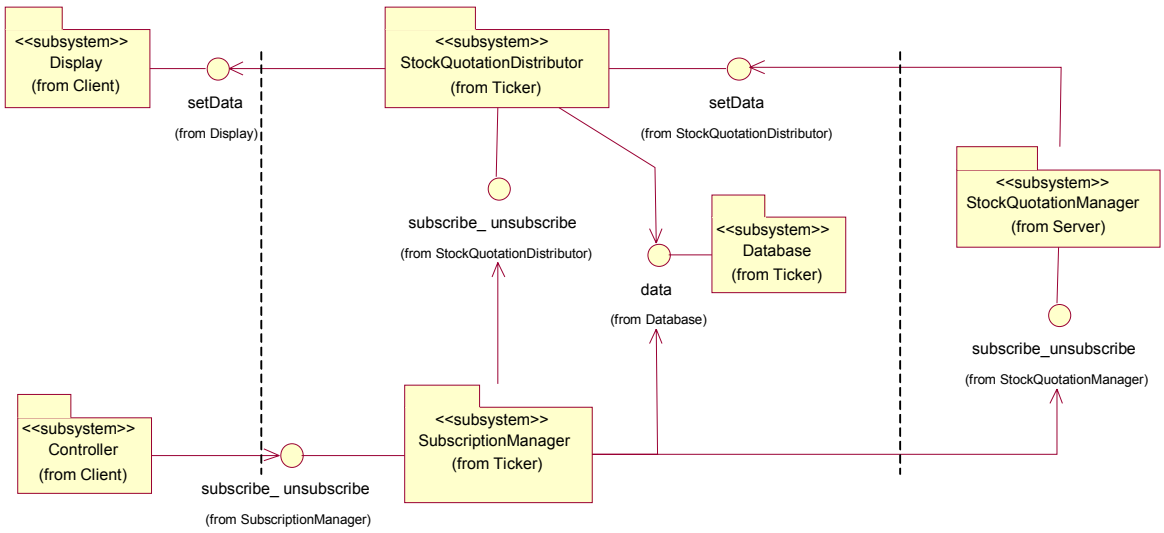

Abbildung 2: Komponentenstruktur des Börsentickers

Will ein Kunde sich für einen Aktienkurs registrieren (s. Abbildung 3), so lässt er sich zunächst eine Liste aller verfügbaren Aktien anzeigen, aus der er anschließend eine auswählt. Die Auswahl wird in eine `subscribe()`-Nachricht an den Börsenticker übersetzt, der nun dafür sorgt, dass der Nutzer in Zukunft über den ausgewählten Aktienkurs informiert wird. Dazu werden die Registrierungsdaten in einer Datenbank gespeichert und die Verteilungskomponente wird informiert. Sollte dies die erste Registrierung für diesen Aktienkurs sein, so muss zusätzlich der Börsenserver informiert werden, damit der Börsenticker in Zukunft benachrichtigt wird.

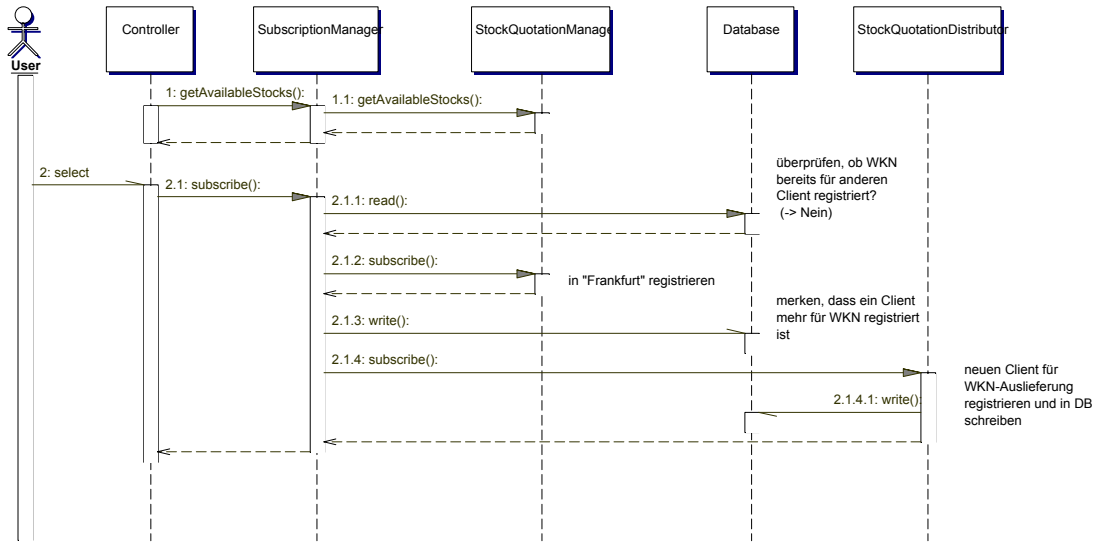

Abbildung 3: UML-Sequenzdiagramm des Ablaufs beim Registrieren für einen Aktienkurs (WKN=Wertpapierkennummer – ein Identifikator für Aktien)

Beim Abwählen eines Aktienkurses werden diese Aktionen im Wesentlichen wieder rückgängig gemacht. Das heißt es wird eine `unsubscribe()`-Nachricht an den Börsenticker versandt, der daraufhin die Datenbank bereinigt und den Verteiler informiert. Sollte dies der letzte Client gewesen sein, der über die entsprechende Aktie informiert wurde, so wird auch der Börsenserver informiert, so dass dieser keine weiteren Informationen über diese Aktie mehr verschickt. Dieser Ablauf ist in Abbildung 4 als Sequenzdiagramm dargestellt.

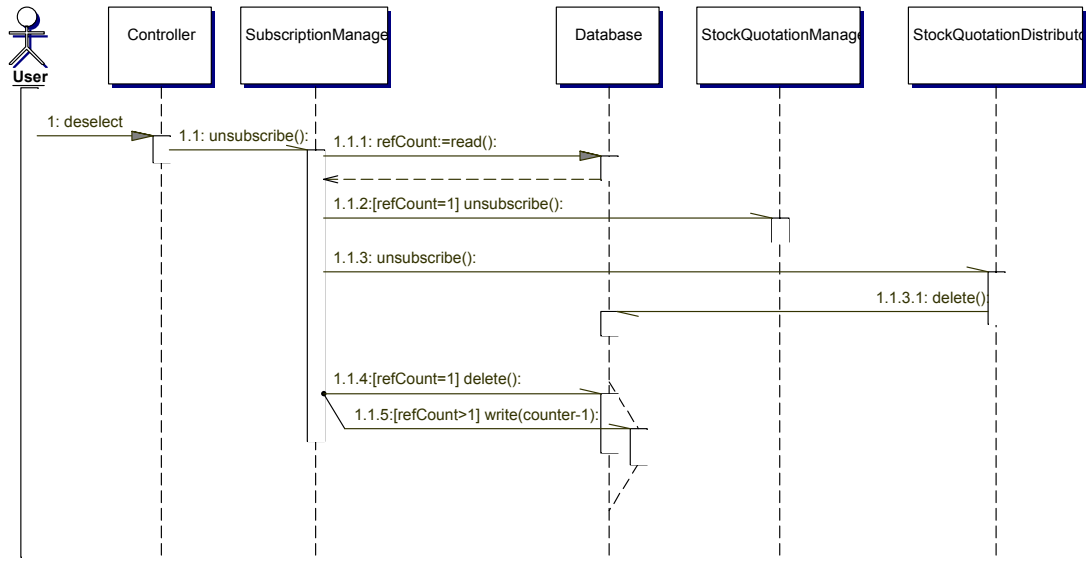

Abbildung 4: Der Ablauf beim Abwählen eines Aktienkurses

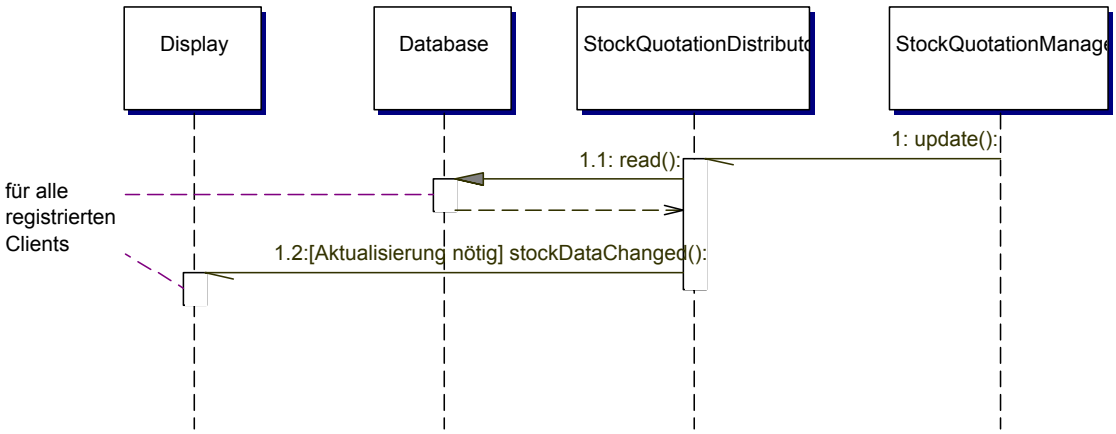

Abbildung 5: Aktualisierung von Aktienkursen

In regelmäßigen Abständen verschickt der Börsenserver Informationen über die geänderten Aktienkurse. Diese werden von der Verteiler-Komponente des Börsentickers ausgewertet und an die interessierten Clients weitergereicht. Dies ist in Abbildung 5 dargestellt.

2.2. Architektur der Systemumgebung

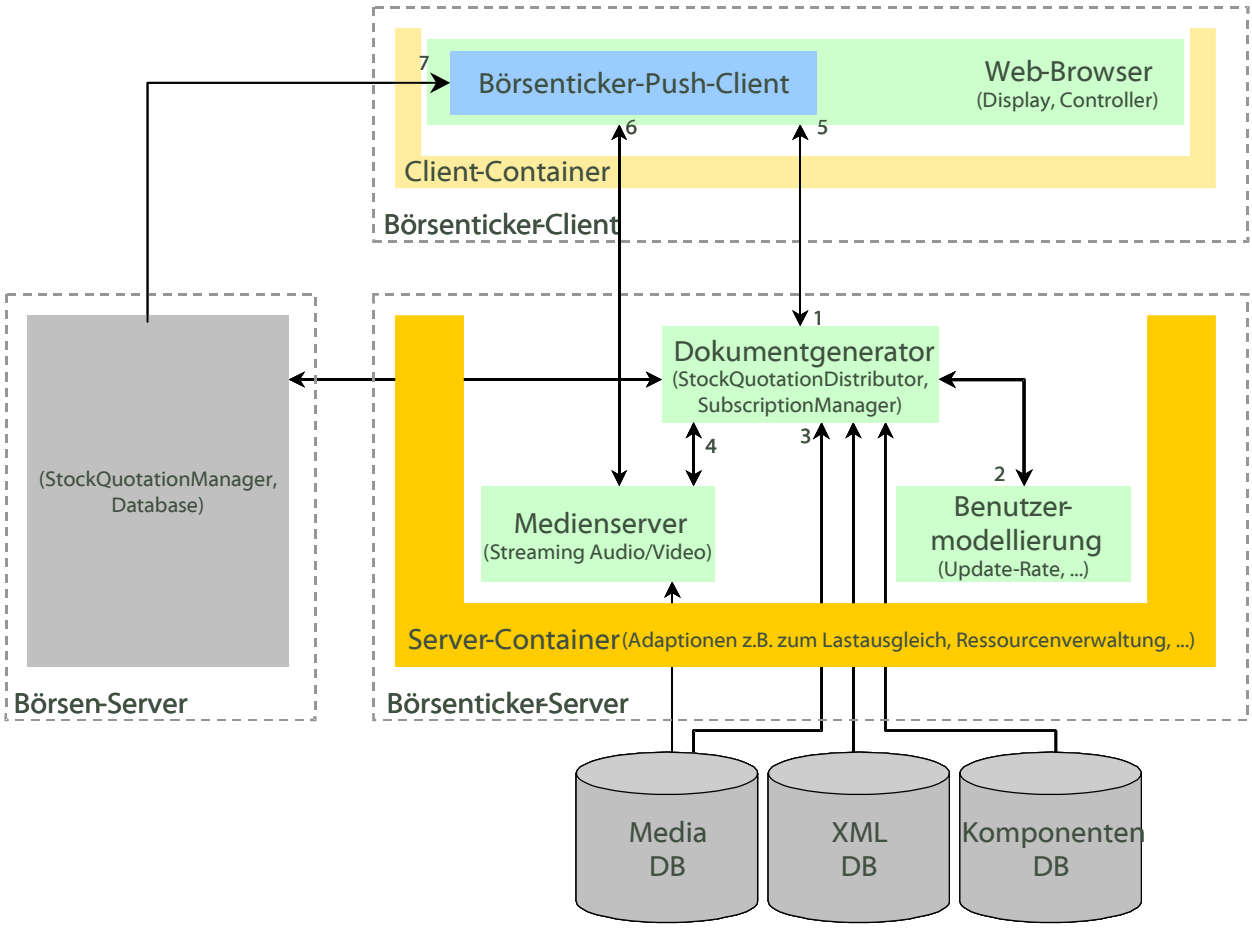

Abbildung 6: Gesamtarchitektur des Prototyps

Die Generierung von Börsen-Dokumenten wird in folgende Schritte unterteilt (Abbildung 6):

1. Clientseitig werden die technischen Parameter der Systemumgebung (Bildschirm- und Fenstereigenschaften, Netzanbindung, …) sowie die erfassten Benutzerinformationen (z.B. die gewünschte Update-Rate des Börsentickers) registriert und gemeinsam mit der Anforderung des nächsten Dokuments an den Dokumentgenerator auf dem Börsenticker-Server gesendet.
2. Der Börsenticker-Server startet zunächst die Benutzermodellierung und erhält das aktualisierte Benutzermodell zurück.
3. Danach liest der Dokumentgenerator das angeforderte Dokument sowie die benötigten Medienobjekte aus den entsprechenden Datenbanken.
4. Der Dokumentgenerator bindet die zur Darstellung von Medien benötigten Informationen in die Präsentation ein und liefert sie
5. zurück an den Börsenticker-Client.
6. Dieser kontaktiert den Medienserver, der das angeforderte Medienobjekt an den Client überträgt.
7. In die Präsentation wird ebenfalls ein Börsenticker-Push-Client z.B. als Java-Applet eingebunden, der unabhängig von der Anforderung eines neuen Börsendokuments vom Börsenserver die aktuellen Kursinformationen erhält.

Nach der Vorstellung der Systemumgebung des Anwendungsszenarios beschäftigt sich das folgende Kapitel mit der Entwicklungsmethodik für adaptive, komponentenbasierte Anwendungen mit nichtfunktionalen Eigenschaften.

3. Entwicklungsmethodik

Für den Bau von Software ist gerade unter dem Aspekt der Wiederverwendung die systematische Nutzung einer Entwicklungsmethodik unerlässlich. Unter einer Methodik versteht man einen Baukasten von Methoden, Konzepten, Verfahren und Technologien. Im einzelnen umfasst die methodische Softwareentwicklung folgende Teilgebiete: Vorgehensmodell, Beschreibungstechniken und Systemmodell, Technologie und Architektur, Managementpraktiken und Werkzeugunterstützung.

Durch das Zusammenspiel der einzelnen Elemente einer Methodik ergibt sich das in Abbildung 7 dargestellte Gesamtbild.

Für die Konzeption einer Methodik zur Entwicklung von komponentenbasierten Sofwaresystemen mit zusagbaren nichtfunktionalen Eigenschaften müssen folgende Punkte beachtet werden.

Im Projekt COMQUAD geht es im Wesentlichen um die Zusagbarkeit von nichtfunktionalen Eigenschaften, funktionale Aspekte werden bewusst ausgeklammert. Diese sinnvolle Einschränkung hat auch Auswirkungen auf die methodischen Aspekte. Im Rahmen des Projekts wird deshalb auch nur der systematische Entwurf der nichtfunktionalen Anforderungen verfolgt, der aber untrennbar mit den funktionalen verbunden ist. Deshalb werden die methodischen Aspekte von Beginn an, an einer funktionalen Entwicklungsmethodik für komponentenbasierte Systeme festgemacht [ChDa01].

Des weiteren betrachtet COMQUAD *Software*komponenten nach der derzeit verbreiteten Begriffsdefinition [Szyp97] als: „*A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A softwarecomponent can be deployed independendly and is subject to composition by third parties.*“ Für jede Komponente sollen außerdem präzise Beschreibungen der nichtfunktionalen Eigenschaften bestehen. Komponenten werden von Containern ausgeführt. Dieser kann verschiedenste physische und

virtuelle Ressourcen sowie allgemeine Middlewaredienste zur Verfügung stellen. Da als Anwendungsgebiet auch multimediale Anwendungen in Betracht gezogen werden, bieten die hier betrachteten Komponenten neben einfachen Aufrufschnittstellen und Ereignismechanismen auch Schnittstellen für kontinuierliche Datenströme an.

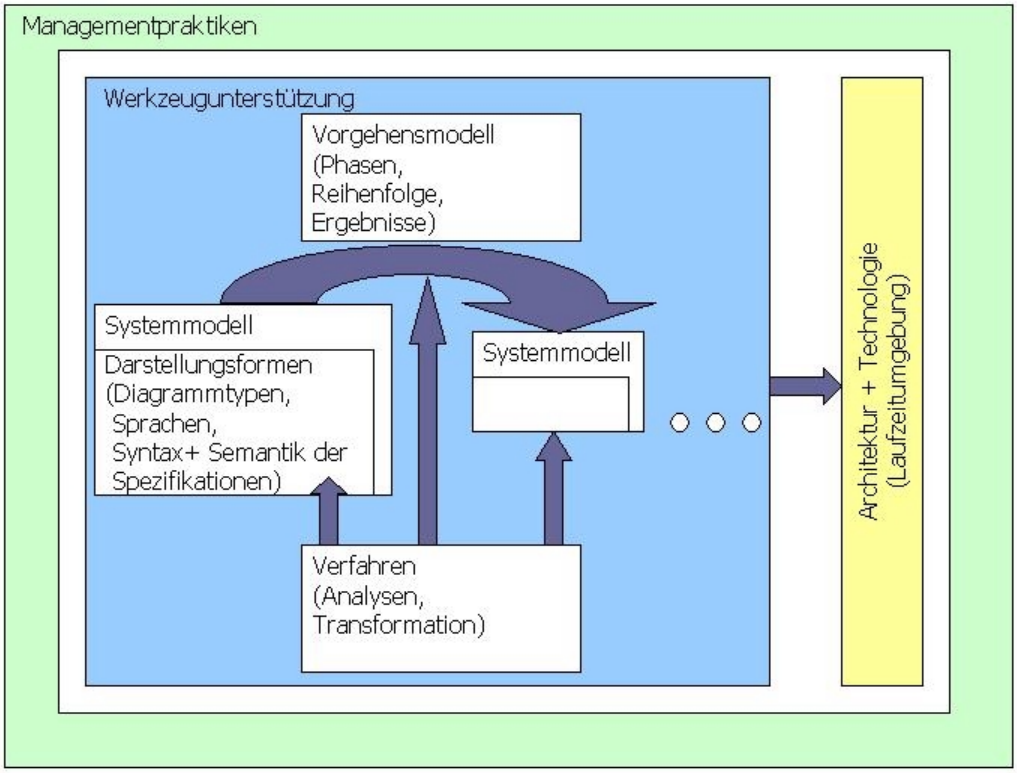

Abbildung 7: Entwicklungsmethodik

Eine weitere Entwicklung in Bezug auf multimedialen Anwendungskontext muss ebenfalls in Betracht gezogen werden. Aktuelle multimediale (insbes. Web-basierte) Anwendungen sind in zunehmendem Maße dokumentbestimmter Natur, d.h. nicht nur die der Anwendung zugrunde liegenden Daten, sondern auch funktionale und nicht funktionale Eigenschaften solcher Anwendungen werden über Dokumentformate beschrieben. Dies hat zur Folge, dass ebenfalls die Adaptionsprozesse in diesen Anwendungen bereits auf Dokumentebene spezifiziert werden müssen. Um nun auch solche Anwendungen komponentenorientiert entwerfen und spezifizieren zu können, ist es nötig, die dem Paradigma der *Software*komponenten zugrunde liegenden Ansätze und Methoden für die Nutzung in dokumentbestimmten multimedialen Anwendungen anzupassen und auf diese Anwendungsklasse abzubilden. Deshalb wird im Rahmen des COMQUAD-Projektes der Ansatz der *Dokument*komponenten verfolgt. Dokumentkomponenten sind dabei komplette Instanzen bzw. Instanzfragmente solcher multimedialen Dokumentformate. Die Schnittstellen dieser Komponenten sowie weitere Eigenschaften und ihr adaptives Verhalten werden über zugeordnete Metadaten beschrieben. Adaptive Dokumente werden durch Aggregation von Dokumentkomponenten gebildet und beschreiben beispielsweise eine komplette Web-Site. Wesentliche Gemeinsamkeiten zu klassischen Softwarekomponenten sind:

- Dokumentkomponenten bilden ebenfalls systemunabhängige, wiederverwendbare und abgeschlossene Einheiten.
- Eine wohldefinierte (und über Metadaten beschriebene) Schnittstelle ermöglicht ihre Anpassung und Einbindung.

- Sie stellen eine wohldefinierte Funktionalität bereit, alle Kontextabhängigkeiten sind explizit bekannt.
- Sie können zu komplexeren Komponenten bzw. ganzen Anwendungen kombiniert werden.

Der einzige wesentliche Unterschied zu herkömmlichen Softwarekomponenten ist, dass sie meist nicht in einem Binärformat vorliegen, was jedoch keine gravierende Einschränkung des Komponentenkonzeptes darstellt.

3.1. Entwicklung serverseitiger Komponenten

Die elementaren Schritte des Vorgehensmodells sind in Abbildung 8 dargestellt. Über eine Anforderungsanalyse werden Use-Case Szenarien entwickelt, an denen nichtfunktionale Kriterien mittels der Spezifikationssprache CQML$^+$ (Kapitel 4.2) beschrieben werden. Die Use-Cases gehen in den Entwurf der Anwendung ein. Die Eigenschaften der Gesamtanwendung müssen nun durch Dekomposition den Einzelkomponenten zugeordnet werden. Für diesen Prozessschritt kommen vorwiegend Komponenten-, Sequenz- und Kollaborationsdiagramme der UML zum Einsatz. CQML$^+$ dient der Beschreibung der nichtfunktionalen Eigenschaften. Des Weiteren ist es notwendig über eine entsprechende Architekturbeschreibung die Interaktionen zwischen den Komponenten zu spezifizieren. An diesem Punkt können bereits existierende Komponenten eingesetzt werden. Es können aber auch Anpassungen (Adapter) bzw. Neuentwicklungen notwendig sein. Den Gegenpol zur Dekomposition bildet die Komposition der Einzelkomponenten zur gesamten Anwendung. Die bisher betrachteten Schritte werden solange iterativ durchgeführt, bis die Gesamtanwendung die auf Spezifikationsebene geforderten funktionalen und nichtfunktionalen Eigenschaften erfüllt. Im letzten Schritt wird die Anwendung einschließlich aller notwendigen Informationen für die Laufzeitumgebung bereitgestellt.

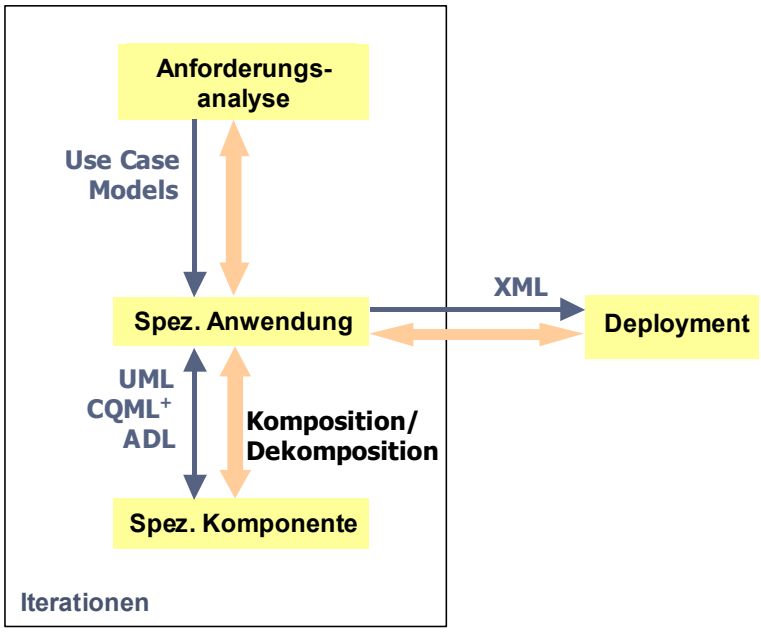

Abbildung 8: Entwurf serverseitiger Komponenten mit Entwurfsschritten und zugehörigen Darstellungsmitteln

Anhand einer Kundenanforderung an das Beispielsszenario werden im folgenden die Eckpunkte der Entwicklung dargestellt. Ein Nutzer des Börsentickers kann fordern, dass er be-

reits 30s nach dem Abonnieren eines Tickers die erste Kursinformation erhält. Im Use-Case (Abbildung 9) wird dafür eine Notiz mit {ResponseTime<=30s} angeheftet.

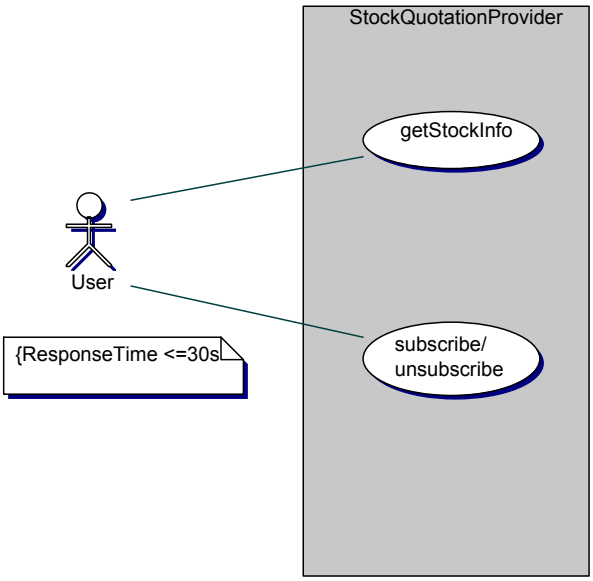

Abbildung 9: Use-Case einer Kundenanforderung mit ResponseTime<=30s

Über die Dekomposition der Anforderungen und mehrere Iterationsschritte in der Spezifikationsphase erhält man die funktionale Spezifikation des erforderlichen Komponentensystems (Abbildung 9). Parallel dazu werden die nichtfunktionalen Anforderungen spezifiziert. Im Beispiel der Antwortzeit wird dazu in CQML$^+$ eine Charakteristik `delay` definiert.

```
quality_characteristic delay (cause: Event, effect: Event) {
  -- Verzögerung zwischen zwei Events
  domain: numeric real [0..) seconds;

  values: effect.time() - cause.time();
}
```

Diese kann nun innerhalb der Spezifikation einer Qualität `response` auf 30s beschränkt werden.

```
quality acceptable_response (cause: Event, effect: Event) {
  delay (cause, effect) < 30;
}
```

Die Antwortzeit garantieren muss letztendlich der `SubscriptionManager`. Deswegen wird für ihn ein entsprechendes Profil definiert.

```
profile good_responsiveness for SubscriptionManager {
  provides acceptable_response (
    subscriptionInterface.subscribe.SR->last(),
    stockQuotationDistributor.outgoingUpdates.SE
      ->any (e |
             e = subscriptionInterface.subscribe.SR->last())
             -- Auswahl des entsprechenden Antwortevents.
}
```

Zum Installieren der Komponente in die Laufzeitumgebung werden die einzelnen Bestandteile der CQML$^+$-Spezifikation in Form eines XML-basierten QoS-Descriptors bereitgestellt.

Die hier beschriebene Vorgehensweise kann durch verschiedene Verfahren unterstützt werden. Vorrangig handelt es sich dabei um die Prüfung von syntaktischer und semantischer Korrektheit der Darstellungen, sowie der Konsistenzprüfung zwischen einzelnen Entwurfsdokumenten. Im Hinblick auf die Szenarien im COMQUAD-Projekt sind hier aber weitaus mehr Verfahren vorstellbar, z.B. das Prüfen von Verträgen auf ihre Realisierbarkeit bereits im Entwurf, die Prüfung auf mögliche Konflikte zwischen nicht-funktionalen Anforderungen, die Berechnung der Eigenschaften kompositer Komponenten sowie die automatische Unterstützung von Abbildungen zwischen Angeboten und Anforderungen von Komponenten.

Die bereits beschriebenen Schritte sowie mögliche Verfahren sollen durch ein Werkzeug unterstützt werden. Derzeit wird an der Werkzeugunterstützung für die Spezifikationssprache CQML$^+$ gearbeitet. CQML$^+$ ist für die Verwendung durch menschliche Entwickler sowie mit starkem Fokus auf Wiederverwendbarkeit von Spezifikationsteilen entworfen. Die Sprache ist daher stark strukturiert und verwendet Konzepte wie Namen und Namensräume. Für die maschinelle Verarbeitung sind viele dieser Konstrukte unnötig und bedeuten einen Verarbeitungs-Overhead. Daher verwaltet und verarbeitet die Laufzeitumgebung für COMQUAD-Komponenten diese Informationen in einem XML- basierten QoS-Deskriptor, welcher für die Verarbeitung durch die Maschine optimiert wurde. Diese Übersetzung, einschließlich der syntaktischen und semantischen Überprüfung der CQML$^+$-Spezifikation, bildet den derzeitigen Schwerpunkt der Arbeit an der Werkzeugunterstützung.

3.2. Entwicklung anwendungsseitiger Komponenten

In den letzten Jahren sind mehrere Entwurfsmodelle hypermedialer Web-Anwendungen entstanden, z.B. OOHDM (*Object-Oriented Hypermedia Model*, [ScRo98]), RMM (*Relationship Management Methodology*, [Isak+95]) oder WebML (*Web Modelling Language*, [Ceri+00]). Sie unterteilen die Entwicklung von Hypermedia-Anwendungen in mehrere Phasen, meist den inhaltlichen Entwurf, den Entwurf der Navigationsbeziehungen, einen mehr oder weniger abstrakten Layoutentwurf sowie die eigentliche Implementierung. Der Schwerpunkt dieser Modelle liegt jedoch nicht auf dem Gebiet adaptiver und komponentenbasierter Web-Anwendungen. Ein komponentenorientierter Ansatz für die Erstellung von Web-Präsentationen wird in [GaGr00] nahe gelegt. Dazu wird in [Gaed+00] die WebComposition Markup Language (WCML) vorgestellt, die Komponenten als uniformes Modellierungskonzept für Artefakte im Lebenszyklus von Web-Anwendungen zur Verfügung stellt. Jedoch ist auch hier Adaption kein zentraler Aspekt des Ansatzes. Eine weitere interessante Technik stellen Web-Content-Managementsysteme [WCMS, JaMe02] dar. Diese konzentrieren sich jedoch, statt auf den konzeptionellen Entwurf von Web-Anwendungen, eher auf ihre konkrete Implementierung und kontinuierliche Wartung.

Ziel ist es deshalb, den in Kapitel 3.1 vorgestellten Ansatz an das Konzept der komponentenorientierten Entwicklung adaptiver Web-Anwendungen anzupassen. Auch beim Entwurf dokumentbestimmter Anwendungen muss zunächst eine Anforderungsanalyse erfolgen. Danach müssen die benötigten Komponenten spezifiziert bzw. aus einem Repository vorhandener Komponenten ausgewählt und konfiguriert werden. Schließlich werden sie zu Web-Dokumenten kombiniert, getestet und kommen zum Einsatz.

Autorenumgebungen für adaptive multimediale Web-Auftritte setzen somit Werkzeugunterstützung u.a. in folgenden Bereichen voraus:

- Spezifikation und Entwurf adaptiver Web-Anwendungen,
- Erstellung der eigentlichen Inhaltskomponenten; manuelle bzw. teilautomatische Bestimmung der beschreibenden Metadaten; Spezifikation des Adaptionsverhaltens,

- Repositorien zur effektiven Speicherung, Versionierung und Wiederauffindung von Inhalten; Unterstützung der Zusammenarbeit verteilt arbeitender Autoren,
- Test adaptiver Web-Anwendungen unter Berücksichtigung variierender Benutzer- und Endgeräte-Parameter.

Eine Autorenumgebung zur Erstellung und Wartung adaptiver, komponentenhafter Web-Auftritte müsste demnach weniger als monolithisches Produkt, vielmehr als das Zusammenspiel unabhängiger, über wohlbekannte Schnittstellen kommunizierender Teillösungen entwickelt werden. Ähnlich zur Architektur von Web-Content-Managementsystemen [JaMe02] erscheint es sinnvoll, ein mehrbenutzerfähiges Asset-Managementsystem für die Verwaltung von Medienobjekten, abstrakten Komponenten sowie Nutzer- und Nutzungsdaten als Grundlage zu nehmen, auf welches während des Lebenszyklus von Web-Inhalten (Analyse, Entwurf, Erstellung, Tests) unterschiedliche Kontexte durch die Einbindung entsprechender Sichten und Editoren geboten werden. Dieses Asset-Managementsystem kann dann sowohl an unabhängige Werkzeuge zur Analyse- und Benutzermodellierung als auch an ein pipelinefähiges Publishing-Werkzeug angebunden werden kann.

4. Spezifikationssprache

Zur Beschreibung der QoS-Anforderungen an ein System wird eine Spezifikationssprache benötigt. Dabei müssen verschiedene Aspekte berücksichtigt werden. Die Sprache muss insbesondere die folgenden Bereiche abdecken:

- Grundlegende Architekturbeschreibung (s. Abschnitt 4.1) — diese beschreibt die einzelnen Komponenten und wie diese miteinander verbunden sind und dient als Anknüpfungspunkt für die QoS-Anforderungen.
- Beschreibungsmöglichkeit für nichtfunktionale Eigenschaften — erlaubt, die quantitativ erfassbaren QoS-Anforderungen an das System, wie beispielsweise Verzögerung, Schwankung, Speicherbedarf, Farbtiefe… (s. Abschnitt 4.2), sowie Sicherheitsanforderungen (Abschnitt 4.3) zu beschreiben.
- Beschreibungsmöglichkeiten für Adaption — dies umfasst sowohl systemseitige (s. Abschnitt 4.4) als auch anwenderseitige Adaption (s. Abschnitt 4.5). Es werden Konzepte geboten, um verschiedene akzeptable QoS-Level zu spezifizieren und Übergänge zwischen diesen zu beschreiben.

Im Folgenden wird auf jeden dieser Bereiche im Detail eingegangen.

4.1. Architekturbeschreibung

Die Komposition von Komponenten manifestiert sich in so genannten Komponentennetzen, die zur Deployment- bzw. ggf. zur Laufzeit durch Verknüpfung von Komponenten auf Typ- und Instanzebene gebildet werden. Solche Netze werden typischerweise durch so genannte Architekturbeschreibungssprachen (Architecture Description Languages / ADL) formuliert.

Klassische Vertreter solcher Sprachen sind z.B. Darwin, Rapide, Acme, Meta-H, C2SADL und xArch/xADL [Dash01]. Leider diktieren die meisten Sprachen gleichzeitig auch ein Software-Architekturkonzept, z.B. C2 Style bei C2SADL, oder sie genügen nicht allen Anforderungen hinsichtlich eindeutiger Identifizierbarkeit von Instanzen und deren Verknüpfbarkeit, z.B. xArch.

Das CORBA Component Model der OMG [Cor02] beschreibt unter anderem den Aufbau so genannter Component Assembly Descriptors (CAD) auf Basis von XML, die die Funktion einer Architekturbeschreibung erfüllen und den COMQUAD-spezifischen Anforderungen bzgl. Komponentenverknüpfung auf Instanzebene durchaus genügen. Es wird derzeit ein entspre-

chend angepasstes Schema dieser Deskriptoren erarbeitet, welches über logische Namen mit den Deskriptoren zur Beschreibung quantitativer Eigenschaften in Beziehung gebracht werden kann.

4.2. Nichtfunktionale Eigenschaften

Zur Spezifikation quantitativer Eigenschaften wird eine Erweiterung der Sprache CQML [Aag01] verwendet. CQML ist eine strukturierte, textbasierte Sprache zur Spezifikation nichtfunktionaler Eigenschaften von Komponenten. Alle Merkmale, über die etwas ausgesagt werden soll, werden als „quality characteristics“ deklariert. Anschließend werden in so genannten „quality statements“ Einschränkungen über den Charakteristiken formuliert. Diese Statements werden anschließend in „profiles“ an Komponenten, Operationen, Ströme etc. gebunden. Dabei erfolgt die Spezifikation stets in Form von Annahmen und Garantien (s. [Jon83]).

Ein wesentliches Manko von CQML ist, dass die Spezifikation des Ressourcenbedarfs einer Komponente nicht unterstützt wird. Da Ressourcen im Gegensatz zu Komponenten vom Container nicht direkt kontrolliert werden, ist eine besondere Behandlung notwendig. Der Container kann zwar Ressourcenanforderungen an die Ressourcenverwaltung (s. Abschnitt 6.1.3) geben und erhält daraufhin eine Bestätigung oder eine Ablehnung, er kann jedoch nicht steuern, wie die Ressourcen anderen Anwendungen zugeteilt werden, ob noch Ressourcen für ihn zur Verfügung stehen und welche von mehreren alternativen Ressourcenreservierungen tatsächlich ausgewählt wird. Für alle diese Fälle kann der Container lediglich Wünsche äußern und Hinweise geben, die die Ressourcenverwaltung aber auch ignorieren kann. Im Gegensatz dazu hat der Container vollständige Kontrolle darüber, welche Komponentenimplementierungen er instanziiert und wie er diese miteinander verbindet.

Ein anderes Problem ist, dass das zugrunde liegende Berechnungsmodell nur sehr informell definiert ist. Deshalb wurden im Projekt COMQUAD die folgenden Erweiterungen und Klarstellungen gefunden (siehe [RöZ03]):

CQML$^+$ erweitert CQML um die folgenden Konzepte:

1. Spezifikation von Ressourcen-Bedarf:

   Durch eine zusätzliche Klausel im Profile wird es ermöglicht, den Ressourcenbedarf einer Komponente zu spezifizieren. Diese Information wird direkt an die Ressourcenverwaltung zur Reservierung der Ressourcen übermittelt.

   Um für den Börsenticker zu spezifizieren, dass der `StockQuotationDistributor` 600 KB Speicher benötigt, kann man die folgende CQML$^+$-Spezifikation verwenden:

```
resource memory {
  quality_characteristic size (r: Resource) {
    domain: numeric real [0..) kilobytes;
  }
}

quality memory_usage (r: Resource) {
  size (r) = 600;
}

profile memory_usage for StockQuotationDistributor {
  resources memory_usage (memory);
}
```

   Zunächst wird in dem durch das Schlüsselwort `resource` eingeleiteten Ressourcenstatement der Ressourcentyp `memory` deklariert. Für jeden Ressourcentyp werden die zulässigen Charakteristiken beschrieben. Diese können im Anschluss wie andere Charakteristi-

ken verwendet werden, die entsprechenden Einschränkungen tauchen allerdings in einer speziellen `resources`-Klausel im Profil auf.

Der spezifizierte Ressourcenbedarf kann direkt an die Ressourcenverwaltung weitergegeben und von dieser interpretiert werden.

2. Strukturierte Charakteristiken

   Es wurde ein neuer Typ – `tuple` – für Charakteristiken definiert, der es erlaubt, die Werte mehrerer Charakteristiken gemeinsam zu betrachten.

   Dieser ist insbesondere bei der Deklaration von Ressourcen nützlich. Zum Beispiel kann CPU-Bedarf dargestellt werden als ein Tupel aus Periode und Ausführungszeit:

```
quality_characteristic cpu_demand (r: Resource) {
   domain: tupel {
             period (r),
             execution_time (r)
           };
  invariant: execution_time < period;
}
```

   Durch die Verwendung des Tupeltyps wird es möglich, die beiden Bestandteile in einer Invariante zueinander in Beziehung zu setzen.

3. Explizites Komponentenmetamodell

   Das Komponentenmetamodell, das CQML zugrunde liegt wurde explizit in UML modelliert. Es definiert alle Typen, die formale Parameter in CQML haben können und die Beziehungen zwischen diesen. Diese werden in den OCL-Ausdrücken beispielsweise der values-Klausel zur Navigation verwendet.

   Es wurde festgestellt, dass je nach zu definierender Charakteristik die Verwendung von Metamodellen auf verschiedenen Abstraktionsbenen sinnvoll ist.

4. Explizite Spezifikation von Abhängigkeiten

   Es ist vorgesehen, eine Syntax zur expliziten Spezifikation des Zusammenhangs zwischen angebotener QoS und benötigter QoS (benötigten Ressourcen) einer Komponente einzuführen. Diese wird vermutlich die Form einer zusätzlichen Klausel im Profile haben. Diese Konzepte werden noch diskutiert und sind noch nicht komplett ausgearbeitet [ZMay03].

4.3. Schutzziele

4.3.1. Besondere Problemstellung bei Komponentensystemen

Unter Sicherheit wird im allgemeinen die Erfüllung der Schutzziele Vertraulichkeit, Integrität, und Verfügbarkeit verstanden (siehe Abschnitt 4.3.2). Um mehrseitige Sicherheit zu erreichen, müssen alle Beteiligten am System dabei unterstützt werden, ihre Anforderungen zu formulieren und bei Konflikten untereinander auszuhandeln. Diese Probleme treten natürlich bei jeder Anwendung auf. Typisch für Komponentensysteme ist die Tatsache, dass der Nutzer nicht mehr nur *einer Anwendung eines* Herstellers trauen muss, sondern *einer Menge von Komponenten möglicherweise unterschiedlicher Hersteller*, die zur Erbringung eines Dienstes in einem Komponentennetz verbunden werden (siehe Abschnitt 6.1.1). Eine weitere Problematik ergibt sich aus der Tatsache, dass das Komponentennetz erst bei Anfrage des Nutzers dynamisch aufgebaut wird; der Nutzer hat also nicht von vornherein die Möglichkeit zu entscheiden, ob er der Anwendung vertraut. Auch die Ausführungsumgebung der Anwendung kann sich infolge Adaption verändern.

Wesentliches Kriterium für die Akzeptanz eines jeden IT-Systems ist die Frage, ob die Nutzer auf die Erbringung der geforderten Dienste vertrauen können. Im Fehlerfall sollte es möglich sein, **Verantwortlichkeit** zuzuweisen und diese auch zu übernehmen. Dafür muss bekannt sein, wovon die Erfüllung der Zusagen abhängt und wer wofür Verantwortung trägt. Dazu ist eine fälschungssichere Protokollierung wesentlicher Systemereignisse notwendig, welche ihrerseits die Identifizierbarkeit der Beteiligten erfordert. Bei Komponenten kann dabei zwischen Typidentität (installierte Version einer Komponente, gemäß Abschnitt 7.1.1 wären die verschiedenen Implementierungen einer Schnittstelle verschiedene Typen) und Inkarnationsidentität (zur Laufzeit erzeugte Instanz einer Komponente) unterschieden werden. Die Verantwortungszuweisung ist Gegenstand weiterer Untersuchungen.

Die **Ausführungssicht** beschreibt, welche Systemteile welche anderen ausführen. Die durch diese Sicht implizierten Schichten entsprechen unterschiedlichen Ausführungsebenen („abstrakten Maschinen“), wobei die tiefere Schicht die höhere ausführt. Die höheren Ausführungsschichten hängen völlig von den tieferen Schichten ab und können ohne diese nicht existieren. Den unteren Schichten muss prinzipiell vertraut werden, insbesondere bzgl. der Durchsetzung der spezifizierten Anforderungen wie dem Ressourcenbedarf.

Da die beim Entwurf definierte Schnittstellenspezifikation beschreibt, welche Leistungen eine Komponente zu erbringen hat, können Fehler durch Überwachung der Schnittstellen lokalisiert werden. Die Ausführungssicht wird benötigt, um mögliche Fehlerorte im Sinne von Ausführungsschichten zu beschreiben. Die beteiligten Ausführungssichten müssen in der Protokollierung erkennbar sein.

4.3.2. Sicherheitsanforderungen am Anwendungsbeispiel

Mögliche Sicherheitsanforderungen der Beteiligten sollen am Beispiel des Client-Rechners des Anwendungsbeispiels dargestellt werden (Tabelle 1).

Vertraulichkeitsziele	Vertraulichkeit der Dokumente gegenüber Beobachtern im Netz (Schutz der Interessensdaten, denn die Kurse an sich sind ja öffentlich zugänglich)
	Vertraulichkeit der Dokumente gegenüber anderen Anwendungen auf dem Server
	Vertraulichkeit gegenüber Komponenten, denen der Client nicht vertraut
	Anonymität der Kommunikationsumstände gegenüber Beobachtern im Netz
	Anonymität der Dienstnutzung (gegenüber dem Börsenticker, der damit keine personenbezogenen Daten über den Client sammeln kann).
Integritätsziele	Integrität der im Börsenticker erstellten Dokumente (richtige und aktuelle Kurse)
	Schutz der Integrität gegenüber Angreifern im Kommunikationsnetz
	Zurechenbarkeit der Dokumente zum Börsenticker (gegenüber Dritten kann der Client nachweisen, dass die Informationen von diesem Ticker stammen)
	Zurechenbarkeit der Diensterbringung (im Fehlerfall wird vom betreffenden Systemteil Verantwortung übernommen)
Verfügbarkeitsziele	Verfügbarkeit des angebotenen Dienstes
	Verfügbarkeit der Dokumente nach der geforderten Zeit

Tabelle 1: Sicherheitsanforderungen am Beispiel des Client-Rechners

Die Interessen des Börsentickers können von den Interessen des Clients u. U. abweichen; beispielsweise ist er nicht daran interessiert, dass der Client seinen Service anonym nutzt. In ei-

ner Aushandlung zwischen den Beteiligten muss entschieden werden, ob es zu einer Einigung kommt und welche Schutzziele dann umgesetzt werden sollen.

Neben diesen Schutzzielen auf „Datenebene“ ergeben sich aus der Flexibilität des Komponentenmodells auch Anforderungen bzgl. der Ausführungsschichten. So kann der Nutzer Interesse daran haben, Vertrauensbereiche bzgl. der Komponenten und ihrer Ausführungsumgebung zu spezifizieren.

4.3.3. Spezifikation der Schutzziele

Hierbei werden zunächst die Schutzziele spezifiziert, die sich durch Einsatz eines Mechanismus erfüllen lassen: Vertraulichkeit, Integrität und Zurechenbarkeit bzgl. der Kommunikation. Eine Sicherheitsbewertung der eingesetzten Mechanismen (in der Art: Stufe der Vertraulichkeit durch Benutzung welches Mechanismus oder welcher Parameter für einen Mechanismus) ist nicht Gegenstand der hier durchgeführten Betrachtungen. Solche Aspekte stellen eine Schnittstelle zur Kryptoanalyse dar, die eine Metrik für Bewertungen dieser Art liefern könnte; die Problematik ist jedoch nicht spezifisch für Komponentensysteme. Hier wird davon ausgegangen, dass das Schutzziel durch Einsatz des Mechanismus erfüllt wird.

Da der Einsatz von Sicherheitsmechanismen wiederum Ressourcen erfordert, müssen beim hier angestrebten Komponentenmodell eventuell Kompromisse zwischen Performance- und Sicherheitsanforderungen gefunden werden. Für die genannten Schutzziele ist demzufolge zu spezifizieren, wie wichtig deren Durchsetzung ist. Dazu wird ein Wert aus der Menge möglicher Wichtungen angegeben (entsprechend dem Projekt SSONET, siehe Abschnitt 8.1.2.3), die sowohl für die oben genannte Aushandlung als auch für die Aushandlung zwischen den Sicherheitsanforderungen unterschiedlicher Teilnehmer benötigt wird:

```
domain: enum (unbedingt, wenn möglich, egal, notfalls, nie)
```

Beispielsweise stellt der Client im Anwendungsszenario folgende Anforderungen:

```
communication_confidentiality(total_negotiable) and
communication_integrity(total) and
communication_accountability(total)
```

4.4. Systemseitige Adaption

Unter systemseitiger Adaption verstehen wir die Adaption innerhalb von Komponenten an Veränderungen der Systemumgebung. Dies beinhaltet vor allem die Anpassung des Bedarfs an genutzten Ressourcen. Die Größe dieses Bedarfes wird bestimmt durch die zu verrichtende Arbeit einer Komponente. Trifft es zu, dass die Arbeit einer Komponente so unterteilt werden kann, dass nach einem bestimmten Zeitpunkt ein Ergebnis vorliegt und dessen Qualität durch zusätzliche Arbeit verbessert wird, dann kann man diese Arbeit in Pflichtteil und Wahlteile unterteilen. Dieses Prinzip ist aus [Liu73] als Imprecise Computation bekannt. Auf das Beispiel des Börsentickers bezogen kann das heißen, dass mindestens die abonnierten Kurse angezeigt werden und wenn möglich noch Chart-Analysen.

Tritt ein Mangel an Ressourcen auf, welche zur Verrichtung der Arbeit benötigt werden, kann eine einfache Art der Adaption darin bestehen, dass Wahlteile der Arbeit weggelassen werden. Dadurch verschlechtert sich auch die Qualität der verrichteten Arbeit. Sind zu einem späteren Zeitpunkt wieder genügend Ressourcen vorhanden, können die Wahlteile wieder ausgeführt und damit die angebotene Qualität verbessert werden. Um eine Komponente über eine Veränderung der Quantität der benutzten Ressourcen zu informieren, haben wir eine *Notification* Schnittstelle definiert, welche von jeder Ressourcen nutzenden Komponente angeboten

werden muss. Unter Benutzung des mathematischen Modells von Hamann ([Ham97], [Ham01]) ist die Zusagbarkeit der geforderten Wahlteile garantiert.

Neben diesen Möglichkeiten parametrischer Adaption von Komponenten kann in komplexeren Komponentennetzen bzw. Instanznetzen zusätzlich noch Strukturadaptionen durchgeführt werden. Unter Strukturadaption verstehen wir Anpassungen von Komponentennetzen durch Hinzufügen, Weglassen und Ersetzen von Komponenten und Komponenteninstanzen aus vorhandenen Netzen.

Um hierbei alle externen Abhängigkeiten zu berücksichtigen, d.h. Kontexte in denen Schnittstellen von Komponenten des Netzes verwendet werden, muss die Komponentenverwaltung alle Verbindungen von Komponenten unter ihrer Kontrolle haben, um Abhängigkeiten zur Adaption erkennen und berücksichtigen zu können.

Ein geeignetes Konzept zur Kapselung solcher Abhängigkeiten bieten komposite Komponenten. Damit wird ein weiterer Komponententyp eingeführt, der in der Lage ist, nach außen wie eine einzige Komponente zu agieren und seine interne Struktur vollständig zu verbergen. Auf diese Weise können die genannten Arten von Strukturadaptionen vollkommen transparent durchgeführt werden, d.h. ohne dass die Benutzer der Schnittstellen der kompositen Komponente von der Anpassung Kenntnis nehmen müssen.

4.5. Anwendungsseitige Adaption

Anwendungsseitige Adaptionen, also Adaptionen die implizit oder explizit durch den Anwender ausgelöst werden und deren Ergebnis die beim Benutzer laufende Anwendung beeinflusst, sollen Thema des folgenden Abschnitts sein. Die in Kapitel 2 vorgestellte Börsenpräsentation ist ein typisches Beispiel für die immer stärker an Bedeutung gewinnende Klasse multimedialer und insbesondere Web-basierter dokumentzentrierter Anwendungen. Deshalb soll im folgenden Kapitel zunächst der Ansatz eines dokumentbasierten Komponentenmodells vorgestellt werden, bevor anschließend genauer auf Adaptionsmöglichkeiten bei diesem Komponententyp eingegangen wird.

4.5.1. Dokumentkomponenten

Bei einem dokumentbasierten Komponentenmodell werden Web-Sites aus konfigurierbaren Komponenten zusammengestellt. Unter Komponenten werden in diesem Kontext Dokumente oder Dokumentfragmente verstanden, die adaptiven Web-Inhalt auf unterschiedlichen Abstraktionsebenen repräsentieren (siehe Kapitel 3). Die hinzugefügten Metadaten beschreiben die Komponentenschnittstellen, indem sie ihre Eigenschaften und ihr adaptives Verhalten spezifizieren. Web-Sites werden durch die Aggregation solcher Komponenten zu komplexen Dokumentstrukturen aufgebaut. Im Prozess der Dokumentgenerierung werden diese anhand eines Benutzermodells sowie unterschiedlicher Endgeräteeigenschaften in ein konkretes Ausgabenformat umgewandelt.

Die Komponenten der im Folgenden vorgestellten Architektur lassen sich in vier Abstraktionsebenen einteilen (Abbildung 10):

Auf der Medienkomponentenebene befinden sich unterschiedliche Medienobjekte, denen sowohl technische als auch nicht-technische Metadaten zugeordnet werden können. Die Menge der Medienobjekte umfasst z.B. Texte, strukturierte Texte (HTML oder XML), Bilder, Audio, Video, und kann beliebig erweitert werden.

Auf der Ebene der Sinneinheiten werden Medienkomponenten zusammengefasst, die eine gemeinsame Aussage bilden und semantisch nicht voneinander getrennt werden sollten. Beispielsweise sollte eine erklärende Bildbeschriftung nicht losgelöst vom dazugehörigen Bild betrachtet werden. Auf dieser Ebene werden zusätzlich Layouteigenschaften definiert, die die

räumlichen Beziehungen bei der Anordnung der Medienkomponenten einer Sinneinheit beschreiben.

Die nächste Abstraktionsebene definiert Dokumentkomponenten: Bestandteile von Web-Dokumenten, die eine wohldefinierte semantische Rolle im Anwendungskontext tragen, z.B. die Komponente „Kursinformation“ mit einer Grafik zum Kursverlauf des letzten Jahres, einem erklärenden Text sowie der Möglichkeit, das entsprechende Wertpapier zu kaufen bzw. zu verkaufen. Dokumentkomponenten können sowohl Sinneinheiten, also auch andere Dokumentkomponenten referenzieren. Auch auf dieser Ebene wird die räumliche Anordnung hierarchischer Dokumentkomponenten durch abstrakte Layouteigenschaften beschrieben.

Während auf der Ebene der Sinneinheitskomponenten und auf der Ebene der Dokumentkomponenten Aggregationsbeziehungen festgelegt werden, ermöglicht die Hyperlinkebene eine Etablierung von Verknüpfungen zwischen Komponenten. Diese werden getrennt von den Komponenten verwaltet und können uni- oder bidirektional sein.

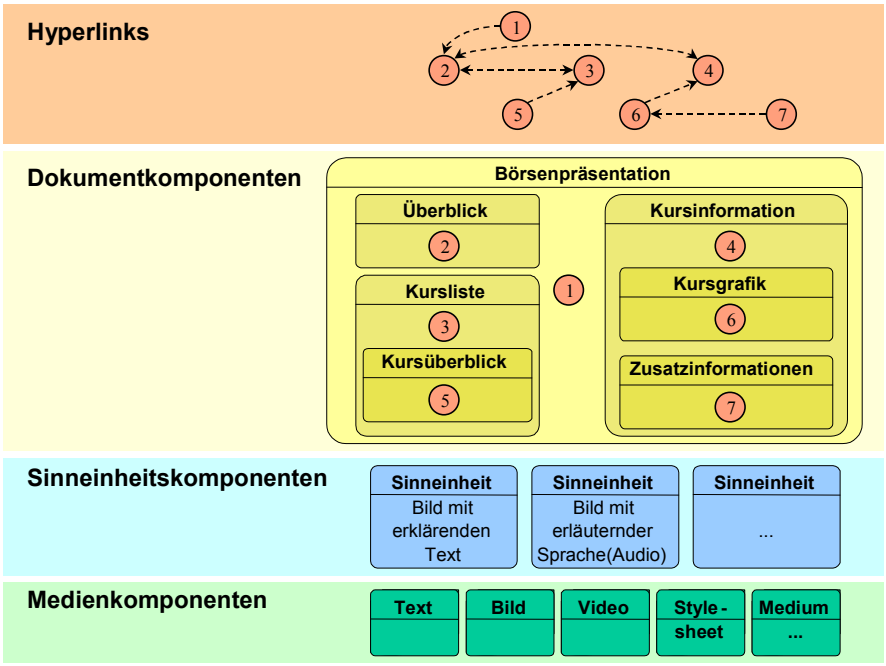

Abbildung 10: Ebenenmodell des Dokumentenformates

4.5.2. Adaptation auf den einzelnen Abstraktionsebenen:

Zuerst können auf der untersten Ebene die Medienobjekte an unterschiedliche Eigenschaften des Client-Endgerätes oder andere technische Parameter angepasst werden. Für das Börsenticker-Beispiel ist es nötig, verschiedene Varianten des Kurs-Diagramms mit unterschiedlichen Größen, Auflösungen oder Farbtiefen zur Verfügung zu stellen, um die Anpassung an vorliegende Bildschirmparameter zu ermöglichen.

Auch auf der Ebene der Sinneinheiten sind Adaptionen von großer Bedeutung. Zum einen können auch hier technische Eigenschaften des Client-Endgerätes berücksichtigt werden. In der Börsenpräsentation kann z.B. Benutzern mit hoher Netzwerkbandbreite das Video eines Interviews mit einem Börsen-Experten gezeigt werden, während anderen Benutzern nur der Interviewtext präsentiert wird. Zum anderen können Präsentationen auch bereits auf dieser Ebene bezüglich semantischer Benutzerpräferenzen adaptiert werden. In der Börsenpräsentation können z.B. je nach Vorliebe einem Anwender die Kursinformationen grafisch und einem anderen tabellarisch präsentiert werden.

Die Adaption auf der Ebene der Dokumentkomponenten betrifft hauptsächlich die Komponentenhierarchie. Dies führt zu unterschiedlichen Varianten von Komponenten-Teilbäumen mit unterschiedlichen Eigenschaften bzgl. Benutzeranforderungen und technischer Endgeräteparameter. Ein Beispiel ist die Anpassung der Börsenpräsentation an das Endgerät des Benutzers. Während am PC mehrere Kurse sowie zusätzliche Informationen parallel präsentiert werden können, ist es auf einem WAP-Handy nur möglich, jeweils einen Kurs darzustellen. Solche alternativen Inhaltsstrukturen werden auf der Dokumentkomponenten-Ebene modelliert.

Auf der obersten Hyperlink-Ebene werden Beziehungen zwischen Komponenten repräsentiert. Diese definieren ein Netzwerk über den Komponenten, das ebenfalls an den Anwender oder Anwendergruppen angepasst werden kann. Um diese Linkadaption zu ermöglichen, können auch auf der Hyperlink-Ebene Adaptionseigenschaften zugewiesen werden. Dadurch werden variable Hyperlink-Strukturen bzw. –sichten zur angepassten Navigation durch die Web-Präsentation ermöglicht.

Das Adaptionsverhalten von Komponenten wird auf den einzelnen Ebenen über zugeordnete Metadaten beschrieben. Dazu kann jede Komponentendefinition eine beliebige Anzahl von sog. Varianten enthalten. Die Entscheidung, welche dieser Varianten auszuwählen und in die zu erzeugende Web-Präsentation einzufügen ist, wird während der Dokumentgenerierung durch Transformatoren (siehe Abschnitt 6.1.5) getroffen, die bestimmte Auswahlmethoden implementieren (Selektion, Konfiguration und Umordnung; anwendungsspezifische Methoden oder Kombinationen von Basismethoden sind jedoch ebenfalls möglich).

Während die vorgestellten inhaltlichen Anpassungen vom Autor während der Erstellung der Präsentation einzubinden sind, können Anpassungen an technische Parameter des Endgeräts zum großen Teil automatisch vorgenommen werden. Ein Beispiel dafür ist die Layoutdefinition auf der Basis des Konzeptes der JAVA-Layoutmanager [Java03], die eine geräteunabhängige Beschreibung der räumlichen Abhängigkeiten zwischen den einzelnen Inhaltskomponenten ermöglicht.

5. Abbildungsmechanismen

In Kapitel 4.2 wurde die im COMQUAD-Projekt verwendete Spezifikationssprache CQML$^+$ vorgestellt, mit der die Dienstgüteeigenschaften einer Komponente spezifiziert werden. Um die geforderte bzw. angebotene Dienstgüte einer Komponente durchsetzen zu können, sind Abbildungen von QoS-Spezifikationen, welche auf einer hohen Abstraktionsebene ausgedrückt werden (z.B. Bildqualität), auf nichtfunktionale Eigenschaften niedrigerer Abstraktionsebenen (z.B. Bildrate oder Farbtiefe) notwendig. Diese Abbildung muss schließlich bis auf die Ebene der Containerschnittstelle bzw. des zugrunde liegenden Echtzeitbetriebssystems durchgeführt werden, da die Zusicherung von Eigenschaften nur bezüglich der Ressourcen, die der Container anbietet, durchgeführt werden kann (etwa Speicher, CPU oder Netzwerkbandbreite).

Die Mechanismen zur Abbildung von nichtfunktionalen Eigenschaften sind eng mit dem in Kapitel 4 vorgestellten Entwicklungsprozess einer komponentenbasierten Anwendung mit nichtfunktionalen Eigenschaften verzahnt. Bei der Dekomposition einer Anwendung werden durch die Abbildung der an die Anwendung insgesamt gestellten nichtfunktionalen Eigenschaften neue Anforderungen erzeugt, welche die jeweiligen Einzelkomponenten erfüllen müssen, um die vom Anwender geforderte Dienstqualität erreichen zu können. Im Beispiel des Börsentickers kann die vom Anwender geforderte Antwortzeit von maximal 30 Sekunden bezüglich des `SubscriptionManagers` bedeuten, dass die von dieser Komponente zur Diensterbringung benötigten anderen Komponenten wie die Datenbank oder der `StockquotationDistributor` entsprechende maximale Antwortzeiten einhalten müssen. Derartige

Abbildungen sind meistens nicht eindeutig und werden gegebenenfalls auch mit Hilfe von Heuristiken durchgeführt. Im Beispiel könnte die maximale Antwortzeit theoretisch beliebig auf die aufgerufenen Komponenten „aufgeteilt werden“, d.h. sowohl eine Forderung, dass die Antwortzeit der Datenbank maximal 1 Sekunde betragen darf und die Abarbeitung des `subscribe()`-Aufrufs am `StockQuotationDistributor` 29 Sekunden dauern darf (hierbei werden zur Vereinfachung Nachrichtenlaufzeiten vernachlässigt) als auch etwa die gleichmäßige Aufteilung auf jeweils 15 Sekunden wären zulässig.

Eine sinnvolle Abbildung, welche zu einer lauffähigen Anwendung führt, die die Forderungen einhalten kann, kann jedoch nur durch eine genaue Kenntnis bzw. Beschreibung der jeweiligen Komponenten erfolgen. Weiß man etwa, dass die Datenbank unabhängig von den Daten und der verwendeten Rechnerplattform eine Antwortzeit von 1 Sekunde niemals anbieten kann, so können bestimmte Abbildungen gleich zu Beginn ausgeschlossen werden. Weitere Kenntnisse, wie etwa Erfahrungswerte der Zusammensetzung einer Antwortzeit (z.B. 60 % der Antwortzeit werden im Durchschnitt von der Datenbank benötigt) aus bereits bekannten Anwendungsszenarien, können helfen, eine entsprechende Abbildung zu finden.

Neben der Abbildung von nichtfunktionalen Eigenschaften auf von anderen Komponenten zu erfüllende nichtfunktionale Eigenschaften muss auch noch eine Abbildung der jeweiligen Eigenschaften auf den hierfür benötigten Ressourcenbedarf einer Komponente erfolgen. Diese Daten müssen beim Deployment einer Komponente bekannt sein und können etwa durch Messungen ermittelt werden. Dann können bei Erstellung einer Anwendung die notwendigen und letztendlich vom Betriebssystem zu reservierenden Ressourcen schrittweise ermittelt werden. So können möglicherweise bestimmte Anforderungen bzw. Konfigurationen einer Anwendung schon frühzeitig ausgeschlossen werden; wird beispielsweise ermittelt, dass die Anforderung an die Datenbank, maximal 1 Sekunde für die Bearbeitung der Anfrage zu benötigen, einen sehr hohen Speicherbedarf zur Erfüllung dieser Anforderung zur Folge hätte, da hierzu sehr viel Information im Hauptspeicher vorgehalten werden müsste, muss eine andere Abbildung gesucht werden. Ebenso können Anforderungen an das Netzwerk bezüglich verfügbarer Bandbreite abgeleitet und bei der Entwicklung somit überprüft werden, ob von der Zielplattform vorgegebene Ressourcenbeschränkungen eingehalten werden können. Dies ist im Extremfall bis auf die Ebene von Methodenaufrufen zu betrachten, da bei verteilten Anwendungen ja dadurch immer auch Netzbandbreite benötigt wird.

In einem ersten Schritt wurde mit der Betrachtung der erzeugten Last bei Aufruf eines Dienstes bzw. einer Methode, die eine Komponente anbietet, die vermutlich einfachste Art einer Abhängigkeit zwischen Komponenten untersucht. Dabei steht die Frage im Vordergrund, wie viele Aufrufe an den Schnittstellen der von einer Komponente zur Erbringung eines Dienstes selbst benötigten Komponenten ein Methodenaufruf an ihrer eigenen Schnittstelle erzeugt. Die Kenntnis dieses Zusammenhangs ist natürlich ein wichtiger Bestandteil der Beschreibung einer Komponente bezüglich der von ihr importierten Schnittstellen. Betrachtet man nochmals das Beispiel des Börsentickers, so ist die oben dargestellte Abbildung der Antwortzeit von maximal 30 Sekunden zur Vereinfachung nur auf der Betrachtungsebene der gesamten anderen Komponenten erfolgt. In Abbildung 3 erkennt man, dass an die Datenbank neben einem `read()`- auch ein `write()`-Aufruf abgesetzt wird. Eine nun bestimmte Antwortzeit für die Datenbank insgesamt muss natürlich entsprechend auf diese konkreten Methodenaufrufe bezogen werden, und hierzu ist es notwendig, zu wissen, ob durch einen `subscribe()`-Aufruf beispielsweise ein `read()`-Aufruf an die Datenbank oder eine Vielzahl davon abgesetzt werden.

In Abbildung 11 wird ein derartiger Aufrufzusammenhang exemplarisch ohne Bezug auf eine konkrete Anwendung dargestellt (ein Aufruf eines speziellen Dienstes an Komponente C1 erzeugt *x* Aufrufe an Komponente C2, *y* Aufrufe an C3 sowie *z* an Komponente C4). Die Ab-

bildung eines solchen Aufrufverhältnisses könnte mittels einer Matrix $A = (a_{jk})$ formuliert werden (wobei a_{jk} die Anzahl der Aufrufe der verwendeten Methode f_k bei Aufruf der angebotenen Methode f_j angibt. Hierbei ist natürlich noch keinerlei Parameterabhängigkeit der betrachteten Methoden enthalten, sondern es wird davon ausgegangen, dass das spezifizierte Aufrufverhältnis vom konkreten Aufruf und den übergebenen Parametern der jeweiligen Methode f_j unabhängig ist). Die konkreten Werte können dann mittels einer einfachen Multiplikation mit einem „Lastvektor“ (dieser gibt für die Komponente an, wie oft ein Dienst dieser Komponente pro Zeiteinheit aufgerufen wird) für die Komponente bestimmt werden.

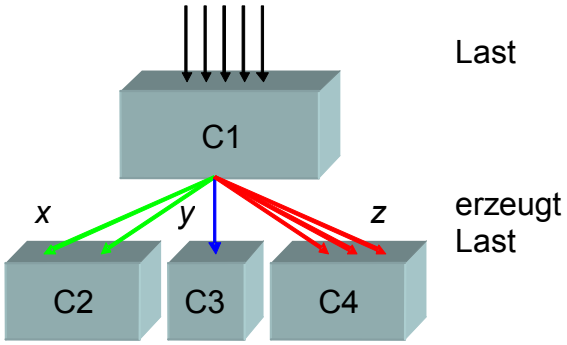

Abbildung 11: Zusammenhang zwischen Aufruf und erzeugter Last

Dies bedeutet, dass mit der Kenntnis eines solchen Zusammenhangs bestimmt werden kann, welche Kosten ein Dienstaufruf an C1 verursacht bzw. welche Anforderungen an das System zu stellen sind, wenn auch die entsprechenden Kosten für die Komponenten C2 bis C4 bezüglich der aufgerufenen Methoden bekannt sind. Versuche mit der Sprache CQML haben gezeigt, dass derartige, konzeptionell simple Zusammenhänge zwar prinzipiell in CQML selbst formuliert werden können bzw. die implizite Abbildung mittels CQML ausgedrückt werden kann. Allerdings ist dies nur für konkrete Werte möglich, eine Parametrisierung (etwa falls die erzeugte Last von den zu verarbeitenden Daten abhängt) oder eine Spezifikation über eine QoS-Charakteristik, wie z.B. Aufrufrate (mit der Einheit Aufrufe/Zeiteinheit), ist dagegen nicht mehr möglich, da hier ein funktionaler Zusammenhang ausgedrückt werden muss.

Unter anderem führten diese Einschränkungen zu Ideen zur Erweiterung von CQML, um Abhängigkeiten bzw. Abbildungen explizit formulieren zu können (siehe Kapitel 4.2). Betrachtet man das Beispiel aus Abbildung 11 und nimmt an, dass die Aufrufe an C2 bis C4 nicht parallel abgesetzt werden, so kann man mit einer expliziten Darstellung eine nicht-funktionale Eigenschaft wie die Antwortzeit wesentlich einfacher ausdrücken. Intuitiv ist klar, dass die Antwortzeit für den aufgerufenen Dienst von C1 u.a. von den Antwortzeiten von C2 bis C4 abhängt (zusätzlich einer von C1 selbst benötigten Zeit für Berechnungen). Explizit kann dies nun in etwa so dargestellt werden (hier sind die erzeugten Lasten für C2 bis C4 bereits miteinbezogen):

$$response\_time(C1) = x \bullet response\_time(C2) + y \bullet response\_time(C3) + z \bullet response\_time(C4) + time(C1);$$

Ohne eine explizite Darstellung müsste diese Abbildung über Intervalle in den provides- und uses-Klauseln der entsprechenden Profile „nachgebildet“ werden. Durch eine explizite Darstellung wird es nicht nur möglich, Zusammenhänge zwischen Komponenten wie im ange-

sprochenen Beispiel zu spezifizieren, sondern es werden auch Beeinflussungen der Adaptionsmöglichkeiten des Systems erwartet, da es nun prinzipiell möglich ist, die Auswirkungen auf das System bei etwa einem Austausch einer Komponente zu bestimmen.

In weiteren Schritten sollen komplexere Abhängigkeiten untersucht und entsprechende Abbildungen gefunden werden. So müssen zusätzlich die von einer Komponente benötigten Ressourcen betrachtet werden sowie in weiteren Schritten Adaptivität und benötigte Mechanismen zur Abbildung von Sicherheitsanforderungen auf vom System angebotene Dienste, um die jeweilige Anforderung umzusetzen. Wie bereits erwähnt, wurde in einem ersten Schritt angenommen, dass etwa Aufrufabhängigkeiten zwischen Komponenten unabhängig von Zustand oder Parametern immer gelten. Bei einer zusätzlichen Betrachtung von Datenabhängigkeit werden sich die Abbildungen allerdings nicht mehr als einfache Operationen mit einer Matrix darstellen lassen. Da sich Zusammenhänge möglicherweise auch nicht immer eindeutig bestimmen lassen bzw. selbst vom Entwickler nicht komplett erkannt werden können, müssen zur Abbildung von quantitativen Eigenschaften auch entsprechende Heuristiken entwickelt und implementiert werden, die an geeigneten Komponenten zu validieren sind.

Allgemein ist das Finden bzw. Spezifizieren von Abbildungsfunktionen eine Aufgabe des Herstellers der jeweiligen Komponente. Dabei ist er durch entsprechende Werkzeuge zu unterstützen und kann auf einen Vorrat von „Standard-Abbildungen“ zurückgreifen (z.B. kann die „Definition“ einer Abbildung das „Ausfüllen“ einer Matrix bedeuten, um entsprechende Abhängigkeiten zu importierten Schnittstellen anzugeben), um nicht bei jeder neuen Komponente von Grund auf eine Abbildung definieren zu müssen.

Die bereits erwähnte Verzahnung mit dem Entwicklungsprozess einer Anwendung bietet eine Integration der Konzepte in das Entwurfswerkzeug von COMQUAD an. Derzeit wird hierzu unter anderem an geeigneten Schnittstellen für die Anbindung eines Abbildungsmoduls gearbeitet.

6. Laufzeitumgebung

6.1. Konzept

COMQUAD-Laufzeit-Komponenten sind abgeschlossene Softwarebausteine, die kooperieren, um eine Anwendungsaufgabe zu lösen. Dabei existieren sie innerhalb eines Containers, der die konkreten Einzelheiten der Plattform vor den Komponenten verbirgt. Dieser Container ist in sich in verschiedene Schichten strukturiert. es gelten die strengen Aufrufregeln der Schichtenarchitektur, nämlich, dass Aufrufe jeweils nur an die direkt untergeordnete Schicht abgesetzt werden, während Ergebnisse und Rückaufrufe jeweils nur an die direkt übergeordnete Schicht erfolgen.

Abbildung 12 zeigt die wesentlichen Schichten im Container. Es gibt also eine Trennung zwischen (Software-) Komponenten und Ressourcen. Diese müssen fundamental unterschiedlich behandelt werden. Während Anforderungen an Komponenten in der Komponentenverwaltung ausgehandelt werden, müssen Ressourcenanforderungen an die Ressourcenverwaltung weitergereicht werden. Andererseits ist es für die Ressourcenverwaltung völlig unwichtig, wie Komponenten aussehen und verwaltet werden, ja sogar die Tatsache, dass es Komponenten gibt.

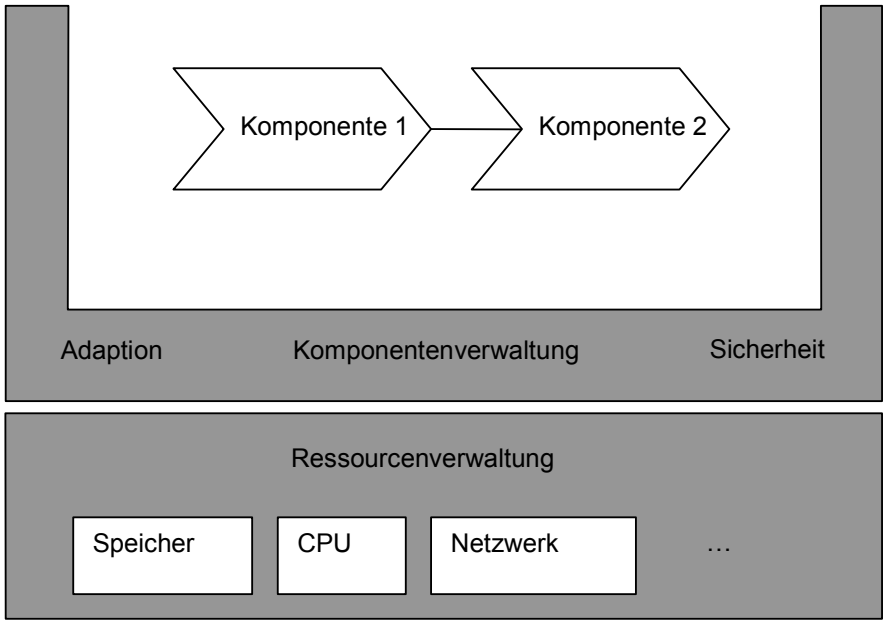

Abbildung 12: Konzeptionelle Übersicht über die Laufzeitumgebung

Das Programmiermodell für COMQUAD-Komponenten orientiert sich in etwa an Enterprise Java Beans (EJB) Session Beans, bzw. CORBA Session Komponenten, denen es aber noch Konzepte wie strombasierte Kommunikation (über schwankungsbeschränkte Ströme) und explizite Deklaration der verwendeten Schnittstellen und Komponenten hinzufügt. Für die Beschreibung der Beziehungen zwischen Komponenten ist dabei noch eine geeignete Spezifikationssprache zu finden oder zu entwickeln (siehe dazu auch Abschnitt 4.1).

Im Folgenden werden die Funktionen der einzelnen Bestandteile der Laufzeitumgebung näher beschrieben.

6.1.1. Komponentenverwaltung

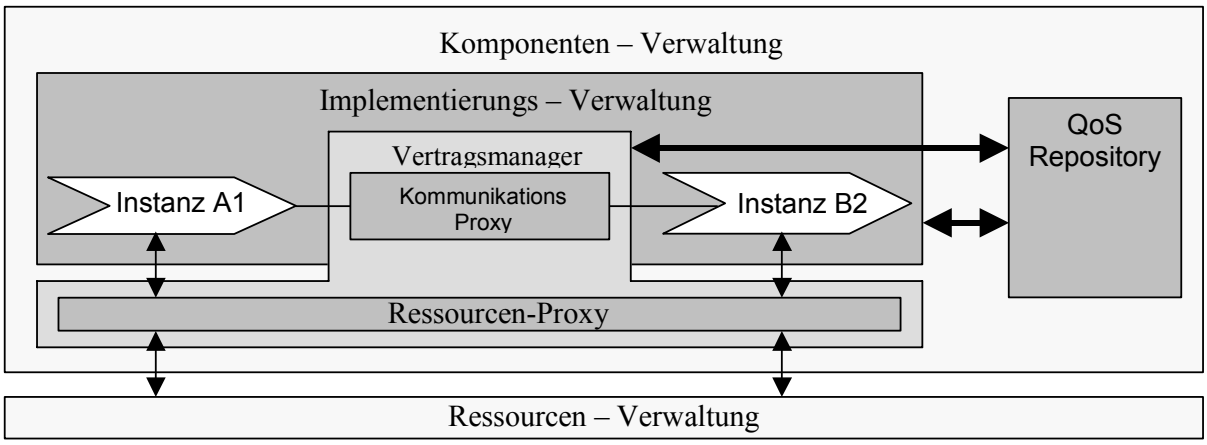

Abbildung 13: Grobstruktur der Komponentenverwaltung

Die Komponentenverwaltung unterstützt Komponenten, die sich in ihrer Struktur nahe an EJB anlehnen. Sie gliedert sich in drei große Bereiche (s. Abbildung 13):

1. Implementierungsverwaltung
2. QoS-Repository
3. Vertragsmanager

Die Implementierungsverwaltung dient im Wesentlichen der Verwaltung der auf dem Server installierten Komponenten. Dabei werden pro funktionaler Spezifikation potentiell mehrere

Implementationen verwaltet, die dieselbe Funktion mit unterschiedlicher QoS umsetzen (s. Abbildung 14).

Bei jedem Request, den ein Client an den Server übermittelt, wird ein Netz von Komponenten instantiiert, das die Anfrage bearbeiten kann. Dabei können natürlich verschiedene Caching-Algorithmen eingesetzt werden, um den Instanziierungsaufwand zu minimieren, indem bereits instanziierte Komponenten für weitere Clients wiederverwendet werden. Die Instanziierung und Verknüpfung der Instanzen zu Komponentennetzen ist die Aufgabe des Vertragsmanagers. Dieser verwendet die funktionalen und nichtfunktionalen Spezifikationen der Komponenten, um zueinander passende Komponentenimplementationen auszuwählen und zu instanziieren. Die Ressourcenanforderungen der einzelnen instanziierten Komponenten (ebenfalls in der nichtfunktionalen Spezifikation enthalten) werden als Ressourcenreservierung and die Ressourcenverwaltung weitergereicht. Zunächst ist bei der Vertragsaushandlung an einen naiven Backtracking-Algorithmus gedacht, der aber noch optimiert werden soll.

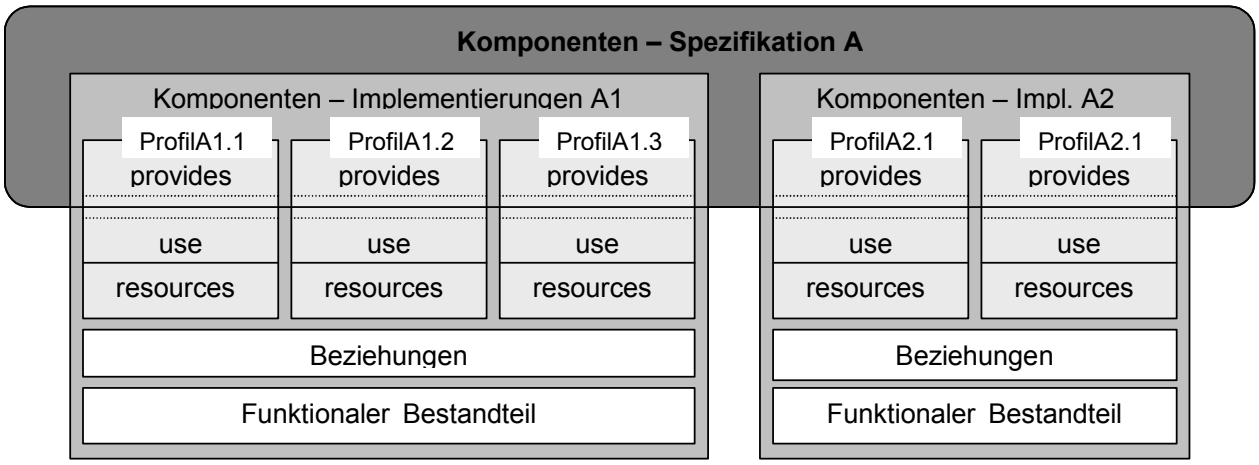

Abbildung 14: Komponentenspezifikation und mehrere Implementierungen

Zur Durchsetzung der ausgehandelten Verträge zwischen den Komponenten instanziiert der Vertragsmanager Kommunikations-Proxies, die sämtliche Kommunikationsvorgänge zwischen den Komponenteninstanzen kontrollieren. Die Kommunikation der Komponenten mit der Ressourcenverwaltung (beispielsweise um eine reservierte Ressource tatsächlich zu benutzen) wird von den — ebenfalls vom Vertragsmanager instanziierten — Ressourcen-Proxies übernommen.

Außerdem enthält die Komponentenverwaltung ein QoS-Repository, das die nichtfunktionalen Spezifikationen der einzelnen Komponenten verwaltet. Diese Spezifikationen werden dem Container in Form eines XML-Deskriptors übergeben, der beim Deployment bzw. beim Starten des Containers gelesen und in das QoS-Repository übertragen wird. Ebenfalls durch einen XML-Deskriptor werden dem Container Informationen über die funktional zulässigen Verknüpfungen von Komponenten mitgeteilt.

6.1.2. Sicherheitsmechanismen

Die bekannten kryptographischen Systeme können für folgende Zwecke eingesetzt werden (Tabelle 2):

	Kommunikationsinhalte	Kommunikationsumstände
Vertraulichkeit	symmetrische und asymmetrische Konzelationssysteme	— *Nutzung spezieller Dienste erforderlich*
Integrität	symmetrische und asymmetrische Authentikationssysteme	asymmetrische Authentikationssysteme (digitale Signaturen)

Tabelle 2: Einsatz kryptographischer Systeme

Die aufgeführten Mechanismen sollten den Komponenten im Container zur Verfügung stehen und von den Anwendungen getrennt werden. Damit wird die Wiederverwendbarkeit der kryptographischen Bausteine unterstützt und der Einsatz sicherer, wohl untersuchter Mechanismen kann vorgesehen werden. Die Anwendungen sind unabhängig von den Mechanismen, welche damit leichter austauschbar bzw. aktualisierbar sind. Außerdem erleichtert dieser Ansatz die Einigung zwischen verschiedenen Containern auf einen Mechanismus.

Neben der Existenz der reinen Verschlüsselungs- und Authentikationsalgorithmen sind weitere Mechanismen in die Laufzeitumgebung zu integrieren, z. B. Hashfunktionen und eine PKI (*Public Key Infrastructure)* zur Verwaltung öffentlicher Schlüssel und Zertifikate.

Für die Realisierung mehrseitiger Sicherheit bzgl. der Kommunikation (und damit den Einsatz oben genannter Mechanismen) sollen existierende Lösungen aus dem Projekt SSONET [SSONET] in den Container integriert werden. Das Projekt SSONET beschreibt eine Sicherheitsarchitektur, welche die Teilnehmer an dem verteilten Szenario dabei unterstützt, ihre Sicherheitsinteressen zu formulieren und entstehende Konflikte mit anderen Teilnehmern auszuhandeln. Erfolgte eine Einigung auf die Durchsetzung bestimmter Schutzziele, wird die Erfüllung der Anforderungen nach Vertraulichkeit, Integrität und Zurechenbarkeit durch Anwendung der entsprechenden Mechanismen erreicht (Annahme: lokale Systeme sind sichere Umgebung).

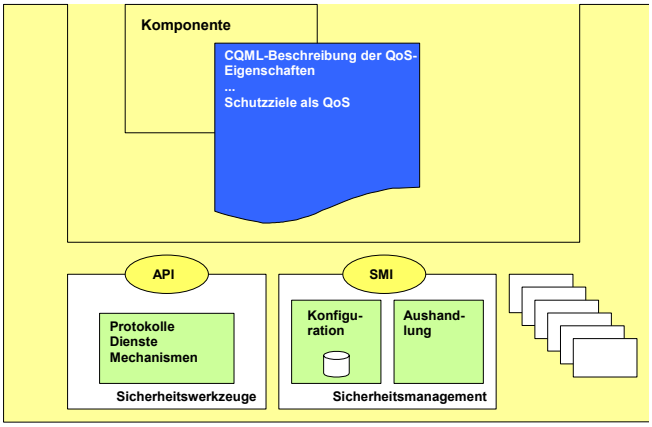

Abbildung 15: Integration der SSONET-Konzepte in die Laufzeitumgebung – erster Überblick

Die aus dem Projekt SSONET vorhandenen Konzepte müssen angepasst werden, da die für die Erfüllung weiterer nicht-funktionaler Anforderungen erforderliche Reservierung von Ressourcen zu berücksichtigen ist. Demzufolge ist nicht nur ein Kompromiss zwischen u. U. gegensätzlichen Sicherheitsanforderungen verschiedener Nutzer, sondern auch zwischen Anforderungen nach Performance und Sicherheit ein und desselben Nutzers auszuhandeln. Diese Integration wird im Rahmen einer Diplomarbeit realisiert.

6.1.3. Ressourcenverwaltung

Bei der Betrachtung der Struktur der Laufzeitumgebung versuchen wir, Konzepte, welche für Komponenten im Container angewandt werden, auf die Ressourcenverwaltung zu übertragen. So setzt sich z.B. die Ressourcenverwaltung selber wieder aus Komponenten zusammen. Wir versuchen zudem, ähnliche Funktionen mit ähnlichen Komponenten zu erfüllen.

Die zentrale Instanz der Ressourcenverwaltung ist der QoS-Manager. Für jede Ressource, welche die Reservierung von Quantitäten ihres Dienstes zulässt, existiert zudem ein Ressourcenmanager. Ressourcenmanager sind hierarchisch organisiert wie in [Här99] beschrieben. Diese Ressourcenmanager sind dem QoS-Manager bekannt mit dem Ressourcentyp, welchen sie verwalten, und ihrer Adresse. Benötigt der Container oder ein anderer Client der Ressour-

cenverwaltung eine bestimmte Quantität einer Ressource, wendet er sich an den QoS-Manager und versucht, diese Quantität zu reservieren.

Um den QoS-Manager so allgemein wie möglich zu halten, sind Anfragen in nur zwei Teile unterteilt. Der erste kann vom QoS-Manager verstanden werden und enthält den benötigten Ressourcentyp. Anhand dieses Typs kann der passende Ressourcenmanager gefunden werden. Der zweite Teil ist für den QoS-Manager unverständlich und enthält eine für die Ressource spezifische Beschreibung der benötigten Quantität und kann nur von dem entsprechenden Ressourcenmanager verstanden werden.

Ressourcen können unterschieden werden in Ressourcen, welche andere Ressourcen benutzen, um ihren Dienst erbringen zu können, und Ressourcen, welche keine weiteren Ressourcen benötigen. Andererseits sind Ressourcen unterscheidbar darin, ob für jeden Client eine Instanz dieser Ressource erzeugt wird oder ob alle Klienten mit der gleichen Instanz kommunizieren.

Stellt der QoS-Manager eine Anfrage an den Ressourcenmanager einer Ressource, welche weitere Ressourcen benutzt, dann erhält er als Antwort die Quantitäten der benötigten Ressourcen. Diese neuen Anfragen müssen reserviert sein bevor für die ursprüngliche Ressource eine Zusage getroffen wird und der QoS-Manager die Anfrage an den Client bestätigen kann.

Ressourcenanfragen können Alternativen enthalten. Die Alternativen sind gewichtet nach den Wünschen des Client. Ein Ansatz zur Reservierung solch einer Anfrage ist, zu versuchen, die wichtigste Alternative zu reservieren. Schlägt diese Reservierung fehl, wird die nächste Alternative ausgewählt. Ein anderer Ansatz ist, dass der QoS-Manager erst alle Alternativen vorreserviert und dann eine globale Auswahlstrategie anwendet, um unter den verschiedenen Alternativen mit ihren erst jetzt detailliert bekannten Ressourcenanforderungen auszuwählen. Ist die Wahl auf eine Alternative gefallen, wird diese reserviert und die anderen Vorreservierungen frei gegeben (siehe [Noth02]).

Damit der QoS-Manager mit allen Ressourcenmanagern kommunizieren kann, müssen diese eine allgemein gültige *Admission* Schnittstelle implementieren. Diese enthält Funktionen zur Reservierung, Vorreservierung und Freigabe von Ressourcen. Damit die Schnittstelle für verschiedenste Ressourcentypen benutzt werden kann, definiert sie als Eingabeparameter eine Zeichenkette, welche den ressourcenspezifischen Teil der Reservierung enthält. Zurück geliefert wird ebenfalls eine Zeichenkette, welche den eigenen Bedarf an Ressourcen enthalten kann oder den Identifikator für die erfolgte Reservierung.

Die Auswahl der passenden Ressourcenmanager zu einer Reservierungsanfrage im QoS-Manager kann verbessert werden. Zur Zeit werden zu einer Reservierungsanfrage alle Ressourcenmanager herausgesucht, welche den angefragten Ressourcentyp anbieten. Der QoS-Manager muss jeden fragen, ob er die gewünschte Reservierung erfüllen kann. Um die Anzahl der Kommunikationen zu verringern, soll der QoS-Manager bereits eine Vorauswahl der Ressourcenmanager treffen. Dazu muss dem QoS-Manager jeder Ressourcenmanager eine Liste von Eigenschaften, der von ihm angebotenen Ressource nennen. Anhand dieser Liste und der geforderten Eigenschaften der Anfrage kann der QoS-Manager die aktuell zu durchsuchende Liste der Ressourcenmanager einschränken. Um diese Überprüfung durchführen zu können, definieren wir eine Eigenschaft als Tupel aus *Name, Typ, Einheit und Wertebereich*. Zum Beispiel kann die Eigenschaft „Größe“ der Ressource „Speicher“ definiert sein als { „size“, Integer, KB, (0-1000]}. Kommen Anfragen nach mehr als 1000KB, gibt es keine Ressourcenmanager, welche diese Eigenschaft anbieten. Damit kann der QoS-Manager diese Anfrage sofort zurückweisen.

6.1.4. Systemseitige Adaption

Im Abschnitt 4.4 wurden komposite Komponenten als ein Konzept zur transparenten Kapselung von Strukturadaptionen in Komponentennetzen vorgestellt. Die Implementierung dieses Konzepts wird in enger Verzahnung mit der Ressourcenverwaltung in Form eines Containergerüstes für einen weiteren Komponententyp realisiert werden. Derzeit werden verschiedene Meta-Programmierkonstrukte zur transparenten Realisierung untersucht und gegenübergestellt, insbesondere Aspekt-orientierte Programmierwerkzeuge als Integrationsmittel zur Entwicklungszeit, sowie *Interceptoren* als Laufzeit-Konstrukte [PoGö03].

Um über Veränderungen im Angebot von Diensten, speziell von Ressourcen, zu wachen, können verschiedene Strategien angewendet werden. Bei einer verteilten Strategie überträgt man die Verantwortung jeder einzelnen Komponente. Diese ist dafür verantwortlich, ihre eingegangenen Kontrakte zu überwachen und festzustellen, ob sie diese erfüllt oder nicht. Ein Nachteil dieser Herangehensweise ist, dass eine Ressource bewusst gegen Verträge verstoßen kann. Der Vorteil ist, dass die Ressource ihr Angebot am besten kennt.

Ein zentraler Ansatz überträgt diese Verantwortung dem Container bzw. der Laufzeitumgebung. Diese muss die Ressourcen überwachen, welche Verträge mit Komponenten oder anderen Ressourcen abschließen (diesen Vertragsabschluss führt der Container u.U. sogar selber durch). Dazu kann es notwendig sein, dass der Container über jede Ressource und Komponente explizites Wissen hat, um eine Verletzung feststellen zu können. Ein Vorteil dieses zentralen Ansatzes ist die Möglichkeit, eine globale Entscheidung treffen zu können, ob die Vertragsverletzung wirklich einer Benachrichtigung bedarf. Beispielsweise kann eine Komponente Rechenzeit von 100 ms pro Sekunde reserviert haben. Dann kommt es einmal vor, dass sie stattdessen 110 ms arbeitet, was ein Verstoß gegen ihren Vertrag ist. Befindet sich die CPU aber gerade im Leerlauf, kann diese Überschreitung geduldet werden.

Die Benachrichtigung über eine Verletzung des Vertrages erfolgt über die genannte Notification Schnittstelle. Durch einen Aufruf der Schnittstelle weiß die Komponente, dass eine Verletzung ihres Vertrages vorliegt. Durch das mitgelieferte Handle – welches sowohl die benutzende Komponente als auch die benutzte Ressource kennen – kann der verletzte Vertrag identifiziert werden. Zudem wäre es möglich, dass die Ressource der Komponente eine neue Quantität anbietet. Die Komponente kann nun entscheiden, ob sie sofort akzeptiert oder ablehnt. Dazu muss die Komponente gegebenenfalls von ihr abhängige andere Komponenten konsultieren.

6.1.5. Anwendungsseitige Adaption

Die anwendungsseitige Adaption multimedialer Web-basierter Anwendungen, die auf der Basis des in Kapitel 4.5.1 vorgestellten Modells der Dokumentkomponenten entwickelt wurden, soll Thema dieses Kapitels sein. Dabei soll der schrittweise Prozess der anwendungsseitigen Adaption multimedialer Inhalte an inhaltliche und technische Benutzerpräferenzen ebenfalls anhand des Börsenpräsentations-Beispieles (Kapitel 2) vorgestellt werden.

Die Dokumentgenerierung auf der Basis eines Pipeline-Konzepts ermöglicht die schrittweise Transformation von Dokumentkomponenten zu einem konkreten Ausgabeformat. Der Vorteil der Aufteilung des Transformationsprozesses in mehrere Schritte liegt in der Möglichkeit der Wiederverwendung von Transformationskode sowie einer größeren Performanz durch das Einbinden effizienter Caching-Mechanismen.

Im ersten Schritt wird das vom Anwender angeforderte Börsen-Dokument aus einem Komponenten-Repository geladen. In ihm ist eine Beschreibung des zu präsentierenden Web-Inhaltes abgelegt. Diese Beschreibung enthält nicht nur Informationen über die einzubindenden Unterkomponenten, sondern auch die für die Adaption benötigten Metadaten. Das bedeutet, dass

alle möglichen, d.h. vom Autor vorgesehenen, Präsentationsvarianten bzgl. Inhalt, Darstellung und Struktur in diesem Dokument enthalten sind.

Abhängig vom Benutzermodell, das sowohl die inhaltlichen Vorlieben des Anwenders, als auch die technischen Eigenschaften seiner Systemumgebung enthalten kann, wird auf das Dokument eine Reihe von Transformationen angewendet. Im letzten Transformationsschritt schließlich wird das bis dahin angepasste Dokument in das benötigte Ausgabeformat, z.B. XHTML, cHTML oder WML umgewandelt. Die Pipeline-basierte Architektur des Dokumentgenerators ist in Abbildung 16 dargestellt.

Die Transformation des Börsendokuments bis hin zur adaptierten Version in das für den Benutzer passenden Ausgabeformat geschieht in vier Schritten:

1. Anpassung der zu präsentierenden Informationsart und –menge an die vorhandene Endgeräteklasse, z.B. Desktop-PC, PDA oder Handy,
2. Anpassung an persönliche Benutzervorlieben, z.B. mediale Präferenzen (Text vs. Video),
3. Anpassung an konkrete Systemparameter, z.B. Videoqualität in Abhängigkeit von der verfügbaren Netzwerkbandbreite,
4. Umwandlung in das benötigte Ausgabeformat, z.B. HTML, cHTML oder WML.

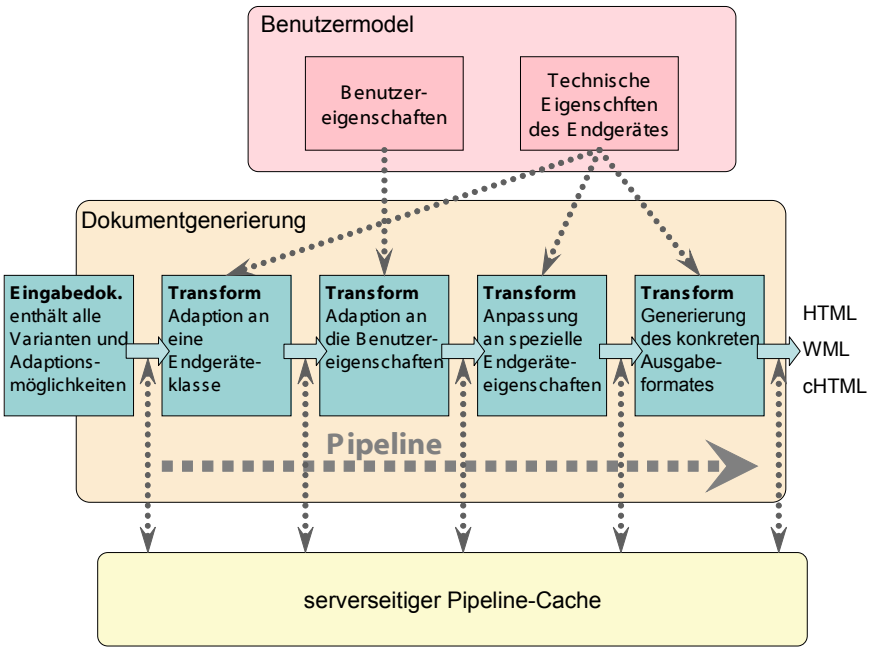

Abbildung 16: Pipeline-basierte Generierung von Dokumenten

6.2. Prototypen

Nachdem sich die bisherige Arbeit auf dem Gebiet der Laufzeitumgebung vor allem auf die Konzepte konzentrierte, wird es in den nächsten Monaten eine Hauptaufgabe im Projekt sein, diese in einem Prototypen umzusetzen. Aufgrund der großen Modularität und leichten Anpassbarkeit (*open source*) ist dabei daran gedacht, den Applikationsserver JBoss an unsere Bedürfnisse anzupassen [JBoss].

Wie in Abbildung 17 dargestellt, beruht die Modularität des Applikationsservers JBoss auf einer JMX[1] Microkernel Architektur. Einzelne Middleware-Dienste werden über dieses Kon-

1 Java Management eXtension: Final Specification Request http://jcp.org/en/jsr/detail?id=003

zept zur Start- oder Laufzeit des Servers / Containers dynamisch konfigurierbar hinzugebunden und über definierte Managementschnittstellen verwaltet. In der Abbildung ist ebenfalls dargestellt, dass sich die geplanten Anpassungen vor allem auf Modifikationen der eigentlichen Container-Architektur, sowie der Sicherheits- und Namensdienste beziehen, wohingegen Transaktionen, Datenbanken, Messaging und Web-Container von den Änderungen zunächst nicht betroffen sind und deshalb vorerst ausgeklammert werden sollen.

Parallel sollen jedoch auch andere Ansätze untersucht werden, beispielsweise die Verwendung aspektorientierter Programmierung zur Umsetzung der Container-Funktionen.

Eine weitere im Rahmen der prototypischen Umsetzung der Laufzeitumgebung zu lösende Aufgabe ist die Definition einer programmatischen Schnittstelle für die Realisierung strombasierter Kommunikation auf der Basis schwankungsbeschränkter Ströme und die Umsetzung dieser Schnittstelle auf verschiedene existierende Kommunikationsprotokolle (DSI, ATM, RTSP, …).

Zwei Richtungen werden dabei zunächst getrennt berücksichtigt: Die Komponentenverwaltung mit Sicherheitsaspekten und Anbindung an die Ressourcenverwaltung einerseits und die Unterstützung von Adaption andererseits. Das liegt daran, dass für die Unterstützung von Adaption noch wesentlich mehr konzeptionelle Arbeit zu leisten ist, bevor an eine Umsetzung gedacht werden kann. Die Umsetzung wird aber ebenfalls auf Basis von JBoss erfolgen und eng in die bereits entwickelte Laufzeitumgebung eingebunden werden.

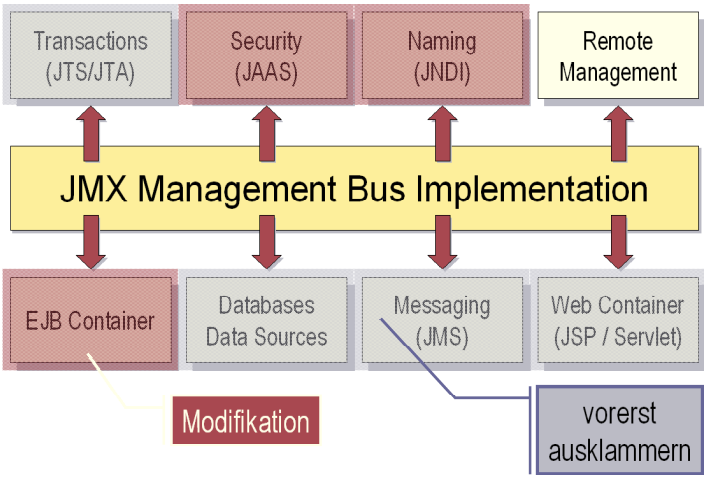

Abbildung 17: JBoss JMX Architektur mit Kennzeichnung der nötigen Anpassungen

Eine weitere Trennung existiert im Moment zwischen Laufzeitumgebung und Ressourcenverwaltung, wie bereits in Abbildung 12 dargestellt. Die geschichtete Architektur hat sich für Systeme dieser Komplexität bewährt. In COMQUAD hilft diese Architektur unter anderem, damit umzugehen, dass die Ressourcenverwaltung in DROPS (Dresden Real-time Operating System [Bau98]) integriert ist, wofür jedoch zur Zeit noch keine *Java Virtual Machine* (JVM) existiert. Hier wurde eine Schnittstelle definiert, über die die beiden Schichten miteinander kommunizieren. Dadurch können die Bereiche zunächst getrennt betrachtet werden und es wird dennoch sichergestellt, dass sie später leicht zusammenarbeiten können. Allerdings können auf der Ebene der Komponentenverwaltung nur Anforderungen gestellt werden, die auch durch die Ressourcenverwaltung durchsetzbar sind.

Im Rahmen der Umsetzung des Börsenpräsentations-Szenarios ist vorgesehen, primär die serverseitig für die Dokumentgenerierung benötigten Funktionen als COMQUAD-Komponenten zu realisieren. Dazu wird in einem ersten Schritt zunächst die Benutzermodellierung umgesetzt. Später ist geplant, auch andere Teile des Dokumentgenerators, die zunächst auf der Basis des Publishing-Frameworks COCOON [Coc03] realisiert werden, sowie einen einfachen Medienserver als COMQUAD-Komponenten zu implementieren.

7. Zusammenfassung

In diesem Bericht wurden zunächst wichtige Teilziele für das Projekt COMQUAD genannt und die wesentlichen Probleme identifiziert.

So wurde die Spezifikationssprache CQML$^+$ entwickelt, welche es ermöglicht, nicht-funktionale Eigenschaften von Komponenten zu spezifizieren. Zudem können mit Hilfe der Abbildungsmechanismen die durch CQML$^+$ bereitgestellten quantitativen Eigenschaften auf den Bedarf der Komponente an quantitativen Eigenschaften und Ressourcen abgebildet werden. Eine weitere Eigenschaft von CQML$^+$ ist die Möglichkeit, den Ressourcenbedarf einer Komponente zu spezifizieren und – durch Nutzung von *profiles* – einfache Adaptionen darzustellen.

Eine genaue Entwicklungsmethodik für Komponenten mit nicht-funktionalen Eigenschaften sowie die einzelnen Schritte bei der Entwicklung von COMQUAD-Komponenten und –Anwendungen konnten in Abschnitt 3 vorgestellt werden.

Es wurde eine Laufzeitumgebung – der Container und die Ressourcenverwaltung – vorgestellt, welche die Aushandlung und Durchsetzung der Verträge übernehmen. Mit Hilfe der beschriebenen Techniken können der Container und die Ressourcenverwaltung garantieren, dass geschlossene Verträge eingehalten werden. Ist dies nicht mehr möglich, werden die betroffenen Komponenten über die Vertragsverletzung informiert ggf. Adaptionsprozesse angestoßen.

Im Bericht wurde ein anschauliches Beispiel verwendet, um die einzelnen Aspekte der Arbeit in den verschiedenen Teilbereichen zu illustrieren.

8. Literatur

[Aag01] Jan Øyvind Aagedal. Quality of Service Support in Development of Distributed Systems. Doktorarbeit, Universität Oslo, 2001.

[Bau98] Baumgartl, R.; Borriss, M.; Härtig, H.; Hamann, Cl.-J.; Hohmuth, M.; Reuther, L.; Schönberg, S.; Wolter, J.: Dresden Realtime Operating System. In the proceedings of the Workshop of System-Designed Automation (SDA'98), March 1998, Dresden, Germany

[ChDa01] Cheesman, J.; Daniels, J.: UML Components. A Simple Process for Specifying Component-Based Software. Addison-Wesley. 2001

[Ceri+00] Ceri, S.; Fraternali, P.; Bongio, A.: Web Modeling Language (WebML): a modeling language for designing web sites. In: Proceedings of the 9th International Conference on the WWW (WWW9), Amsterdam, May 2000

[Coc03] Cocoon Homepage, http://xml.apache.org/cocoon/

[Cor02] CORBA Components, Version 3.0. Object Management Group, Inc., June 2002

[Dash01] Dashofy, E.; van der Hoek, A.; Taylor, N.: A Highly-Extensible, XML-Based Architecture Description Language. In Proceedings of the Working IEEE/IFIP Conference on Software Architectures (WICSA 2001), Amsterdam, Netherlands, 2001

[Gaed+00] Gaedke, M.; Segor, C.; Gellersen, H.-W.: WCML: Paving the Way for Reuse in Object-Oriented Web Engineering. ACM Symposium on Applied Computing (SAC2000), Villa Olmo, como, Italy, 2000

[GaGr00] Gaedke, M.; Graef, G.: WebComposition Process Model: Ein Vorgehensmodell zur Entwicklung und Evolution von Web-Anwendungen. In: R.G. Flatscher, K. Turowski (Hrsg.): Tagungsband 2. Workshop Komponentenorientierte betriebliche Anwendungssysteme (WKBA 2), Wien, S. 21-38.

[Här99] H. Härtig, L. Reuther, J. Wolter, M. Borriss, T. Paul: Cooperating Resource Managers**,** In: Proceedings of Workshop on QoS Support for Real-Time Internet Applications, Vancouver, Canada, June 1999

[Ham01] Hamann, Löser, Reuther, Schönberg, Wolter, Härtig: Quality Assuring Scheduling - Deploying Stochastic Behavior to Improve Resource Utilization In: Proceedings of the 22th IEEE Real-Time Systems Symposium (RTSS-XXII), London, UK, Dezember 2001.

[Ham97] Hamann, Cl.-J.; On the Quantitative Specification of Jitter Constrained Periodic Streams In: Proceedings of MASCOTS' 97, Haifa (Israel) Januar 1997

[Isak+95] Isakowitz, T.; Stohr, E.A.; Balasubramanian, P.: RMM: A Methodology for Structured Hypermedia Design. Communications of the ACM 38, No. 8 (1995) Aug. 1995, S. 34-44.

[JaMe02] Jablonski, S.; Meiler, C. : Web-Content-Managementsysteme. In : Informatik Spektrum 25/2, Springer Verlag, 2002

[Java03] Creating a GUI with JFC/Swing. In: The Java Tutorial http://java.sun.com/docs/books/tutorial/uiswing/layout/using.html

[JBoss] JBoss Application Server; http://www.jboss.org/

[Jon83] Jones, C.: Specification and design of (parallel) programs. In R. E. A. Manson, editor. Proceedings of IFIP '83. IFIP, North-Holland, 1983, pages 321-332.

[Liu73] Liu, C. L.; Layland, J. W.: Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment. In: Journal of the ACM, Vol. 20(1), 1973, S. 46 - 61.

[Noth02] Nothnagel, J.: Ressourcenverwaltung in DROPS. Diplomarbeit. Juli 2002.

[PoGö03] Pohl, C., Göbel, S: Integrating Orthogonal Middleware Functionality in Components Using Interceptors. In Kommunikation in verteilten Syste-men (KiVS 2003); Informatik Aktuell, Springer Verlag, Februar 2003

[RöZ03] Röttger, Simone; Zschaler, Steffen: CQML$^+$: Enhancements to CQML, In Proceedings of the 1st International Workshop on Quality of Service in Component-Based Software Engineering, Cépaduès-Éditions, Toulouse, France, June 2003, pages 43-56.

[ScRo98] Schwabe, D.; Rossi, G.: An object-oriented approach to web-based application design. In: Theory and Practice of Object Systems (TAPOS), Special Issue on the Internet, v4#4, Oktober 1998

[SSONET] DRIM (Dresden Identity Management); http://drim.inf.tu-dresden.de/

[Szyp97] Szyperski, C.: Component Software –Beyond Object-Oriented Program-ming; Addison-Wesley, Harlow, England, 1997/98

[ZMay03] Zschaler, Steffen; Mayerhöfer, Marcus: Explicit Modelling of QoS-Dependencies, In Proceedings of the 1st International Workshop on Quality of Service in Component-Based Software Engineering, Cépaduès-Éditions, Toulouse, France, June 2003, pages 57-66.

Anhang A: IDL des Börsenticker-Beipiels

```
module Boersenticker {
  struct StockInfo {
    string stockName;
    float stockValue;
  };

  eventtype singleStockDataReport {
    public StockInfo data;
  };

  eventtype multiStockDataReport {
    public sequence<StockInfo> data;
  }

  module StockExchange {
    interface SubscribeUnsubscribe {
      void getAvailableStocks (out sequence<string> stocks);

      boolean subscribe (
          in string stockName,
          in Server::StockQuotationDistributor listener);

      void subscribe (
          in string stockName,
          in Server::StockQuotationDistributor listener);
    }

    component StockQuotationManager {
      publishes multiStockDataReport outgoingUpdates;
      provides SubscribeUnsubscribe subscriptions;
    }
  }

  module Client {
    component Display {
      consumes singleStockDataReport incomingUpdates;
    }
  }

  module Server {

    typedef long Handle;

    interface SubscribeUnsubscribe {
      void getAvailableStocks (out sequence<string> stocks);
      Handle subscribe (in string stockName,
                        in Client::Display viewer);

      void unsubscribe (in Handle subscription);
    }
```

```
    interface XferSubscriptions {
      boolean subscribe (in string stockName,
                         in Client::Display viewer);

      void unsubscribe (in string stockName,
                        in Display::setData viewer);
    }

    interface DataAccess {
      void read (in struct query,
                 out struct recordset);
      void write (in struct query,
                  in struct dataset);
      void delete (in struct query);
    }

    component SubscriptionManager {
      provides SubscribeUnsubscribe subscriptionInterface;
      uses XferSubscriptions distributor;
      uses DataAccess database;
    }

    component StockQuotationDistributor {
      publishes singleStockDataReport outgoingUpdates;
      consumes multiStockDataReport incomingUpdates;
      uses DataAccess database;
    }

    component Database {
      provides DataAccess dataManagement;
    }
  }

  module Client {
    component Controller {
      uses Server::SubscribeUnsubscribe subscriptionManager;
    }
  }
}
```