TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Informatik

TUD-FI02-10 - Nov. 2002

COMQUAD Forschergruppe

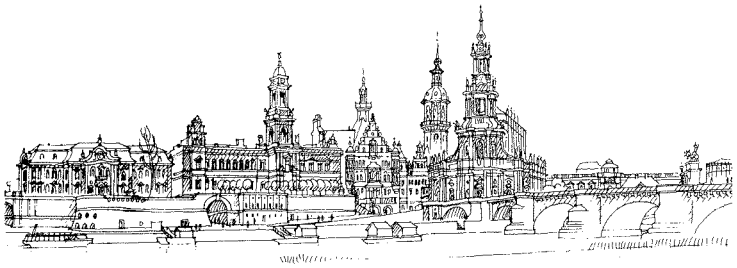
Institut für Systemarchitektur,
Institut für Softwaretechnik

COMQUAD – Komponentenbasierte Softwaresysteme mit zusagbaren quantitativen Eigenschaften und Adaptionsfähigkeit

Technische Berichte

Technical Reports

ISSN 1430-211X



Technische Universität Dresden
Fakultät Informatik
D-01062 Dresden
Germany

URL: http://www.inf.tu-dresden.de/

COMQUAD – Komponentenbasierte Softwaresysteme mit zusagbaren quantitativen Eigenschaften und Adaptionsfähigkeit

Ronald Aigner, Henrike Berthold, Elke Franz,
Steffen Göbel, Hermann Härtig, Heinrich Hußmann,
Klaus Meißner, Klaus Meyer-Wegener, Marcus Meyerhöfer,
Andreas Pfitzmann, Simone Röttger, Alexander Schill,
Thomas Springer, Frank Wehner

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

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

Zusammenfassung

Die DFG-Forschergruppe 428 COMQUAD an der TU Dresden hat zum 1. Oktober 2001 die Arbeit aufgenommen. Sie befasst sich mit nichtfunktionalen Eigenschaften von komponentenbasierten Software-Systemen, insbesondere mit zusagbaren quantitativen Eigenschaften, Adaptionsfähigkeit und Sicherheit. Dieser Beitrag beschreibt die Ziele der Forschergruppen und die geplante Vorgehensweise.

1 Einleitung

Bei Methoden, Werkzeugen und Sprachen der Software-Entwicklung war in den vergangenen Jahren ein Wandel von prozeduralen und modularen Ansätzen über objektorientierte Konzepte hin zu komponentenbasierten Lösungen zu beobachten. Das Prinzip der Komponentenbasierung folgt einer einfachen Idee zur Optimierung des Entwicklungsvorgangs, die so alt ist wie die systematische Betrachtung der Software-Entwicklung selbst [24]: An die Stelle des grundlegenden Entwurfs monolithischer Systeme treten Auswahl, Konfigurierung und Einbau von vorhandenen Komponenten, die eine für die jeweilige Anwendung gewünschte Funktionalität erbringen. Das entspricht einem schon lange angestrebten Idealbild der Software-Entwicklung, speziell hinsichtlich des hohen Potenzials an Wiederverwendbarkeit („Software-ICs“) [10].

In jüngerer Zeit wurde klar, dass das ursprüngliche Ziel der Wiederverwendung mit Komponenten nur in relativ wenigen, besonders gut verstandenen und ausgereiften Anwendungsbereichen vollständig erreichbar ist. Das Konzept der Komponentenorientierung ist dennoch ausgesprochen erfolgreich, da erkannt wurde, dass die *Trennung von Infrastruktur und Fachlogik* einen ganz wesentlichen ökonomischen

Effekt in der Software-Entwicklung bewirken kann. Rein fachlogische Komponenten, die eine standardisierte Infrastruktur wiederverwenden, sind schnell und relativ zuverlässig bereitzustellen.

Zwei wichtige Eigenschaften moderner, komplexer Systeme werden jedoch in bestehenden Konzepten komponentenbasierter Systeme nicht berücksichtigt:

- *Quantitative Eigenschaften* wie erzielbare Bandbreiten oder Antwortzeiten lassen sich in bestehenden Komponentenmodellen weder spezifizieren noch garantieren und meist noch nicht einmal abschätzen. Insbesondere fehlt eine Methodik, die es erlaubt, bei der Komposition komplexer Komponenten aus Einzelkomponenten mit bekannten quantitativen Eigenschaften Aussagen über die quantitativen Eigenschaften der komplexen Komponenten abzuleiten. Ähnliches gilt für die Sicherheitseigenschaften von Komponenten.

- *Dynamische Adaption der Komponenten* wird nicht explizit unterstützt; Komponenten sind vielmehr relativ statische Konstrukte, die sich nur aufwändig an veränderte System- und Nutzerumgebungen in einem heterogenen Umfeld anpassen lassen. Gerade bei interaktiven Telediensten ist diese bisher vorhandene Einschränkung problematisch.

Besonders interessant ist das Wechselspiel dieser beiden Eigenschaften, wenn sich etwa aufgrund von Änderungen eines Basissystems die quantitativen Eigenschaften einer Anwendung ändern.

Die DFG-Forschergruppe FOR 428 an der Fakultät Informatik der TU Dresden hat am 1. Oktober 2001 ihre Arbeit aufgenommen, um für diese beiden offenen Probleme Lösungen zu suchen. Der Projektname „COMQUAD“ steht dabei für „COMponents with QUantitative properties and ADaptivity“. Das Vorhaben hat eine Systemarchitektur und eine dazu gehörende Entwicklungsmethodik zum Ziel, die die Komposition adaptiver Software aus Komponenten unter Berücksichtigung zusagbarer nichtfunktionaler Eigenschaften unterstützen. Beispiele für nichtfunktionale, quantitative Eigenschaften sind Durchsatz und Verweilzeit oder auf höheren Abstraktionsebenen etwa auch Bildraten, Bildqualität, Transaktionsanzahl pro Sekunde, Anzahl gleichzeitig bedienbarer Clients eines Servers etc. Es ist Ziel des Vorhabens, eine große Vielfalt an nichtfunktionalen Eigenschaften von Komponenten abzudecken. Insbesondere sollen neben den genannten rein quantitativen Eigenschaften auch Sicherheitseigenschaften unterstützt werden.

Dieser Beitrag beschreibt die Ansätze der Forschergruppe. Dazu wird zunächst ein Überblick über die Komponententechnologie gegeben, um die Voraussetzungen zu klären und eine gemeinsame Grundlage zu schaffen. Darauf aufbauend wird die Zielsetzung noch einmal verdeutlicht, und es werden die verwandten Arbeiten in diesem engeren Feld vorgestellt. Daraus leiten sich offene Fragen ab, zu denen erste Lösungsansätze vorgestellt werden können. Mehr ist zu diesem frühen Zeitpunkt noch nicht möglich. Es erscheint dennoch sinnvoll, auch dies schon an die wissenschaftliche Öffentlichkeit zu bringen, um die Diskussion zu beginnen und den frühzeitigen Austausch von Ideen zu befördern.

2 Komponenten-Technologie im Überblick

Zur Klarstellung wird der Begriff der *(Software-)Komponente* noch weiter zu präzisiert. Das Projekt übernimmt die derzeit verbreitetste Begriffsdefinition [38]: „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.“ Für die folgenden Ausführungen werden einige Aspekte aber noch weiter spezialisiert: Die Schnittstellen einer Komponente umfassen auch Kommunikationsmechanismen, für die die

Definition von Dienstgüteparametern wesentlich ist, z.B. Schnittstellen zur Verarbeitung von regelmäßigen Ereignisströmen. Eine Komponente spezifiziert außerdem ihre Komponentenabhängigkeiten sehr präzise, etwa als Maßzahlen für benötigte Ressourcen. Schließlich enthält eine Komponente Vorkehrungen für die dynamische Reaktion zur Laufzeit auf Änderungen in der Konfiguration und der Ressourcenverfügbarkeit. In den drei zuletzt genannten Punkten unterscheiden sich COMQUAD-Komponenten von den derzeit üblichen Komponenten-Technologien.

Komponenten können eingebaut oder zusammengesteckt werden. Bei der Komposition geht eine Komponente einen *Kontrakt* ein, der besagt, dass die Komponente Zusagen einhält unter der Voraussetzung, dass hierfür im Kontrakt aufgeführte andere Komponenten ebenfalls ihre Zusagen einhalten.

Die derzeit bekannten und verbreiteten Komponententechnologien lassen sich grob unterscheiden in Komponentenkonzepte, die den Schwerpunkt auf die visuelle Programmierung legen, und Konzepte, die vor allem auf die Separation zwischen Fachlogik und Basistechnologien achten. Beispiele für die erste Kategorie sind JavaBeans [36] und ActiveX Controls. In der Praxis werden diese Technologien vorwiegend als Bausteine zur Gestaltung von Benutzeroberflächen eingesetzt. Beispiele für die zweite Kategorie sind Enterprise JavaBeans (EJB) [37] und der noch nicht fertiggestellte Ansatz der „CORBA Components“ [30]. Diese Technologien dienen derzeit vorwiegend zur flexiblen Gestaltung von Server-Anwendungen. Eine Sonderrolle nehmen die von Microsoft definierten Komponentenstandards COM, DCOM und COM+ ein, die historisch aus recht frühen Ansätzen zur Programmkommunikation (OLE) gewachsen sind. Im derzeit vorliegenden Zustand unterstützt die Summe dieser Technologien (COM+) ebenfalls Server-Komponenten für verteilte Anwendungen.

Für die Zwecke dieses Projekts ist die Technologie der Enterprise JavaBeans sicher von größtem Interesse. EJBs sind Server-Komponenten für verteilte Mehrbenutzeranwendungen mit persistenter Datenhaltung. Das Ziel ist hierbei die Trennung der Geschäftslogik von der Infrastruktur. Ein wesentliches Element der EJB-Architektur ist der sogenannte *Container*, der die Infrastruktur für die Server-Komponenten darstellt. Er ist also die Laufzeitumgebung für die Komponenten, bietet den Komponenten Dienste über Standard-APIs an und nutzt Dienste der Hardware- und Software-Umgebung. Der Container arbeitet eng zusammen mit Werkzeugen zur Konfiguration und Installation von Komponenten, die insbesondere Code zur Laufzeitunterstützung von konkreten Komponenten generieren.

Ausgehend von diesen Prinzipien der komponentenbasierten Programmierung, fügt das Projekt COMQUAD spezielle Konzepte für die Unterstützung von quantitativen Eigenschaften, also insbesondere für Medienkomponenten und für Adaptivität, hinzu. Da die hierbei entstehenden Komponenten oft komplex strukturiert sind, wird darüber hinaus angestrebt, dass Komponenten *hierarchisch komponierbar* sind. Grundsätzlich können Komponenten jederzeit zusammengesetzt werden, um größere Systeme aufzubauen. Hierarchische Komposition liegt dann vor, wenn aus Komponenten kein konkretes Anwendungssystem, sondern eine weitere Komponente zusammengesetzt wird. Diese heißt *komposite* Komponente. Der Unterschied zwischen einer allgemeinen Komponente und einer kompositen Komponente ist für den Anwender (Klienten) der Komponente nicht relevant. Dieses hierarchische Kompositionsprinzip wird ansatzweise in COM+ unterstützt; hier wird aber eine wesentlich allgemeinere Realisierung des Prinzips angestrebt.

3 Zielbestimmung

Gesamtziel des Projekts ist es, eine Systemunterstützung zur Erstellung und Komposition komponentenbasierter Software mit zusagbaren nichtfunktionalen Eigen-

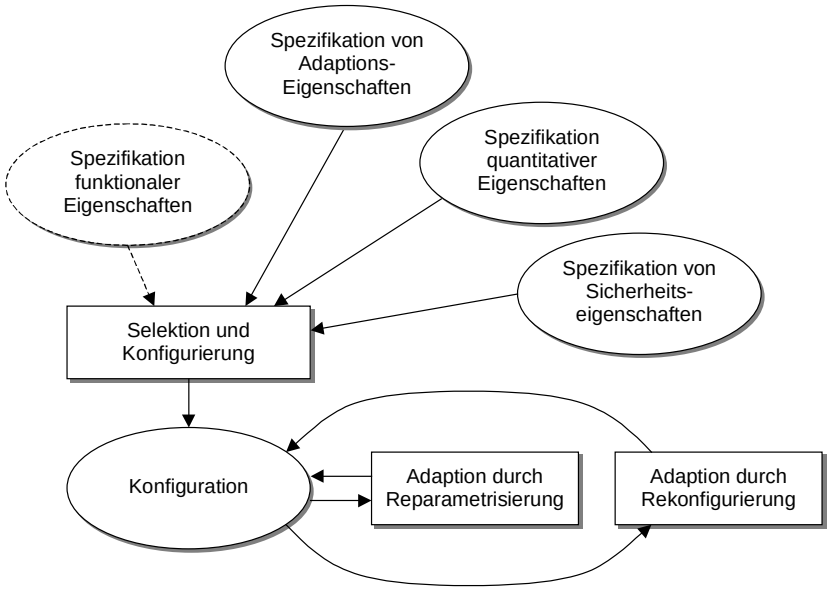


Abbildung 1: Added Value für Komponentensysteme: Adaption und Quantitative Eigenschaften

schaften und Adaption zu schaffen. Dazu sind Methoden und Werkzeuge für die Spezifikation, Konfiguration, Kommunikation und Adaption verteilter und komplexer Softwarekomponenten zu entwickeln, die explizit den Umgang mit nichtfunktionalen Eigenschaften einbeziehen.

Quantitative Zusagen müssen in realen Systemen unter ständigen Änderungen von Parametern durchgesetzt werden. Beispiele für Umgebungsparameter sind etwa die verfügbare Bandbreite bei mobilen Endgeräten oder überhaupt die Verfügbarkeit von Betriebsmitteln, die sich bei Hinzukommen einer neuen Last verändert. Aber auch von der Seite des Benutzers und des Anwendungsprogramms können die Anforderungen an quantitative Eigenschaften zeitlich oder abhängig von Benutzerprofilen wechseln. Die COMQUAD-Entwicklungsmethodik soll diese Dynamik von vornherein mit einbeziehen. Das heißt, dass auch eine Spezifikation des Verhaltens bei sich ändernden Eigenschaften der Umgebung bzw. variierten Anforderungen der Anwendung möglich sein muss.

Abb. 1 soll den *Added Value* des Projekts verdeutlichen: In der klassischen komponentenbasierten Software-Konstruktion (linke Seite) werden ausgehend von einer Spezifikation funktionaler Anforderungen geeignete Komponenten selektiert und konfiguriert. Dies wird ergänzt (rechte Seite) um eine Berücksichtigung quantitativer Eigenschaften bei der Selektion, Parametrisierung und Konfigurierung von Komponenten sowie um Reparametrisierung und Rekonfigurierung für den Fall, dass aufgrund der inhärenten Dynamik eines Systems gegebene Zusagen in Bezug auf quantitative Eigenschaften nicht mehr eingehalten werden können. Der Schwerpunkt des Projekts liegt bei der Unterstützung einer formalen Spezifikation von quantitativen, Sicherheits- und Adaptionseigenschaften. Die Spezifikation funktionaler Eigenschaften wird bewusst als informell angenommen[1].

Von besonderer Bedeutung sind heute verteilte Komponentensysteme mit Echtzeitanforderungen, was impliziert, dass Komponenten verteilbar sind und über Netz-

[1]In einer längerfristigen Fortsetzung des Projekts wäre die Einbeziehung formaler Spezifikations- und Verifikationsansätze für funktionale Eigenschaften denkbar.

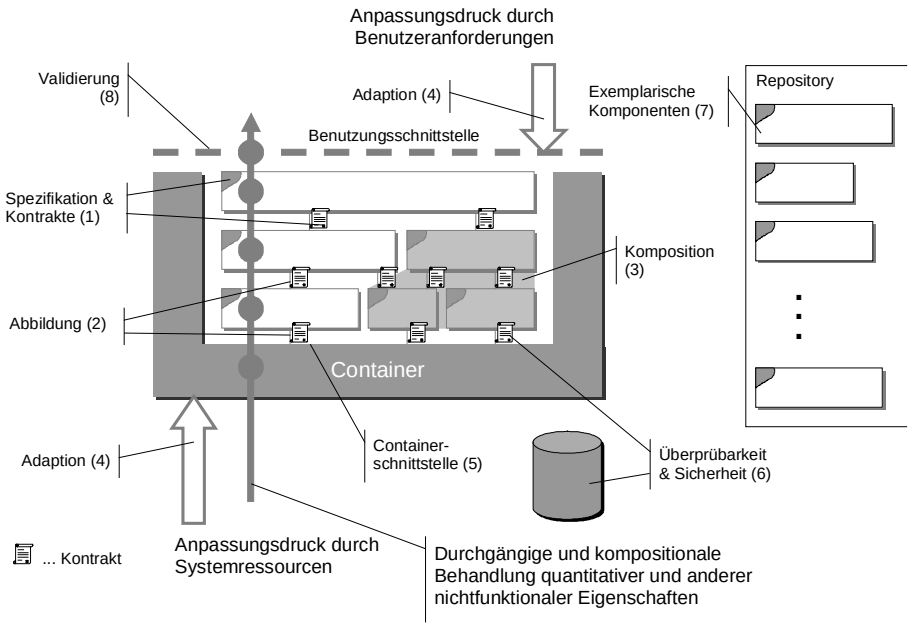


Abbildung 2: Gesamtstruktur des Systems

werke kommunizieren. Dies zu gewährleisten ist Aufgabe moderner *Middleware*, die als Container gegenüber den Komponenten ausgeprägt ist. In dem Projekt wird deshalb eine Struktur mit Container-Schnittstellen gewählt, die die Verteilung von Komponenten über mehrere Rechnersysteme bei Einhaltung der Ziele der beschriebenen Entwicklungsmethodik unterstützt. Das bedeutet, dass spezifizierte quantitative Eigenschaften sich an den Container-Schnittstellen wiederfinden und durchsetzen lassen müssen. Die quantitativen und adaptiven Eigenschaften von Container-Schnittstellen basieren ihrerseits auf quantitativen Eigenschaften von Kommunikationskanälen, Prozessen, Dateien und anderen persistenten Datenobjekten. Damit übernimmt der Container die Rolle einer flexibel konfigurierbaren, bidirektionalen Übersetzung von anwendungsnahen in systemnahe Begriffe und Maße.

Die zugrundeliegende Gesamtstruktur des Systems und die resultierenden Teilaufgaben sind in Abb. 2 skizziert; sie werden im Abschnitt 5 näher aufgeschlüsselt.

4 Verwandte Arbeiten

Softwarekomponenten, Komponentenkontrakte und Komponentenkataloge sind bereits seit sehr langer Zeit Gegenstand der Diskussion [24]. Erst in jüngster Zeit gewinnen aber nichtfunktionale Komponenteneigenschaften an Bedeutung. Die folgenden Abschnitte geben einen groben Überblick über den aktuellen Stand der Forschung.

Spezifikation von nichtfunktionalen Eigenschaften und Kontrakten

Bei der Spezifikation von nichtfunktionalen Eigenschaften liegt ein deutlicher Schwerpunkt bei den quantitativen Eigenschaften, insbesondere unter dem Stichwort „Quality of Service“ (QoS). Mehrere Arbeiten (z.B. die „Quality Objects“ (QuO)) beschäftigen sich im CORBA-Umfeld mit dieser Thematik [22]. QuO ist

ein Framework für verteilte (CORBA-) Anwendungen, die über Weitverkehrsnetze (WANs) miteinander kommunizieren.

Das Ziel der „Trusted Components Initiative“ ist die Bereitstellung von Techniken zur Produktion qualitativ hochwertiger Softwarekomponenten mit Hilfe formaler Spezifikations- und Verifikationsmethoden. Schwerpunkt ist hier der Entwurf unter Einhaltung präziser „Verträge“ [25]. Aus dieser sehr jungen Initiative sind bisher noch keine technischen Resultate hervorgegangen.

Zwei Teilprojekte (B5 und B10) des SFB 501 „Entwicklung großer Systeme mit generischen Methoden“ (GeneSys) beschäftigen sich mit Softwarearchitekturen und sog. DesignSpaces, die nichtfunktionale Eigenschaften, wie z.B. Zeit- und Speicherkomplexität, berücksichtigen. Das Konzept generischer, mehrdimensionaler DesignSpaces erlaubt die Spezifikation von Komponenten auch hinsichtlich nichtfunktionaler Eigenschaften [3]. Konkrete Spezifikationssprachen bzw. -werkzeuge werden allerdings nicht definiert.

Ein ebenfalls sehr interessanter Ansatz ist die Nutzung der Unified Modelling Language (UML) zur Bereitstellung eines generischen Frameworks zur Modellierung von Ressourcen [35]. Die Idee ist, Software-Entwickler bereits in der Analyse- bzw. Entwurfsphase bei der Berücksichtigung von quantitativen und Echtzeit-Eigenschaften der zu implementierenden Software zu unterstützen, indem eine standardisierte Möglichkeit zur Modellierung abstrakter Ressourcen, Dienste, QoS-Kontrakte etc. zur Verfügung gestellt wird.

Keiner der hier genannten Ansätze unterstützt auf konsequente Art und Weise die konkrete Spezifikation nichtfunktionaler Eigenschaften für Komponenten. Viele Arbeiten konzentrieren sich nur auf Teilaspekte wie Quality of Service oder Schutzziele, berücksichtigen nichtfunktionale Eigenschaften aber nicht in ihrer Gesamtheit. Eine weitere Einschränkung ist, dass diese zusätzlichen Beschreibungsmöglichkeiten oft nur für verteilte Anwendungen, basierend auf CORBA, vorgesehen sind. Andere Arbeiten weisen zwar mögliche Wege auf, generische Mittel und Methoden zur Spezifikation von nichtfunktionalen Eigenschaften zu benutzen, definieren aber (noch) keine konkreten Beschreibungsmittel für die genannten Anforderungen.

Techniken zur Abbildung quantitativer Anforderungen

Im Bereich Quality of Service [2] sind vorwiegend Mechanismen und Konzepte entwickelt worden, die nur ausgewählte Aspekte, wie z.B. die Reservierung von Netzwerk- und Betriebsystemressourcen bzw. die Abbildung von QoS-Anforderungen auf die zugrundeliegende Infrastruktur behandeln, jedoch keine durchgehend vertikale Abbildung und Durchsetzung quantitativer Anforderungen unterstützen. Im Rahmen der Standardisierungsbestrebungen im Internet-Bereich sind mit RSVP (Resource Reservation Protocol, [43]) und Differentiated Services [5] die beiden wichtigsten Vertreter genannt, die Möglichkeiten zur Reservierung von Ressourcen auf Netzwerkebene bieten.

Erste Ansätze zur Integration flexibler Protokollarchitekturen in objektorientierte Middleware sind in [33] beschrieben. Lösungsansätze für die systemseitige Adaption zur Gewährleistung von Quality-of-Service-Eigenschaften finden sich in [20]. Die automatische Berücksichtigung quantitativer Anforderungen in der Entwicklungsphase ist Gegenstand des Performance Requirements Frameworks [28]. Ein Anforderungsgraph orientiert sich dabei an den verschiedenen Implementierungsebenen. Eine Einbeziehung von Performance-Vorhersagen in den Software-Entwicklungsprozess ist Ziel einer anderen Arbeit [42].

Es fehlt aber nach wie vor eine konsequente Integration der Abbildungsmechanismen mit dem Ansatz der Komponentenbasierung.

Mechanismen zur Komposition von Komponenten

Das Thema der Kompositionsmechanismen für Komponenten ist generell noch wenig systematisch untersucht, und das Zusammenspiel mit quantitativen Aspekten ist überhaupt noch Neuland. Seit etwa zehn Jahren wird aber intensiv an Software-Architekturen zur Unterstützung von Quality of Service (QoS) gearbeitet. Einen guten Überblick über den Stand der Arbeiten bis ca. 1996 bietet [2]. Während zunächst Fragen der Netzarchitektur und Protokollarchitektur im Mittelpunkt standen, trat in den letzten Jahren immer mehr die Anbindung an Anwendungen in den Vordergrund, insbesondere die Erweiterung generischer Objekt-Kommunikations-Middleware (vor allem CORBA).

An erster Stelle stehen hier die Bestrebungen, Real-Time- und Quality-of-Service-Aspekte im kommenden CORBA-Standard zu berücksichtigen. So standardisiert Real-Time CORBA (RT-CORBA) die Ressourcensteuerung für echtzeitfähige Ende-zu-Ende-Dienste [31]. Darüber hinaus enthält die CORBA Messaging Specification u.a. ein QoS-Framework, das auf Policies beruht [29]. Es definiert aber keine konkreten QoS-Beschreibungsmittel.

Die Distributed Object Computing Group an der Washington University, St. Louis (Missouri, USA), hat einen effizienten echtzeitfähigen Object Request Broker realisiert (Projekt TAO [39]), der RT-CORBA implementiert. QoS für Anwendungen wird durch Integration der Middleware mit der unterliegenden Netz- und Betriebssystem-Infrastruktur erreicht. Die Distributed Multimedia Research Group an der University of Lancaster untersucht Ansätze zur Entwicklung von Middleware-Plattformen, die ihre Unterstützung für Komponenten adaptiv anpassen und die unterliegende Infrastruktur beobachten und beeinflussen können. Eine inhaltliche Nähe mit dem hier beschriebenen Vorhaben zeigen insbesondere Arbeiten zu einer verteilten Multimedia-Komponenten-Architektur [40]. Hier wird die Komposition von Komponenten (im Sinne von Microsofts COM) über sog. „Binding“-Objekte beschrieben, die die Möglichkeit von dynamischer QoS-Adaptierung und Zugriff auf ein Ressourcen-Management ermöglichen. Eine systematische Konstruktion ganzer Systeme aus Komponenten mit Einhaltung definierter QoS-Anforderungen wird aber nicht untersucht. Die Distributed Multimedia Systems Group am UniK-Institut der Universität Oslo befasst sich mit der flexiblen Einbindung von Multimediaobjekten in eine Netzinfrastruktur. Adaptivität spielt hier eine Rolle [34], allerdings auf der Ebene der dynamischen Konfigurierung und Optimierung von Kommunikationsprotokollen und nicht im Sinne der Adaption an Benutzeranforderungen.

Allen diesen Projekten ist gemeinsam, dass sie das Konzept der Komposition von Anwendungen aus „Black-Box“-Komponenten (wie z.B. JavaBeans) nicht direkt unterstützen. Sie zeigen jedoch, dass eine ausreichende technologische Ausgangsbasis für COMQUAD besteht.

Adaptionsmechanismen

Bereits seit Jahren gibt es verschiedenste Forschungsansätze zur systemseitigen statischen Adaption verteilter Anwendungen im Rahmen des sogenannten Konfigurationsmanagements. Klassische Vertreter sind hier die Projekte CONIC und REX [23, 18], die aber keinen komponentenbasierten Ansatz im modernen Sinne verwenden.

Eine neuere Entwicklung mit einem komponentenbasierten Ansatz zur statischen Adaption stellt Koala [32]dar. Hier lassen sich innerhalb einer Strukturbeschreibung alternative Komponenten spezifizieren. Mittel zur dynamischen Adaption fehlen jedoch, und quantitative Eigenschaften werden nicht berücksichtigt.

Im Bereich der sog. adaptiven Middleware befassen sich Arbeiten wie [17] mit der Rekonfiguration, d.h. Adaption der ORB- bzw. der Server-Struktur zur Lauf-

zeit. [4] erlaubt die Introspektion und Rekonfiguration auf einem feingranularen Niveau (Objektebene, Methodenebene). Das AspectIX-Projekt [14] schließlich besitzt ein abstraktes, von der unterliegenden Middleware (CORBA, mobile Agenten etc.) unabhängiges Objektmodell. Alle diese Arbeiten behandeln entweder die relativ eingeschränkte, grobgranulare Rekonfiguration von Teilen des ORB[2] oder die feingranulare Rekonfiguration auf Objekt- oder Methodenebene. Die Adaption wird jedoch ausschließlich auf der Seite des Laufzeitsystems betrachtet, eine Einbeziehung der Anwendung erfolgt nicht.

Adaptionsvorgänge können ihre Ursache aber auch in sich verändernden Benutzerpräferenzen haben. Bei der Benutzermodellierung adaptiver Hypermedia-Anwendungen und -Systeme spielen Aspekte der Wissensakquisition [7], Benutzermodellierung [15] und Adaptionstechniken [13] eine Rolle. Das Konzept der Adaptation Spaces wird für multimediale Anwendungen genutzt, um Komponenten mit dynamischen, adaptiven Eigenschaften auszustatten [6]. Im Projekt TELLIM werden Ansätze zur temporären Benutzermodellierung untersucht und Adaptionen multimedialer Web-Präsentationen abhängig von inhaltlichen und gestalterischen Präferenzen der Nutzer sowie den technischen Eigenschaften des Präsentations- und Übertragungssystems dynamisch erzeugt [16].

Containerschnittstelle und -realisierung

Eine übergreifende Dienstgüteverwaltung ist Ziel einer Arbeit über QoS Translation (oder QoS Mapping) und QoS Negotiation [27]. Die bidirektionale Abbildung der QoS-Eigenschaften erfolgt aber nur zwischen den Schichten Anwendung, Betriebssystem/Netz und Geräte. Darüber hinaus ist eine Ressourcenverwaltung für die Aufgaben Reservierung, Verarbeitungssteuerung und Anpassung an Ressourcenänderungen vorgesehen.

Zu den wichtigsten Forschungsergebnissen auf dem Gebiet der Echtzeitsysteme gehört, dass die Einschränkung auf geschlossene Prozess-Systeme mit festen Zeitschranken teils durch neue, teils durch modifizierte traditionelle Scheduling-Techniken aufgehoben wurde. Einen ausgezeichneten Überblick zum aktuellen Stand gibt [21]; als inhaltlich nächstgelegene Ansätze und Ausgangspunkte für die Forschergruppe sind Imprecise Computations [9] und statistisches ratenmonotones Scheduling [1] zu nennen. Keiner dieser Ansätze hat jedoch bisher Eingang in die komponentenbasierte Software-Konstruktion gefunden.

Überprüfbarkeit und Sicherheit

Ansätze zum Schutz von Software durch Software z.B. durch vorherige Interpretation und Prüfung werden in [41] beschrieben, beziehen sich aber vor allem auf Java. Im Gegensatz dazu steht die Authentifikation von Komponenten (z.B. Java-Applets) zum Schutz des ausführenden Systems vor der Komponente; Methoden hierzu wurden in [11] vorgestellt.

Interessant im Hinblick auf den Schutz der Software-Komponente vor dem System (Container) ist auch der Ansatz verteilter Berechnungsprotokolle [8]. Mehrere Container müssten zusammenarbeiten, um eine Unterschrift zu generieren.

Die Zusicherung von (Komponenten-)Eigenschaften ist in [19] beschrieben. Zur Überprüfbarkeit von Fehlerverursachern werden neben „normalen“ Zusicherungen auch Zusicherungen für den Fehlerfall gegeben.

[2]Zur Laufzeit können Instanzen des modular aufgebauten Laufzeitsystems durch Instanzen alternativer Implementierungen ersetzt werden.

5 Offene Fragen und erste Lösungsansätze

Die Systemunterstützung soll im Rahmen einer verteilten Plattform realisiert werden, wobei auf jedem beteiligten Rechner entsprechende Container bereitzustellen sind. Durch Einsatz von Techniken aus dem Bereich der Echtzeitbetriebssysteme sind diese in die Lage zu versetzen, quantitative Zusagen einzuhalten. Teil der Container-Funktionalität muss ferner eine Infrastruktur zur Verwaltung und Komposition von Komponenten sein. Darauf aufbauend werden die genannten Abbildungs- und Adaptionsmechanismen realisiert. Zunächst orthogonal hierzu wird der Entwurfsprozess adressiert, der insbesondere die Spezifikation quantitativer Eigenschaften sowie die Beschreibung von Adaptionsmöglichkeiten bereits in einer frühen Phase berücksichtigt. Über geeignete Werkzeuge wird dann die Kopplung mit der Initialisierungs- und Laufzeitphase realisiert.

Die Komponentenverwaltung wird durch eine Bibliothek exemplarischer Komponenten validiert, wobei diese auf das Anwendungsszenario zugeschnitten werden. Dabei werden sowohl Komponenten grober Granularität (z.B. Teleteaching-Server) als auch Komponenten feiner Granularität (z.B. Video-Codecs oder Pufferkomponenten für schwankungsbeschränkte Ströme) explizit im Hinblick auf quantitative Eigenschaften betrachtet.

Im folgenden werden die einzelnen Teilaufgaben, die sich aus diesen Anforderungen ergeben, genauer betrachtet.

5.1 Spezifikation von Kontrakten

Um Kontrakte zwischen Komponenten spezifizieren zu können, müssen die notwendigen begrifflichen Grundlagen geschaffen und in einen konkreten sprachlichen Rahmen abgebildet werden. Dabei empfiehlt es sich, die Spezifikation von quantitativen Eigenschaften, von Sicherheitseigenschaften und von Adaptionseigenschaften zu unterscheiden. Sie fließt direkt in die Formulierung von Kontrakten in einer *Component Description Language* (CDL) ein. Darüber hinaus werden Mechanismen zur expliziten Spezifikation der Umgebungsbedingungen einer Komponente (Plattform, Einbindung in eine Verteilungsstruktur) benötigt.

Als konzeptionelle Basis werden Strukturen für Kontrakte entwickelt, die die Bezüge zwischen den geforderten Eigenschaften auf unterschiedlichen Abstraktionsebenen beschreiben. Wichtige Einzelfragen betreffen die Art der angebotenen Abstraktions- und Beschreibungsebenen, etwa subjektive Qualitätseigenschaften, Bildqualität in groben Abstufungen, technische Qualitätsparameter sowie konkret auf erforderliche Ressourcen bezogene Anforderungen. Ferner wird geklärt, wie die Kontrakte selbst strukturiert sind (z.B. bzgl. Techniken zur Abbildung quantitativer Anforderungen, Umgebungsbedingungen, Beschreibung von Sicherheitseigenschaften oder zeitlicher Gültigkeit). Kontrakte werden als Bestandteile von Software-Komponenten verstanden, so dass eine Komponente zu einem wirklich eigenständigen und in sich abgeschlossenen Software-Baustein wird. Die Prinzipien der Vererbung und komponentenspezifischen Modifikation werden auch auf Kontrakte angewandt.

Bereits zur Entwurfszeit wird eine Komponente durch die Wahl spezieller Parameterwerte an die Anforderungen der konkreten Anwendung (statisch) adaptiert. Darüber hinaus wird die Möglichkeit einer dynamischen Adaption von Komponenten an sich verändernde Bedingungen zur Laufzeit unterstützt (z.B. Ressourcenänderungen, geänderte Benutzerpräferenzen). Für diesen Zweck wird die CDL um Sprachmittel ergänzt, die die Regeln für solche Adaptionen festlegen. Die Adaption kann sich dabei auf alle Eigenschaften einer Komponente beziehen, also quantitative ebenso wie Sicherheitseigenschaften (u.U. sogar rekursiv wieder auf Adaptionseigenschaften). Solange nur Parameterwerte einer Komponente verändert

werden, sind dynamische Adaptionseigenschaften den statischen noch sehr ähnlich (nur mit einer zusätzlichen Abhängigkeit von bestimmten Umgebungsbedingungen). Ein komplizierterer Fall tritt ein, wenn die konkrete Auswirkung einer Adaption zu strukturellen Änderungen in der Komponentenkonfiguration führt, also z.B. zum Austausch von Komponenten durch andere Versionen oder zur Einbindung neuer, in der bisherigen Konfiguration nicht vorhandener Komponenten. Alle diese Adaptionsformen werden im Zusammenhang mit den konkreten Anwendungsbeispielen evaluiert und ggf. in die CDL integriert.

5.2 Techniken zur Abbildung quantitativer Anforderungen

Für die spezifizierbaren quantitativen Anforderungen wird eine systematische Abbildung auf die Eigenschaften von beliebigen anderen Komponenten und Containern entwickelt. Die Abbildung der Anforderungen kann in sehr einfachen Fällen tabellengesteuert erfolgen (z.B. bei der Abbildung abstrakter Video-Qualitätsstufen auf Steuerparameter eines Codecs). Im Regelfall sind aber viele Einflussgrößen zu berücksichtigen und auch verschiedene Teilkomponenten einzubeziehen. Deshalb werden komplexe Abbildungsfunktionen bzw. -heuristiken entwickelt, wie sie ansatzweise aus dem Bereich des QoS Mapping bekannt sind.

Tatsächlich erscheint die Vorstellung als „Abbildung“ bereits als zu eng. Zumindest bei quantitativen Anforderungen gibt es in der Regel viele Möglichkeiten, sie zu erfüllen. Als Beispiel kann der Ausgleich zwischen Prozessornutzung und Speicherbelegung dienen: Die Bereitstellung von mehr Speicher erlaubt es, auf wiederholte Berechnungen zu verzichten. Insofern handelt es sich eher um eine Suche nach eine Parameterkonfiguration, bei der es auch eine Zielfunktion geben muss. Es ist eine entfernte Ähnlichkeit mit der Anfrageoptimierung in Datenbanksystemen zu erkennen [26]. Allerdings ist das Verhalten der relationalen Operatoren sehr viel stärker eingeschränkt und damit auch besser abschätzbar als das einer Komponente mit beliebiger Funktionalität.

Zusätzlich sind Sicherheitseigenschaften bei der Abbildung zu berücksichtigen. Hierbei ist zu unterscheiden zwischen Schutzzielen auf hohem Abstraktionsniveau (z.B. Integrität, Vertraulichkeit, Zurechenbarkeit, Anonymität) und Sicherheitsmechanismen auf niedrigerer Ebene (an der Schnittstelle des Containers etwa Kryptoverfahren, digitale Signaturen oder Mixe).

5.3 Mechanismen zur Komposition von Komponenten

Aufbauend auf den Spezifikations- und Abbildungstechniken wird untersucht, wie Komponenten systematisch komponiert und parametrisiert werden können. Dabei wird vorausgesetzt, dass die quantitativen Eigenschaften der Komponenten bereits bekannt sind (siehe Abschnitt 5.1). Beim Entwurf einer Anwendung auf der Basis vorgefertigter Komponenten ist die übliche Vorgehensweise, eine Anzahl von Komponenten statisch zu instanziieren und für weitere, dynamisch erzeugte Komponenten ihre Instanziierung zur Laufzeit vorzubereiten. Für diese Komponenten werden konkrete Parameterwerte ausgewählt, um ein geeignetes Verhalten der Komponenten zur Laufzeit zu erreichen. Über Ereignismechanismen und andere Kommunikationsparadigmen werden die Komponenten untereinander und mit dem Container verbunden. Um die hierarchische Komposition von Komponenten zu ermöglichen, ist es wesentlich, dass für eine Gruppe von Komponenten, die in der oben beschriebenen Weise zusammengefügt wurde, eine Beschreibung der nichtfunktionalen Eigenschaften aus den Beschreibungen der Einzelkomponenten ableitbar ist.

Der Gesamtprozess der Konfigurierung von Komponenten zu einem Anwendungssystem wird durch Werkzeuge und eine Infrastruktur unterstützt, die statische

und dynamische Adaption von Komponenten unter Berücksichtigung der quantitativen Eigenschaften erlaubt. Da in vielen Fällen bei der Abbildung nur mit Heuristiken gearbeitet werden kann (siehe Abschnitt 5.2), sollen diese Werkzeuge auch explizite Leistungstests zur Überprüfung quantitativer Eigenschaften vorsehen.

Die Komposition kann durch Werkzeuge effektiv unterstützt werden, weil die Spezifikationen der Komponenten in maschinenlesbarer Form als CDL vorliegt. Ein solches Entwurfswerkzeug ermöglicht die Auffindung und Auswahl von Komponenten aus einer Bibliothek vorhandener Komponenten, unterstützt die Instanziierung, Parameter-Spezialisierung und Zusammensetzung von Komponenten in Form einer graphischen Benutzeroberfläche, bietet insbesondere Algorithmen zur Analyse der spezifizierten nichtfunktionalen Eigenschaften an und bereitet die Ausführung der Komponenten vor. Konkrete „Vertragsabschlüsse“ zwischen Komponenten bzw. zwischen einer Komponente und dem Container werden auf ihre Realisierbarkeit hin untersucht. Darüber hinaus werden abgeleitete CDL-Spezifikationen für zusammengesetzte Kontrakte errechnet. Im Falle eines in sich schlüssigen Systems von Komponenten werden Vorbereitungen dafür getroffen, dass die Komponenten zur Laufzeit tatsächlich unter den spezifizierten Bedingungen ausgeführt werden (d.h. Codegenerierung für die Laufzeitumgebung).

5.4 Adaptionsmechanismen

Systemseitige Adaptionsmechanismen dienen der Anpassung an sich verändernde Systemeigenschaften. Die Auswirkungen der Adaptionsvorgänge sollten begrenzt bleiben, so dass Vertragsverletzungen auf höheren Schichten vermieden werden. Interessant ist dabei insbesondere, wie die Entwicklung systemseitiger Adaptionsmechanismen durch Autorenumgebungen unterstützt, wie anwendungsseitige Adaptionsmechanismen auf systemseitige abgebildet und wie Konflikte während dieses Adaptionsprozesses aufgelöst werden können.

Anwendungsbezogene Adaptionsmechanismen können entweder direkt vom Benutzer oder indirekt durch eine Anwendung angestoßen werden. Praktische Anwendungsfälle sind z.B. die Anpassung der Darstellung an sich ändernde Qualitätsanforderungen oder Interessen des Benutzers. Während der erste Fall noch recht gut auf quantitative Eigenschaften und systemseitige Adaptionsmechanismen abgebildet werden kann, ist der zweite relativ unscharf, weil ein gewisser Spielraum bei der Abbildung auf quantitative Eigenschaften darunterliegender Komponenten bleibt.

Ausgehend von der Spezifikation von Adaptionseigenschaften werden Mechanismen entwickelt, die zulässige Adaptionen durchführen, dabei aber eng verzahnt arbeiten mit den Techniken zur Abbildung quantitativer Anforderungen (siehe Abschnitt 5.2) und mit den Kompositionsmechanismen (speziell bei Konfigurationsänderungen; siehe Abschnitt 5.3).

5.5 Container-Schnittstelle und -Realisierung

Es wird eine mit den Spezifikations- und Abbildungsverfahren verträgliche Container-Schnittstelle entworfen. Sie hat die Aufgabe, Kontrakte auf der (aus Sicht der Komponenten) untersten Ebene durch systemseitige Mechanismen und Ressourcen abzusichern. Hierfür sind Reservierungstechniken für Betriebsmittel wie Prozessor, Speicher und Kommunikationskanäle einzusetzen.

Container können in ihrem Verhalten Änderungen der Umgebung widerspiegeln. Beispiele sind Änderungen in der verfügbaren Bandbreite bei mobilen Endgeräten oder Schwankungen in den verfügbaren Betriebsmitteln, die durch Hinzukommen einer neuen Last ausgelöst werden. Die Entwicklungsmethodik wird diese Dynamik von vornherein mit einbeziehen. Ein möglicher Ansatz ist die Unterscheidung zwischen Pflicht- und Wahl-Teilen einer Leistung (Imprecise Computations [9]) mit der

Angabe, welche Ereignisse zum Wegfall der optionalen Leistung führen.

Für die Kooperation von Komponenten in unterschiedlichen Containern werden zusagenfähige Kommunikationskonzepte zwischen Containern bereitgestellt, die auch individuelle logische Kanäle zwischen dedizierten Komponenten ermöglichen.

5.6 Überprüfbarkeit und Sicherheit

Das Vertrauen des Anwenders in die Software wird durch das Komponentenkonzept beeinträchtigt, da verstärkt Software unterschiedlicher Hersteller eingesetzt wird.

Mit Hilfe der Kontrakte soll es möglich sein, im Nachhinein die Nichterfüllung zugesagter Eigenschaften zu beweisen. Eine notwendige Voraussetzung für die Zuweisung von Verantwortung im Fehlerfall ist die Identifizierbarkeit aller an der Ausführung Beteiligten. Neben den Komponenten muss auch deren Ausführungsumgebung identifizierbar sein.

Die Einhaltung der Kontrakte muss zur Laufzeit überwacht, d.h. Kontraktverletzungen müssen erkannt und dokumentiert werden. Durch entsprechende Reaktionen (insbesondere systemseitige Adaption, siehe Abschnitt 5.4) soll die Erfüllung der Zusagen der Anwendung wieder gesichert werden. Für die genannten Aufgaben sind geeignete Mechanismen zu entwickeln.

Bei der Komponentenkonfigurierung werden die Kompatibilität von Sicherheitsmechanismen mit der Zusammensetzung von Komponenten, die Zertifizierung dieser Mechanismen bei vorhandenen Komponenten und die Klassifikation von Komponenten in bezug auf Sicherheitsmerkmale adressiert.

Auch die Definition von Restriktionen bezüglich zulässiger Adaptionsvorgänge ist denkbar, falls dadurch Sicherheitseigenschaften verletzt werden könnten. Dies betrifft zum einen den Einsatz von Komponenten mit anderen Sicherheitseigenschaften, insbesondere aber auch die Veränderung des Vertrauensbereichs des Anwenders durch Adaption.

6 Zusammenfassung

Dieser Beitrag hat ein Projekt vorgestellt, das sich der Einbeziehung von nichtfunktionalen Eigenschaften in die Entwicklung von Software-Systemen aus Komponenten widmet. Dabei sollen nichtfunktionale Anforderungen an das zu entwickelnde System die Auswahl und Konfigurierung der Komponenten beeinflussen, und andererseits sollen sich aus bekannten nichtfunktionalen Eigenschaften der Komponenten die Eigenschaften des Systems ableiten lassen. Die im Projekt betrachteten Eigenschaften betreffen insbesondere die Leistung, die Adaptivität und die Sicherheit. Das Zusammenwirken von Forschungsgruppen mit Kompetenz jeweils in den Bereichen Software Engineering, Datenbanken, Betriebssysteme, Rechnernetze, Multimedia-Technik und Datensicherheit ermöglicht einen neuen Lösungsansatz. So können die Spezifikationstechniken von vornherein abgestimmt werden mit dem, was systemintern durch Reservierung und Ablaufplanung (in der Tradition der Echtzeitsysteme wie auch der QoS-Architekturen) überhaupt erreichbar ist. Zwischen diesen beiden Rändern des Entwurfsraums bewegen sich die Techniken zur Abbildung von Anforderungen, zur Komposition, zur Adaption und zur Sicherheitsüberprüfung. Ein konkretes Anwendungsszenario (Teleteaching) wird herangezogen, um eine Bibliothek exemplarischer Komponenten aufzubauen und die Entwicklungsmethodik einer prototypischen Validierung zu unterziehen.

Literatur

[1] A. Atlas and A. Bestavros. Statistical rate monotonic scheduling: Algorithms, analysis and evaluation. Technical Report TR 98-010, Boston Univ., 1998.

[2] C. Aurrecochea, A.T. Campbell, and L. Hauw. A survey of QoS architectures. *Multimedia Systems Journal*, pages 138–151, May 1998. Special Issue on QoS Architectures.

[3] L. Baum, M. Becker, L. Geyer, G. Molter, and P. Sturm. Driving the composition of runtime platforms by architectural knowledge. In *Proc. 8th ACM SIGOPS European Workshop: Support for Composing Distributed Applications (Sintra, Portugal, Sept. 1998)*, 1998.

[4] G.S. Blair, G. Coulson, P. Robin, and M. Papathomas. An architecture for next generation middleware. In *Proc. IFIP Int. Conf. on Distributed Systems Platforms and Open Distributed Processing (Middleware'98, Lake District, UK)*. Springer, 1998.

[5] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. RFC 2475: An architecture for differentiated services, December 1998. URL http: //www.ietf.org /rfc /rfc2475.txt.

[6] S. Bowers et al. Applying adaptation spaces to support quality of service and survivability. In *Proc. DARPA Information Survivability Conf. & Exposition*, volume II of II, January 2000.

[7] P. Brusilovsky. Methods and techniques of adaptive hypermedia. *User Modeling and User Adapted Interaction*, 6(2–3):87–129, 1996.

[8] D. Chaum. The Spymasters double-agent problem: Multiparty computations secured unconditionally from minorities and cryptographically from majorities. In *Proc. Crypto '89*, number 435 in LNCS, pages 591–602, Berlin, 1990. Springer.

[9] J.-Y. Chung, J.W.S. Liu, and K.-J. Lin. Scheduling periodic jobs that allow imprecise results. *IEEE Trans. on Computers*, 39(9), 1990.

[10] Brad J. Cox. Planning the software industrial revolution. *IEEE Software*, 7(6), November 1990.

[11] G.I. Davida, Y. Desmedt, and B.J. Matt. Defending systems against viruses through cryptographic authentication. In *Proc. IEEE Symposium on Security and Privacy*, pages 312–318, Washington, 1989. IEEE Computer Society Press.

[12] *Proc. IFIP/ACM Int. Conf. on Distributed Systems Platforms (New York, April 2000)*, number 1795 in LNCS, Berlin, 2000. Springer.

[13] J. Fink, A. Kobsa, and A. Nill. Benutzerorientierte adaptivität und adaptierbarkeit im projekt avanti. In *Proc. Software Ergonomie '97, Dresden*, pages 135–143, 1997.

[14] F. Hauck, U. Becker, M. Geier, E. Meier, U. Rastofer, and M. Steckermeier. AspectIX: an aspect-oriented and CORBA-compliant ORB architecture. Technical Report TR-I4-08-08, IMMD IV, Univ. Erlangen-Nürnberg, 1998.

[15] H. Hohl, H.-D. Böcker, and R. Gunzenhäuser. An adaptive hypertext system for exploratory learning and programming. *User Modeling and User-Adapted Interaction*, 6(2–3):131–156, 1996.

[16] Tanja Hölldobler. *Temporäre Benutzermodellierung für multimediale Produktpräsentationen im World Wide Web*. Europäische Hochschulschriften. Peter Lang Verlag, 2001. TU Dresden, Fakultät Informatik, Dissertation.

[17] F. Kon, M. Roman, P. Liu, J. Mao, T. Yamane, L. Magalhaes, and R. Campbell. Monitoring, security and dynamic configuration with the DynamicTAO reflective ORB. In DSP00 [12], pages 121–143.

[18] J. Kramer, J. Magee, M.S. Sloman, and N. Dulay. Configuring object based distributed programs in rex. *IEEE Software Engineering Journal*, 7(2):139–149, 1992. Special Issue on Object Oriented Systems.

[19] P.A. Lee and T. Anderson. *Fault Tolerance – Principles and Practice*. Number 3 in Dependable Computing and Fault-Tolerant Systems. Springer, Wien, 2nd rev. ed. edition, 1990.

[20] B. Li and K. Nahrstedt. Configuring adaptive applications. In DSP00 [12], pages 256–272.

[21] J.W.S. Liu. *Real-Time Systems*. Prentice Hall, 2000.

[22] J. Loyall, D. Bakken, R. Schantz, J. Zinky, D. Karr, R. Vanegas, and K. Anderson. QoS aspect languages and their runtime integration. In *Proc. 4th Workshop on Languages, Compilers, and Run-time Systems for Scalable Computers (LCR '98, Pittsburgh, USA, May 1998)*, number 1511 in LNCS, Berlin, 1998. Springer.

[23] J. Magee, J. Kramer, and M. Sloman. Constructing distributed systems in Conic. *IEEE Transactions on Software Engineering*, 15(6):663–675, 1989.

[24] M.D. McIlroy. Mass produced software components. In *Proc. Nato Software Eng. Conf. (Garmisch, 1968)*, pages 138–155, 1968.

[25] B. Meyer, C. Mingins, and H. Schmidt. Providing trusted components to the industry. *IEEE Computer*, pages 104–105, May 1998.

[26] Bernhard Mitschang. *Anfrageverarbeitung in Datenbanksystemen : Entwurfs- und Implementierungskonzepte*. vieweg, Braunschweig / Wiesbaden, 1995.

[27] K. Nahrstedt and R. Steinmetz. Resource management in networked multimedia systems. *IEEE Computer*, pages 52–63, May 1995.

[28] B.A. Nixon. Managing performance requirements for information systems. In *Proc. 1st Int. Workshop on Software and Performance (WOSP '98, Santa Fe, NM, USA, Oct. 12-16, 1998)*, pages 131–144. ACM Press, 1998.

[29] Object Management Group. CORBA messaging joint revised submission. OMG Document, May 1998. URL http:// www.omg.org/ cgi-bin/ doc?orbos/ 98-05-05.

[30] 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.

[31] Object Management Group. Real-time CORBA joint revised submission. OMG Document, March 1999. URL http:// cgi.omg.org/ cgi-bin/ doc?orbos/ 99-02-12 or http:// cgi.omg.org/ cgi-bin/ doc?orbos/ 99-03-29.

[32] R. Ommering, F. van der Linden, J. Kramer, and J. Magee. The Koala component model for consumer electronics software. *IEEE Computer*, 33(3):78–85, March 2000.

[33] C. O'Ryan, F. Kuhns, D. Schmidt, O. Othman, and J. Parsons. The design and performance of a pluggable protocols framework for real-time distributed object computing middleware. In DSP00 [12], pages 372–395.

[34] T. Plagemann, F. Eliassen, V. Goebel, T. Kristensen, and H.O. Rafaelsen. Adaptive QoS-aware binding of persistent multimedia objects. In Z. Tari, R. Meersman, R. Soley, and O. Burkhes, editors, *Int. Symposium on Distributed Objects and Applications (Edinburgh, Sept. 1999)*. IEEE Press, 1999.

[35] B. Selic. A generic framework for modeling resources with UML. *IEEE Computer*, pages 64–69, June 2000.

[36] Sun Microsystems. JavaBeans API Spezification, Version 1.01, July 1997.

[37] Sun Microsystems. Enterprise JavaBeans Specification, Version 2.0. Proposed Final Draft 2, April 2001.

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

[39] TAO. Real-time CORBA with TAO. URL http:// www.cs.wustl.edu/ ~schmidt/ TAO.html.

[40] D.G. Waddington and G. Coulson. A multimedia component architecture. In *Proc. 1st IEEE Int. Workshop on Enterprise Distributed Object Computing (EDOC '97, Surfer's Paradise, Australia, Oct. 1997)*, 1997. URL ftp:// ftp.comp.lancs.ac.uk/ pub/ mpg/ MPG-97-11.ps.Z.

[41] D.S. Wallach, D. Balfanz, D. Dean, and E.W. Felten. Extensible security architectures for java. In *Proc. 16th ACM Symp. on Operating Systems Principles (Oct. 1997)*, volume 31 of *Operating System Review*, pages 116–128, December 1997. no. 5.

[42] M. Woodside, C. Hrischuk, B. Selic, and S. Bayarov. A wideband approach to integrating performance prediction into a software design environment. In *Proc. 1st Int. Workshop on Software and Performance (WOSP '98, Santa Fe, NM, USA, Oct. 1998)*, pages 31–41. ACM Press, 1998.

[43] L. Zhang, S. Berson, S. Herzog, and S. Jamin. RFC 2205: Resource reservation protocol (rsvp) – version 1, functional specification, September 1997. URL http: //www.ietf.org /rfc /rfc2205.txt.