# Diplomarbeit

zum Thema

## Ressourcenverwaltung in DROPS

Jörg Nothnagel

Technische Universität Dresden
Fakultät Informatik
Institut für Systemarchitektur
Professur Betriebssysteme

22. Juli 2002

**Selbständigkeitserklärung**

Hiermit erkläre ich, daß ich diese Arbeit selbständig und nur mit den zugelassenen und aufgeführten Hilfsmitteln erstellt habe.

Dresden, der 22. Juli 2002

Jörg Nothnagel

# Inhaltsverzeichnis

# Abbildungsverzeichnis

# Kapitel 1

# Einleitung

Eine der Hauptaufgaben von Betriebssystemen besteht in der Verwaltung von Systemressourcen (z.B. Prozessorzeit, Speicher, Netzwerk). Diese Ressourcen werden Prozessen für die Durchführung ihrer Aufgabe zur Verfügung gestellt. Die Ressourcenvergabe erfolgt dabei unter Berücksichtigung von Kriterien, die je nach Einsatzzweck eines Systems variieren. Verteilte Betriebssysteme wie MOSIX streben beispielsweise eine möglichst gute Lastverteilung innerhalb des Systems an und weisen ihre Ressourcen entsprechend zu, andere Systeme vergeben Ressourcen nach anderen Gesichtspunkten, z.B. Fairneß. Voraussetzung für die Durchsetzung solcher Kriterien ist eine kontrollierte Ressourcenvergabe. Insbesondere ist es nötig, vergebene Ressourcen den benutzenden Prozessen zuzuordnen, um bei Verletzung der Systemvorgaben angemessen reagieren zu können.

Die Ressourcenverwaltung einer Echtzeitumgebung muß zudem in der Lage sein, Ressourcenbeträge im Vorfeld des Echtzeitbetriebes zu reservieren und deren Verfügbarkeit während der Ausführung der Echtzeitanwendung unter Einhaltung vereinbarter Eigenschaften zu garantieren.

Am Lehrstuhl Betriebssysteme der TU Dresden wird seit einigen Jahren an DROPS[1], einem Echtzeitbetriebssystem, gearbeitet. Das System basiert auf einem Mikrokern der 2. Generation. Die Ressourcenverwaltung wird durch eine Anzahl von Ressourcenmanagern realisiert, welche unabhängig voneinander agieren. Mechanismen, die eine Ressourcenvergabe unter Berücksichtigung von Systemvorgaben erlauben, existieren zur Zeit noch nicht.

Ziel eines in Zusammenarbeit mehrerer Lehrstühle der Fakultät Informatik durchgeführten Projekts (COMQUAD[2]) ist der Entwurf einer auf Komponenten basierenden Echtzeitumgebung in einem Rechnernetz. Die Grundfunktionalität soll durch DROPS bereitgestellt werden. Dazu muß die DROPS-Umgebung um oben angesprochene Eigenschaften der kontrollierten Ressourcenvergabe erweitert werden.

[1] **D**resden **R**eal Time **Op**erating **S**ystem

[2] **Com**ponents with **Qu**antitive Properties and **Ad**aptivity; `http://www.comquad.org`

Zielstellung dieser Arbeit ist der Entwurf solcher Funktionalität für DROPS. Insbesondere sollen eine einheitliche Schnittstelle zur Ressourcenreservierung bzw. -freigabe, ein Format zur einheitlichen Beschreibung von Ressourcen, eine Komponente, die Anwendungen bei der Ressourcenreservierung und -freigabe unterstützt sowie ein Mechanismus zur Durchsetzung von Ressourcenlimits entworfen werden. Im Rahmen einer Beispielimplementierung soll die Anwendbarkeit des Entwurfs gezeigt werden.

Diese Arbeit gliedert sich wie folgt: Im zweiten Kapitel werden für das weitere Verständnis wichtige Begriffe eingeführt und definiert. Im Kapitel *Stand der Technik* wird auf einige Projekte eingegangen, die sich mit dem hier betrachteten Thema beschäftigen sowie das DROPS-System und das COMQUAD-Projekt näher vorgestellt. Anschließend werden für die Umsetzung der Aufgabenstellung getroffene Entscheidungen im Kapitel *Entwurf* erläutert und begründet. Im Kapitel *Implementierung* wird dann im Rahmen einer Beispielimplementierung gezeigt, wie die Entwurfsentscheidungen umgesetzt wurden. Abschließend werden eine Zusammenfassung des Erreichten sowie Anregungen für die weitere Entwicklung dieses Projektes gegeben.

# Kapitel 2

# Theoretische Grundlagen

Dieses Kapitel führt in das Gebiet der Ressourcenverwaltung ein. Die Betrachtung beschränkt sich auf die in der Einleitung angedeuteten, für die Bearbeitung der Aufgabenstellung bedeutsamen Aspekte.

In einem Computersystem werden *Ressourcen* durch *Ressourcenmanager* kontrolliert und zur Verfügung gestellt. Die Gesamtheit der im System aktiven Ressourcenmanager wird durch eine *Steuerungsschicht* verwaltet. *Anwendungen* nutzen bereitgestellte Ressourcen zur Ausübung ihrer Aufgabe. Ressourcenmanager und Steuerungsschicht bilden die *Ressourcenverwaltung* des Systems. Ist die Ressourcenverwaltung in der Lage, Systemressourcen unter Einhaltung von Qualitätsgarantien anzubieten, spricht man von einem *QoS-Managementsystem*.

## 2.1 Ressource

Nach Goscinski [Gos91] sind Ressourcen (wiederverwertbare) Hardware- oder Softwarekomponenten eines Computersystems, die von Prozessen während ihrer Laufzeit angefordert, genutzt und wieder freigegeben werden.

Anstelle des Begriffes „Ressource“ werden auch die Begriffe „Dienst“ oder „Service“ verwendet.

In der Literatur werden Ressourcen verschiedenartig klassifiziert. Zwei dieser Klassifizierungen werden nun näher vorgestellt.

### 2.1.1 Low-level und high-level Ressourcen

Ressourcen können in low-level und high-level Ressourcen eingeteilt werden [Gos91].

**Low-level Ressourcen** bilden die grundlegenden Ressourcen eines Systems. Sie werden in physische und logische Ressourcen unterteilt.

Die Hardware eines Rechners (CPU, RAM, E/A-Geräte, etc.) bildet die Gruppe der physischen Ressourcen. Informationen, die in physischen Ressourcen gespeichert sind (z.B. Prozesse, Dateien, Namensräume), werden als logische Ressourcen bezeichnet.

**High-level Ressourcen** definieren höherwertige Dienste (z.B. Dateiservice, MPEG-Dekoder, virtueller Speicher) und nutzen dazu andere Ressourcen. Die Kombination von Ressourcen zu immer leistungsfähigeren Diensten führt zur Bildung einer Ressourcenhierarchie (siehe Abb. 2.1).

### 2.1.2 Aktive und passive Ressourcen

Liu [Liu00] klassifiziert Ressourcen entsprechend ihrer „Aktivität“ (siehe Abb. 2.1).

**Aktive Ressourcen** führen Anweisungen aus, transportieren Daten oder verarbeiten Anfragen. Jede Anwendung benötigt wenigstens eine aktive Ressource, um ihrem Ziel näher zu kommen. Aktive Ressourcen besitzen die Eigenschaft der „Ausführungsgeschwindigkeit“. Die CPU eines Rechners ist ein Beispiel für eine aktive Ressource.

**Passive Ressourcen** werden von Anwendungen neben aktiven Ressourcen benötigt, um arbeiten zu können. Passive Ressourcen sind z.B. Speicher oder Sperrvariablen. Passive Ressourcen besitzen das Attribut der „Ausführungsgeschwindigkeit“ nicht.

Die Zuordnung von Ressourcen zur aktiven bzw. passiven Gruppe ist vom betrachteten Modell abhängig.

## 2.2 Ressourcenmanager

Ein Ressourcenmanager verwaltet eine spezielle Ressource und stellt sie anderen Komponenten des Systems zur Verfügung. In Anlehnung an die im Abschnitt 2.1.1 getroffene Unterteilung für Ressourcen spricht man auch von low-level und high-level Ressourcenmanagern.

Ein Ressourcenmanager kann auch als „Server“ oder „Dienstanbieter“ bezeichnet werden.

## 2.3 Anwendung

Anwendungen sind Komponenten, die zur Erfüllung ihrer Aufgabe auf Ressourcen des Systems zurückgreifen. Insbesondere stellen high-level Ressourcenmanager Anwender anderer Ressourcenmanager dar. Somit ist jeder high-level Ressourcenmanager eine Anwendung.

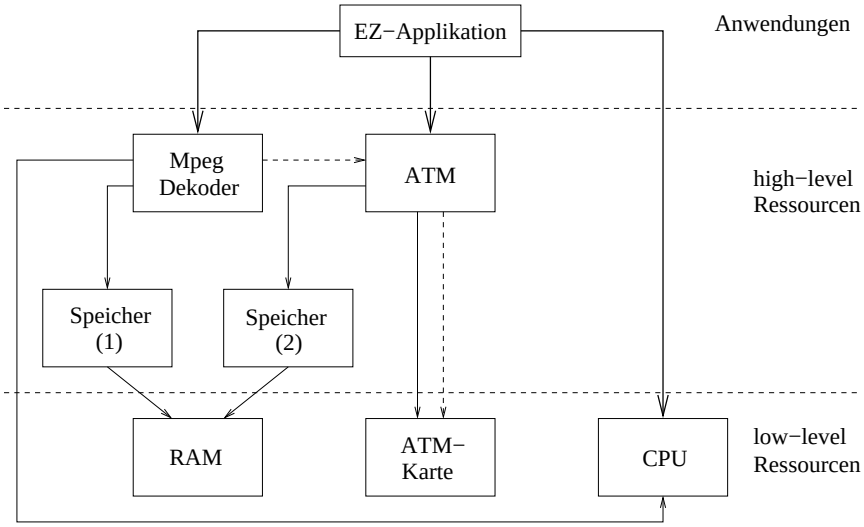


Abbildung 2.1: Eine Ressourcenhierarchie; Daten verarbeitende und somit aktive Ressourcen sind hier MPEG-Dekoder, ATM, Netzwerk und CPU. Der Datenfluß ist gestrichelt dargestellt.

Im Kontext dieser Arbeit erfolgt eine Einschränkung dieser Definition. Als Anwendung oder Applikation wird hier ein Dienstnutzer bezeichnet, der selber keinen Dienst zur Verfügung stellt. Eine Anwendung ist Initiator von Ressourcenforderungen (vgl. Abb. 2.1).

## 2.4 Ressourcenverwaltung

Als Ressourcenverwaltung (auch Ressourcenmanagement, Ressourcenmanagementsystem) wird der Teil eines Computersystems bezeichnet, der die Gesamtheit seiner Ressourcen verwaltet. Sie besteht aus einer Menge von Ressourcenmanagern und einer oder mehrerer Komponenten zur Steuerung des Zugriffs auf Ressourcen (Steuerungsschicht).

Die Steuerungsschicht eines Ressourcenmanagementsystems muß laut Goscinski [Gos91] Ressourcenanfragen von Nutzerprozessen entgegennehmen und die gewünschten Ressourcen allokieren oder bei Bedarf neu konstruieren. Die Anfrage eines Nutzerprozesses muß dazu eine ausreichende Beschreibung der geforderten Ressourcen enthalten (z.B. Ressourcentyp und -betrag). Einer Ressourcenzuteilung muß die Überprüfung von Sicherheitsaspekten wie z.B. Zugriffsbeschränkungen vorausgehen.

Nach der erfolgten Zuweisung von Ressourcen muß das Ressourcenmanagement sicherstellen, daß diese durch die entsprechende Anwendung fortlaufend genutzt werden können. Auf Fehlersituationen innerhalb der Ressourcenverwaltung (z.B. Ausfall eines Ressourcenmanagers) muß es angemessen reagieren.

Nutzer sollten die Möglichkeit haben, Statusinformationen ihrer Ressourcen abzurufen und diese Ressourcen wieder zurückzugeben.

Weiterhin müssen Mechanismen vorhanden sein, die die Aufnahme neuer Ressourcen in das System bzw. die Entfernung von Ressourcen aus dem System erlauben.

Im Kontext dieser Arbeit werden die eben beschriebenen drei Phasen der Ressourcennutzung als Reservierungs-, Nutzungs- und Freigabephase bezeichnet.

## 2.5 QoS-Managementsysteme

Als QoS[1]-Managementsystem wird eine Ressourcenverwaltung dann bezeichnet, wenn sie in der Lage ist, Ressourcen mit einer gewissen *Güte* anzubieten.

In einem solchen System können Anwendungen ihre Ressourcenforderungen um qualitative und quantitative Eigenschaften einer gewünschten Ressource erweitern. Eine qualitative Eigenschaft eines Speichermanagers wäre z.B. „pinned“ (Speicher, der bei einem Zugriff keine Seitenfehler und damit keine unkalkulierbaren Verzögerungen auslöst). Eine quantitative Eigenschaft wäre die Spezifizierung eines konkreten Speicherbetrags, z.B. „5 MB“.

Am Ende einer erfolgreichen Ressourcenreservierungsphase steht der Abschluß eines sogenannten QoS-Vertrags zwischen Anwendung und QoS-Managementsystem. Durch diesen Vertrag garantiert das QoS-Managementsystem, daß die geforderten Ressourcen mit den gewünschten Eigenschaften zur Verfügung stehen. Die fordernde Anwendung benötigt diese Garantien, um Aussagen über ihr eigenes Ausführungsverhalten zu treffen. Nur so kann sie entscheiden, ob sie ihre Aufgabe mit der von ihr geforderten Qualität ausführen kann.

Nach erfolgreicher Aushandlung eines QoS-Vertrages kann die Anwendung die spezifizierten Ressourcen mit den entsprechenden Eigenschaften nutzen.

Eine spezielle Ausprägung von QoS-Managementsystemen sind Echtzeitsysteme. Hier werden qualitative Eigenschaften geforderter Ressourcen durch Einhaltung von Zeitschranken realisiert. Anwendungen von Echtzeitsystemen, die Zusagen über die Qualität des Ergebnisses ihrer Arbeit treffen, werden als Echtzeitanwendungen bezeichnet.

[1] **Q**uality **o**f **S**ervice; Durchführung einer Aufgabe unter Einhaltung im Vorfeld spezifizierter Qualitätsanforderungen

# Kapitel 3

# Stand der Technik

In diesem Kapitel wird zunächst näher auf das Betriebssystem DROPS und das Projekt COMQUAD eingegangen.

Anschließend werden ausgewählte Projekte aus dem Gebiet der QoS-Managementsysteme vorgestellt. Die diesbezüglichen Ausführungen konzentrieren sich auf die für die Aufgabenstellung relevanten Teile der Abbildung von Ressourcenforderungen und der Beschreibung von Ressourcen.

Da das Problem der Mechanismen für die Durchsetzung von Ressourcenlimits in den vorgestellten Projekten nicht berührt wird, beschäftigt sich der abschließende Abschnitt des Kapitels mit diesem Teilaspekt des Ressourcenmanagements.

## 3.1 DROPS

Am Lehrstuhl Betriebssysteme der TU Dresden wird seit einigen Jahren an dem Betriebssystem DROPS[1] gearbeitet. Grundlage dieses Betriebssystems bildet eine Implementierung des L4-Mikrokerns [AH99], einem Mikrokern der 2. Generation. Der Mikrokernphilosophie folgend, sind klassische Betriebssystemkomponenten wie Treiber und Ressourcenmanager als Nutzerprozesse implementiert (siehe Abb. 3.1). Die Kommunikation erfolgt über den Mikrokern, welcher die sichere Nachrichtenübermittlung gewährleistet.

Ein Ziel des DROPS-Projektes ist die Unterstützung von Anwendungen mit „weichen“ Echtzeitanforderungen. Eine solche Anwendung kann mit der Tatsache umgehen, daß innerhalb eines QoS-Vertrages zugesicherte Ressourcen in Ausnahmefällen kurzzeitig nicht zur Verfügung stehen. Für den Benutzer einer Videoanwendung ist es beispielsweise zumutbar, wenn ab und zu ein Bild des Videos nicht angezeigt wird. Die Möglichkeit der Verletzung von QoS-Verträgen durch das System erlaubt eine weit bessere Ausnutzung

[1] **D**resden **R**eal Time **Op**erating **S**ystem [BBH+98]

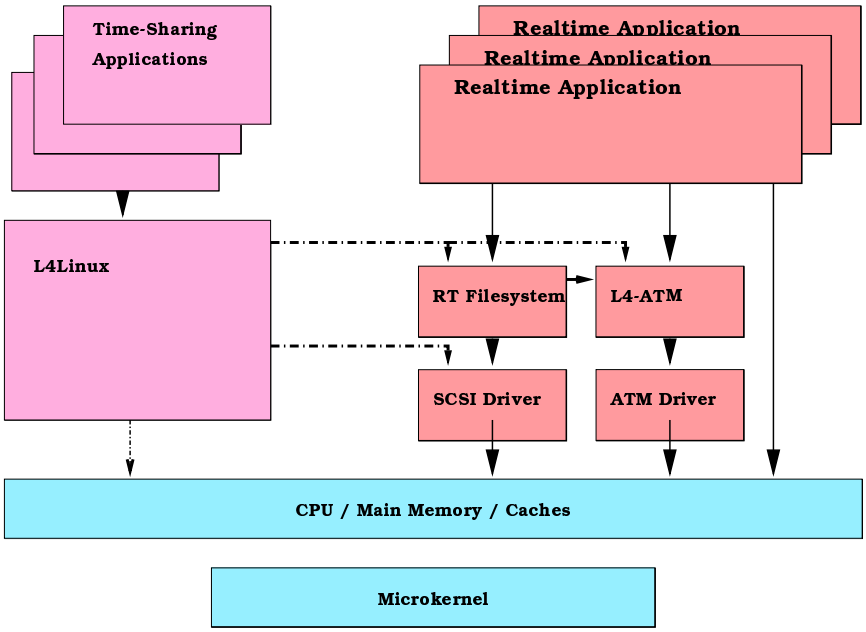


Abbildung 3.1: DROPS-Architektur (aus [HRW+99])

der vorhandenen Ressourcen, da nicht für den „worst case“ reserviert werden muß. Echtzeitfähigkeit unter DROPS wird durch den Mikrokern und echtzeitfähige Ressourcenmanager gewährleistet.

Ein weiteres Ziel besteht darin, Echtzeit- und Nicht-Echtzeitanwendungen parallel betreiben zu können. Im Rahmen von Referenzimplementierungen wurde gezeigt, daß beide genannten Ziele erreichbar sind [BH99].

Zur Zeit existierende Ressourcenmanager entstanden intuitiv, was dazu führte, daß ihre Schnittstelle zur Ressourcenreservierung erheblich voneinander abweichen. Härtig u. a. stellen deshalb in [HRW+99] ein einheitliches Design dieser Schnittstelle sowie Mechanismen der Abbildung von Forderungen auf andere Ressourcen, der Ressourcenverhandlung und der Änderung von QoS-Verträgen zur Laufzeit vor.

Neben einer einheitlichen Ressourcenreservierungsschnittstelle („Managementschnittstelle“) definiert [HRW+99] auch eine Schnittstelle für den Datenaustausch zwischen Ressourcenmanagern („Datenschnittstelle“) (siehe Abb. 3.2).

Die Managementschnittstelle dient der Ressourcenreservierung, der Abbildung von Ressourcen auf Ressourcenmanager sowie Kurz- bzw. Langzeitanpassungen (Neuverhandlung) im Falle der Verletzung von QoS-Verträgen durch unvorhergesehene Ressourcenknappheit. Sie soll von allen DROPS-Ressourcenmanagern implementiert werden.

Ressourcenmanager, die auf Datenströmen operieren (Manager aktiver Ressourcen im Sinne von 2.1.2), implementieren außerdem die Datenschnitt-

stelle. Über sie werden Daten transportiert und die Synchronisation zwischen den auf dem Datenstrom operierenden Komponenten vorgenommen.

Mit DSI[2] wurde ein Mechanismus implementiert, der die genannten Eigenschaften der Datenschnittstelle unterstützt.

Eine andere im Rahmen des DROPS-Projekts durchgeführte Arbeit schlägt ein einheitliches Format für die Spezifizierung periodisch wiederkehrender Ereignisse vor [HLR+01]. Dieses Modell erlaubt es, Forderungen an so verschiedene aktive Ressourcen wie CPU, Plattenbandbreite oder einen Video-Dekoder als „Jitter Constrained Periodic Stream“ einheitlich zu beschreiben. Die Eigenschaften „weicher“ Echtzeitanwendungen werden in diesem Modell dadurch unterstützt, daß solche Anwendungen den Prozentsatz der Forderungen angeben können, der vom System erfüllt werden muß. Für die Dekodierung eines MPEG-Videos könnte beispielsweise spezifiziert werden, daß 80 Prozent aller Bilder pro GOP[3] dargestellt werden soll.

Somit existiert eine Infrastruktur, die die Konstruktion von Echtzeitumgebungen grundsätzlich erlaubt. Verschiedene Funktionalität wie z.B die Unterstützung der Reservierung und Freigabe von Ressourcen durch einen einheitlichen Mechanismus, sowie Aspekte der Systemsicherheit, z.B. eine Möglichkeit zur Festlegung und Durchsetzung von Ressourcenlimits, fehlt jedoch. Die Entwicklung dieser Funktionalität ist Ziel dieser Arbeit.

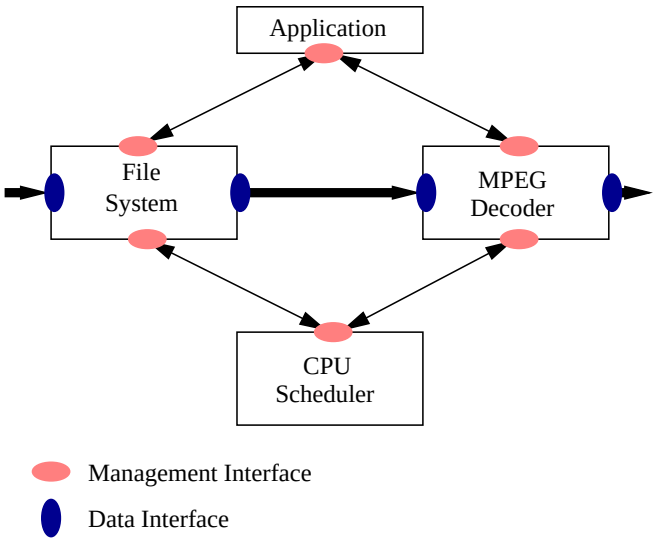


Abbildung 3.2: DROPS-Ressourcenmanager-Schnittstellen (aus [HRW+99])

[2] **D**ROPS **S**treaming **I**nterface [LHR01]

[3] **g**roup-**o**f-**p**ictures; Folge von Einzelbildern; Ein MPEG-Videostrom besteht aus einer Folge von GOPs

## 3.2 COMQUAD

Im Rahmen des COMQUAD-Projektes wird seit September 2001 an einer Systemarchitektur geforscht, die die Entwicklung adaptiver Software auf Grundlage von „Komponenten“ in einer Rechnernetzumgebung unterstützt. Eine solche Komponente stellt dem Anwender einen Dienst zur Verfügung. Neben Schnittstellen zur Erfüllung ihres Dienstes, bietet eine Komponente eine Schnittstelle an, die zur Spezifizierung zusagenfähiger, nicht-funktionaler Eigenschaften (z.B. Verweilzeit, Bildrate, Sicherheitseigenschaften) benutzt werden kann. Komponenten können kombiniert werden, um z.B. Funktionalität erhöhter Komplexität zu implementieren.

Grundlage einer solchen Komposition ist ein Vertrag, in dem beteiligte Komponenten Garantien über die Einhaltung o.g. nicht-funktionaler Eigenschaften geben. Voraussetzung für das Einhalten solcher Garantien ist, daß im Vertrag spezifizierte andere Komponenten ebenfalls ihre diesbezüglichen Zusagen einhalten. Ein MPEG-Dekoder kann beispielsweise nur dann garantieren, eine bestimmte Anzahl von Bildern pro Sekunde zu dekodieren (Forderung), wenn ihm ausreichend Speicher und Rechenzeit sowie eine entsprechende Eingangsdatenrate zugesichert werden kann. Verträge spezifizieren neben den Anforderungen für den „Normalbetrieb“ auch Regeln für den Fall, daß Schwankungen während der Laufzeit auftreten (z.B. Betriebsmittelknappheit durch Lasterhöhung).

Unter dem Begriff „nicht-funktionale Eigenschaften“ werden in COMQUAD qualitative und quantitative Eigenschaften zusammengefaßt, die über die Beschreibung der reinen Funktionalität eines Dienstes hinausgehen (siehe Abschnitt 2.5).

Das COMQUAD-Projekt wird auf DROPS aufsetzen und dessen Ressourcenmanager als eine untere Komponentenschicht nutzen. DROPS-Ressourcenmanager müssen dazu um eine einheitliche Managementschnittstelle erweitert werden. Die Spezifizierung einer solchen Schnittstelle ist Bestandteil dieser Arbeit.

## 3.3 Echtzeitfähige Ressourcenmanagementsysteme

Mit dem hier behandelten Thema des QoS-Managements haben sich während der letzten 10 Jahre viele Arbeitsgruppen weltweit beschäftigt. Die meisten existierenden Projekte konzentrieren sich auf das Gebiet der Übertragung und der Wiedergabe von Multimediaströmen (Video-on-demand, Telekonferenzen). Beispiele hierfür sind die Systeme GARA [CFK+98] und Darwin [CCF+00]. Dort stehen Reservierung und Optimierung von Netzwerkverbindungen im Vordergrund. Eine Architektur mit Augenmerk auf die Zusammenarbeit von Ressourcenmanagern auf einem Einzelrechner ist z.B. Rialto [JLD95].

Alle hier betrachteten Systeme haben gemein, daß der eigentlichen Arbeitsphase eine Ressourcenreservierungsphase vorausgeht. Auf die Notwendigkeit dieser Trennung wurde bereits im Punkt 2.4 hingewiesen.

Die meisten Systeme verfügen über Mechanismen, die die Änderung von QoS-Verträgen zur Laufzeit ermöglichen. So können Überlastsituationen gelöst und prioritätsgesteuerte Ressourcenvergabe durchgesetzt werden.

### 3.3.1 Rialto

Im Kontext der bei Mircrosoft durchgeführten Entwicklung des Betriebssystems Rialto, wurden Untersuchungen bezüglich eines modularen, verteilten Ressourcenmanagements unternommen. Ergebnisse und Ideen wurden in [JLD95] veröffentlicht.

Rialto stellt eine Echtzeit-Programmierumgebung auf Basis eines Echtzeitkerns bereit. Die Zielsetzung für das in Rialto verwendete Ressourcenmanagement ist folgende:

> „An extensible modular distributed real-time resource-management scheme with which programs can reason about their own real-time resource needs and negotiate for resource reservations based on those needs.“ [JLD95]

Rialto definiert für das Design seiner Ressourcenverwaltung folgende Komponenten (siehe Abb. 3.3):

- „Resource Provider“ (Ressourcenmanager)
- „Activitiy“ (Anwendung)
- „Resource Planner“ (Steuerungsschicht)

„Resource Provider“ besitzen neben der Schnittstelle für den Zugriff auf den implementierten Dienst eine weitere Schnittstelle, über die der Bedarf an Ressourcen ermittelt werden kann, die dieser „Resource Provider“ für seine Arbeit benötigt.

Die Ressourcenaushandlung wird wie folgt durchgeführt.

„Activities“ müssen ihren Ressourcenbedarf bestimmen und reservieren, bevor sie ihren Echtzeitbetrieb aufnehmen können. Dazu können sie die entsprechende Schnittstelle der „Resource Provider“ nutzen. Im Falle des in Abbildung 3.3 dargestellten Szenarios erfragt die „Activity“ ihren Bedarf bei den Ressourcenmanagern „FileSys“ und „Network“.

Handelt es sich bei dem kontaktierten Ressourcenmanager um einen high-level Manager, wandelt dieser die Anfrage in Forderungen an von ihm benötigte Ressourcen um und leitet diese an Ressourcenmanager weiter, die diese Ressource bereitstellen. Im Szenario der Abbildung 3.3 erfragt z.B.

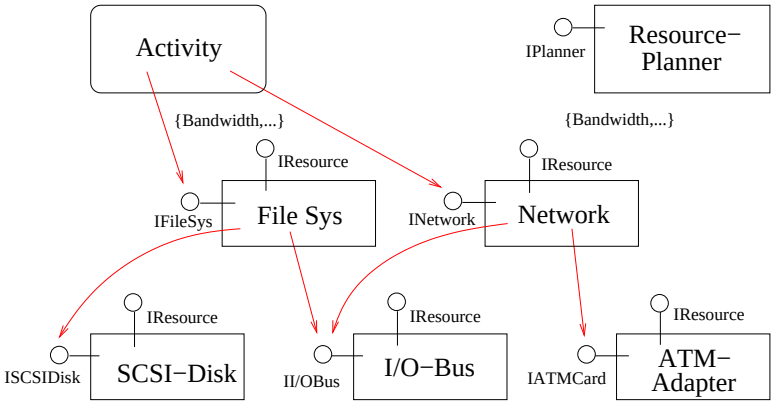


Abbildung 3.3: Komponenten in Rialto; eine „Activity“ während der Bestimmung ihres Ressourcenbedarfs (nach [JLD95]).

„FileSys“ seinen Bedarf bei „Scsi Disk“ und „I/O Bus“. Low-level Manager geben den geforderten Ressourcenbetrag als Prozentsatz der Gesamtressourcenmenge zurück.

Im Ergebnis erhält die Anwendung eine Menge von `(Ressource, Ressourcenbetrag)`-Paaren als Antwort auf ihre Anfrage und hat somit komplettes Wissen über die von ihr benötigten Ressourcen. Es wird nicht gefordert, daß Anwendungen ihre Bedarfsmenge auswerten müssen.

Für die eigentliche Reservierung übergibt die Anwendung die zuvor ermittelte Bedarfsmenge an den „Resource Planner“. Dieser entscheidet anhand der aktuellen Vergabestrategie, ob die geforderten Ressourcen reserviert werden können oder nicht. Im Namen der Anwendung reserviert er dann die benötigten Ressourcen (oder einen geringeren Betrag, falls die Kapazität erschöpft oder Konflikte mit der Vergabestrategie aufgetreten sind) und informiert die Anwendung über die tatsächlich reservierten Beträge.

Es wird davon ausgegangen, daß „Activities“ in mehreren Modi betrieben werden können. Verschiedene Modi entsprechen unterschiedlichen Qualitätsstufen mit entsprechend unterschiedlichen Ressourcenanforderungen.

Falls die geforderten Ressourcen durch das System nicht zur Verfügung gestellt werden konnten, liegt es im Ermessen der „Activity“, die Reservierung zu einem späteren Zeitpunkt erneut anzustoßen. Alternativ kann sie versuchen, Ressourcen für einen anderen Betriebsmodus zu reservieren oder sich aufgrund von Ressourcenmangel beenden.

Es liegt in der Verantwortung der „Resource Provider“, die Einhaltung von Reservierungen zu überwachen und den „Resource Planner“ zu kontaktieren, wenn „Activities“ versuchen, mehr als den für sie reservierten Betrag zu nutzen.

Anwendungen sind sich nicht darüber bewußt, ob sie lokale oder entfernte Ressourcen anfordern. Es ist Aufgabe des „Resource Planners“, entspre-

chende Anfragen an den „Resource Planner“ der entfernten Maschine zu übermitteln.

Eine einheitliche Form der Ressourcenbeschreibung ist nicht vorgesehen. Je nach Ressource variieren die Parameter, die der entsprechenden Ressourcen-Ermittlungs-Funktion übergeben werden.

Nach der Veröffentlichung von Jones u. a. [JLD95] wurde noch für einige Zeit an diesem System weitergeforscht. Allerdings ist es nicht über das Forschungsstadium hinaus fortgeführt worden.

Resultate dieses Projekts flossen in andere Arbeiten ein. Die für Rialto entworfenen Scheduling Mechanismen bilden beispielsweise die Grundlage für den Scheduler von „Rialto/NT“ [JR99], einer Forschungsversion von Windows NT.

#### Einschätzung

Der Ansatz enthält einige Aspekte, die sich gut für DROPS umsetzen lassen. Wie in DROPS wird von einer modularen Struktur des Systems mit unabhängigen Ressourcenmanagern ausgegangen.

Die Umwandlung des Ressourcenbedarfs in ressourcenunabhängige `(Ressource, Ressourcenbetrag)`-Paare erlaubt die einfache Aufnahme neuer Ressourcen in das System, ohne daß eine Änderung des „Resource Planners“ (er muß die so spezifizierte Bedarfsmenge auswerten) nötig ist.

Einen interessanten Aspekt stellt die Annahme dar, daß Anwendungen in verschiedenen Modi mit jeweils unterschiedlichen Anforderungen an das System arbeiten können.

Nachteil der Art und Weise der Bestimmung des Ressourcenbedarfs ist, daß „Activities“ direkt mit „Resource Providern“ kommunizieren müssen, was die Flexibilität des Systems bezüglich der Austauschbarkeit der Ressourcenmanager einschränkt und die Komplexität von Anwendungen erhöht, da die Ressourcenbestimmungsphase jeweils selbst implementiert werden muß.

### 3.3.2 GARA

Die meisten Projekte auf dem Gebiet des QoS-Managements beschäftigen sich mit der Realisierung von Ende-zu-Ende QoS-Garantien in Netzwerken, um Anwendungen wie „Video-on-demand“ zu unterstützen. Diese Systeme haben gemein, daß sie eine Vielzahl von Ressourcenmanagern verwalten müssen. Häufig werden hier nur spezielle Ressourcen wie CPU, Bandbreite oder Netzwerkknoten betrachtet, wodurch eine einfache Erweiterbarkeit um andere Ressourcen erschwert wird.

Während der Reservierungsphase werden die entsprechenden Komponenten der Steuerungsschicht oft mit der Situation konfrontiert, daß eine Anforderung mit Hilfe verschiedener Ressourcenmanager umgesetzt werden kann

(siehe Abb. 3.4). Die vorhandenen Alternativen zur Erfüllung der Anfrage müssen daher bewertet werden, um die beste auswählen zu können. Bei dieser Bewertung können neben den geforderten QoS-Parametern auch das Gesamtsystem betreffende Kriterien, wie Lastbalancierung, mit einfließen.

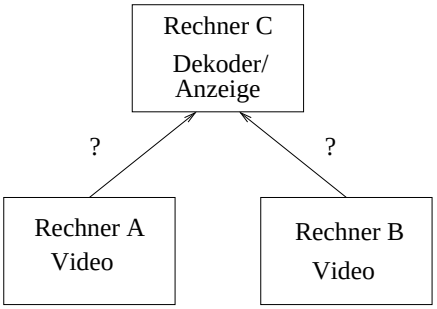


Abbildung 3.4: Szenario mit Alternativen; Für die Anfrage „Abspielen von Video auf Rechner C“ kann sowohl Rechner A als auch Rechner B als Datenquelle dienen.

Als ein Vertreter für ein QoS-Managementsystem aus diesem Gebiet soll GARA [FKL+99] nun näher betrachtet werden.

GARA steht für „Globus Architecture for Reservation and Allocation“ und baut auf den während des Projektes Globus [CFK+98] gewonnen Erfahrungen auf.

Zielsetzung des GARA-Projektes ist die Realisierung von Ende-zu-Ende QoS-Garantien für netzwerkbasierte Anwendungen. Der Schwerpunkt liegt dabei auf der Umsetzung folgender 4 Punkte:

- dynamisches Auffinden von Ressourcen
- Reservierung von Ressourcen für den sofortigen Gebrauch bzw. den Gebrauch zu einem zukünftigen Zeitpunkt
- Unterstützung beliebiger Ressourcentypen und -implementierungen
- Unterstützung unabhängig voneinander kontrollierter und verwalteter Ressourcen

In Abbildung 3.5 sind die im GARA-Szenario existierenden Objekte und ihre Interaktionen dargestellt.

GARA trennt Reservierung und Allokierung von Ressourcen. Dadurch wird es möglich, Reservierungen für einen in der Zukunft liegenden Zeitraum vorzunehmen, ohne schon in der Gegenwart Ressourcen belegen zu müssen. Es wird außerdem davon ausgegangen, daß Reservierungen oft „billiger“ sind als Allokierungen, da bei den beteiligten Ressourcenmanagern keine Objekte etc. erzeugt werden müssen. Eine Reservierung beinhaltet das Recht, die reservierten Ressourcen mit der zugesicherten Quantität und Qualität für

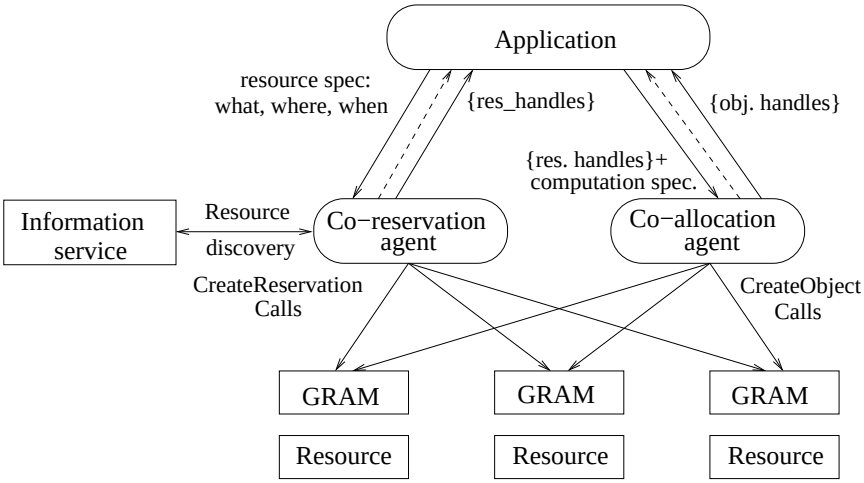


Abbildung 3.5: GARA-Architektur (nach [FKL+99]); die gestrichelten Linien symbolisieren „Up-Calls“, die zum Aufruf anwendungsspezifischer Funktionen genutzt werden können (z.B. Fehlerroutinen)

die vereinbarte Zeit nutzen zu können. Erst während der Allokationsphase führen beteiligte Ressourcenmanager Maßnahmen durch, die die Ressourcennutzung erlauben.

Für die Spezifizierung von Ressourcen und Ressourcenforderungen wurde eine erweiterbare Beschreibungssprache (RSL[4]) entworfen. Diese Sprache baut auf der Syntax von Filterspezifikationen auf, die innerhalb des LDAP[5] angewendet werden. LDAP bildet auch die Grundlage des in GARA verwendeten Namensdienstes, wodurch das Auffinden von Ressourcen ohne Konvertierung der Beschreibungen möglich ist.

RSL erlaubt die Beschreibung von Ressourcen und der geforderten Parameter durch „und“- bzw. „oder“-Verknüpfung einfacher Name-Wert-Paare (siehe Abb. 3.6).

```
&(executable=myprog)
(|(&(count=5)(memory>=64))
(&(count=10)(memory>=32)))
```

Abbildung 3.6: Beispiel für eine RSL-Beschreibung; spezifiziert wird hier eine Forderung nach entweder 5 Rechenknoten mit jeweils wenigstens 64 MB RAM oder 10 Rechenknoten mit jeweils wenigstens 32 MB RAM (aus [CFK+98]).

Im folgenden soll der typische Ablauf einer Ressourcenreservierung dargestellt werden (vgl. Abb. 3.5).

[4] **R**esource **S**pecification **L**anguage [FKL+99]
[5] **L**ightweight **D**irectory **A**ccess **P**rotocol; `http://www.openldap.org/`

Eine Anwendung ruft die `CreateReservation`-Funktion eines „Co-reservation agents“ auf und übergibt die Beschreibung der geforderten Ressourcen als RSL-Spezifikation. Mit Hilfe des „Information service“ lokalisiert das „Co-reservation agent“ in Frage kommende „GRAM[6]“s und leitet die Anfrage an diese weiter.

„GRAM“s verbergen die Heterogenität von Ressourcen und Ressourcenmanagern. Ein „GRAM“ authentifiziert Ressourcenanfragen unter Zuhilfenahme der „Globus Security Infrastructure“ (in Abb. 3.5 nicht dargestellt). Nach erfolgreicher Authentifizierung wird in Zusammenarbeit mit dem darunterliegenden Ressourcenmanager entschieden, ob die Forderung erfüllt werden kann und eine entsprechende Antwort zurückgegeben.

Das „Co-reservation agent“ konstruiert nach erfolgreicher Reservierung der geforderten Ressourcen ein `reservation-handle` und gibt es der Anwendung zurück.

Um die reservierten Ressourcen zu allokieren, wird dieses `reservation-handle` einem „Co-allocation agent“ übergeben. Dieses findet anhand der im `reservation-handle` enthaltenen Informationen die an der Reservierung beteiligten „GRAM“s und veranlaßt die Allokierung.

Nach erfolgreicher Allokierung aller Ressourcen werden `object-handle`s generiert und der Anwendung übergeben. Ein `object-handle` kann benutzt werden, um Informationen über die entsprechende Allokierung abzurufen bzw. diese freizugeben.

„Co-allocation agents“ und „Co-reservation agents“ spielen eine wichtige Rolle innerhalb der GARA-Welt, da sie sowohl die Schnittstelle zum Nutzer als auch zu den Ressourcen des Systems bilden. GARA erlaubt die Koexistenz vieler „agents“ in einer Umgebung und definiert lediglich deren Schnittstellen. Dieses Design erlaubt die Implementierung verschiedenster Funktionalität auf Ebene der „agents“. Ein „agent“ kann beispielsweise ausschließlich für eine einzelne Anwendung arbeiten, ein anderes kann für eine komplette Domäne verantwortlich sein. Erlaubt sind auch Hierarchien von „agents“.

Die GARA-Architektur wurde im Rahmen einer UNIX-Testumgebung in „C“ implementiert. Um die Einhaltung von QoS-Verträgen zu gewährleisten, wurden eigene low-level Ressourcenmanager entwickelt und in die verwendeten UNIX-Kerne integriert. Die Arbeiten an GARA sind noch nicht abgeschlossen.

### Einschätzung

Das in GARA vorgeschlagene Design ist auf verteilte Systeme mit vielen Ressourcenmanagern zugeschnitten und weist durch seine vielen beteiligten Komponenten („GRAM“, „Co-reservation agent“, „Co-allocation agent“,

[6] **G**lobus **R**esource **A**llocation **M**anager [FKL+99]

„Information service“, „Globus Security infrastructure“) eine hohe Komplexität auf. Für den in dieser Arbeit angestrebten ersten Entwurf eines QoS-Managementsystems ist der vorgestellte Ansatz daher nicht geeignet.

Einige Ideen wie z.B. die Verwendung einer ressourcenunabhängigen Beschreibungssprache, die zur Verbreitung von Forderungen im System genutzt wird und die Trennung von Reservierung sowie Allokierung sind trotzdem von Interesse, da sie auch in ein weniger komplexes System einfließen können.

Leider werden Struktur und Aufgabe der „Globus Security Infrastructure“ nicht genauer beschrieben. Handelt es sich dabei um eine Komponente mit globaler Sicht auf das System, kann hier die Durchsetzung von Ressourcenlimits u.ä. vorgenommen werden.

## 3.4 Durchsetzung von Ressourcenlimits

Neben der Entscheidung, ob Ressourcenforderungen von Anwendungen erfüllt werden können oder nicht, muß das Ressourcenmanagement sicherstellen, daß einzelne Anwendungen durch Belegung vieler freier Ressourcen nicht die Arbeit des Gesamtsystems behindern bzw. unmöglich machen. Für Anwendungen müssen daher Obergrenzen eingeführt werden, die belegbare Ressourcen beschränken. Da dieser Aspekt in den vorgestellten Systemen nicht behandelt wurde, für diese Arbeit aber relevant ist, soll nun gesondert darauf eingegangen werden.

### 3.4.1 Notwendige Funktionalität

Für die Umsetzung einer solchen Ressourcenbegrenzung sind folgende Voraussetzungen erforderlich:

- Belegte Ressourcen müssen den belegenden Anwendungen zugeordnet werden.
- Es müssen Ressourcenlimits für Anwendungen festgelegt werden.
- Bei drohender Verletzung dieser Grenzen müssen entsprechende Maßnahmen eingeleitet werden.

Ein System, das solche Grenzen festlegt und überwacht, ist JRes und wird im nächsten Abschnitt kurz erläutert.

### 3.4.2 JRes

JRes [CE98] unterstützt Ressourcenlimits für die Ressourcen heap-memory, CPU und Netzwerk auf Programmiersprachenebene (Java). Das beabsichtigte Einsatzgebiet sind erweiterbare Umgebungen wie z.B. Web-browser oder Internet-Server. Belegte Ressourcen werden Threads bzw. Thread-Gruppen

zugeordnet. Es existieren Schnittstellen, über die Threads ihren aktuellen Status bezüglich ihrer genutzten Ressourcen und ihre Ressourcenlimits erfragen können. Eine weitere Schnittstelle erlaubt es vertrauenswürdigen Komponenten, Grenzen festzulegen und Handles zu registrieren, die bei drohender Verletzung solcher Obergrenzen aufgerufen werden (siehe Abb. 3.7).

Bei ihrer Erzeugung werden Threads bei der „Accounting-Komponente“ angemeldet. Diese wird bei Ressourcenbelegung und -freigabe aufgerufen, aktualisiert die Ressourcenbelegungsstatistik des entsprechenden Threads und kontaktiert bei drohender Überschreitung der eingestellten Grenzen die für diesen Fall registrierten Handles.

### Einschätzung

Die Aktualisierung der Accounting-Komponente erfolgt in Zusammenarbeit mit den angeschlossenen Ressourcenmanagern. Für das in diesem Szenario angestrebte Umfeld von Nicht-Echtzeitanwendungen ist dieser Ansatz günstig, da der Ressourcenbedarf einer Anwendung nicht schon während der Reservierungsphase feststeht, sondern sich während der Laufzeit erhöhen kann.

Ein Nachteil der JRes-Implementierung besteht darin, daß die Schnittstellen der Accounting-Komponente ressourcenspezifisch sind, d.h. Parameter zur Festlegung von Obergrenzen für CPU oder heap-memory unterscheiden sich. Dadurch ist der Aufwand der Erweiterung des Accountings um weitere Ressourcen hoch.

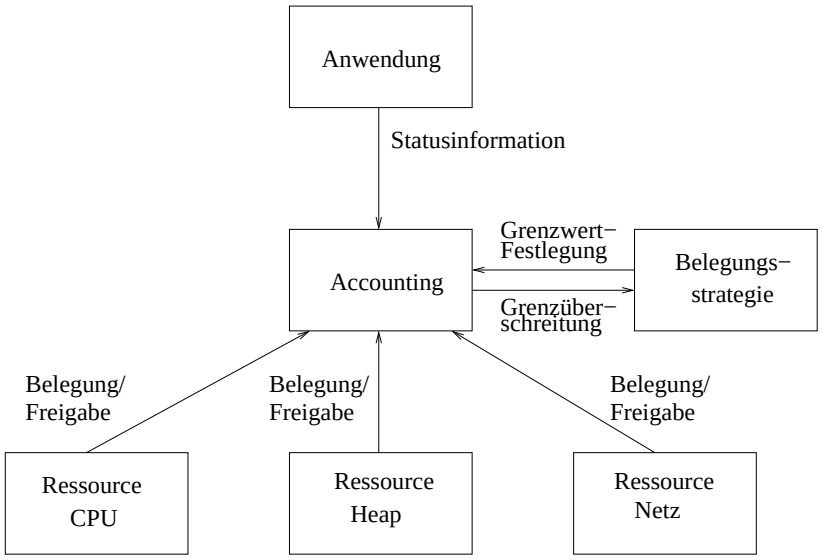


Abbildung 3.7: Schnittstellen des Ressourcen-Accounting Systems von JRes

### 3.4.3 Strategien für die Festlegung von Ressourcenlimits

Für die Art und Weise, wie Obergrenzen der Ressourcenbelegung für einzelne Anwendungen festgelegt werden, lassen sich zwei Ansätze unterscheiden.

#### Statischer Ansatz – Quotas

Diese Art der Festlegung von Ressourcenlimits wird oft in Multi-User UNIX-Systemen für die Ressource „Festplattenplatz“ angewandt. Bei diesem Verfahren wird jedem Nutzer des Systems ein Maximalbetrag zugeordnet, über den er frei verfügen kann. Eine Überschreitung dieser Grenze ist nicht möglich.

**Einschätzung** Hauptnachteile dieses Ansatzes bestehen in seiner Inflexibilität und in der darauf begründeten schlechten Ressourcenauslastung. Kapazitäten von Nutzern, welche nur Teile des ihnen zugeordneten Platzes belegen, liegen brach. Andere Nutzer, die (möglicherweise nur kurzzeitig) einen höheren Bedarf haben als durch ihre Quota festgelegt, können ihre Grenze nicht überschreiten, obwohl im System potentiell freie Kapazitäten vorhanden sind.

Vorteilhaft ist die einfache Umsetzbarkeit dieses Ansatzes, wodurch er für diese Arbeit interessant wird.

#### Dynamischer Ansatz – Ökonomische Modelle

Um die Nachteile des statischen Ansatzes aufzuheben, wurden verschiedene Modelle entwickelt, die Belegungsobergrenzen unter Berücksichtigung mehr oder weniger komplexer Gesetze des Marktes dynamisch bestimmen [HLR98, SHS99, WW94] .

Diese Modelle ordnen einer Ressource einen Preis zu. Preise können steigen bzw. sinken, wenn der verfügbare Betrag einer Ressource sinkt bzw. steigt. Anwendungen müssen für Ressourcen bezahlen. Sie tun dies mit „Geld“, welches sie vom System zugeteilt bekommen.

Als Vertreter für ein solchen Modells wird das von Heiser u. a. in [HLR98] für die Ressource „Plattenplatz“ beschriebene Verfahren nun vorgestellt.

Für jeden Nutzer des Systems wird ein „Bankkonto“ geführt. Das System schreibt diesem Konto periodisch einen gewissen Betrag gut. Nutzer müssen für jedes Objekt, das sie auf der Platte speichern, „Miete zahlen“, welche durch das System periodisch von ihrem Konto „abgebucht“ wird. Sinkt das „Guthaben“ eines Nutzers auf 0, kann kein weiterer Plattenplatz allokiert werden. Nutzer sind dann gezwungen, einige ihrer Objekte zu löschen und somit Ressourcen freizugeben.

Die „Miete“ für Plattenplatz steigt mit zunehmender Ressourcenknappheit.

**Einschätzung** Dynamische Verfahren sind komplexer als statische Verfahren, lösen aber den oben angesprochenen Nachteil der schlechten Ressourcenauslastung. Aufgrund ihrer Komplexität sind sie für den hier zu entwickelnden Prototyp ungeeignet.

# Kapitel 4

# Entwurf

In diesem Kapitel wird ein Entwurf von Teilen der im Kapitel *Einleitung* angeführten Aufgabenstellung für das Betriebssystem DROPS durchgeführt. Dabei werden im Kapitel *Stand der Technik* beschriebene Ideen aufgegriffen.

Nach einer Aufstellung der Hauptpunkte der Aufgabenstellung wird die Aufteilung der Funktionalität auf Komponenten vorgestellt und begründet. Durch Analyse verschiedener Szenarien erfolgt anschließend die Identifizierung benötigter Schnittstellen und weiterer Funktionalität. Nachdem der Entwurf für eine einheitliche Ressourcenbeschreibung vorgestellt wurde, erfolgt eine Zusammenfassung der am Ressourcenmanagement beteiligten Komponenten. Abschließend werden einige Probleme diskutiert, die im Zuge der Entwurfsphase auftraten.

Die Teile der Aufgabenstellung, die die Implementierung der Beispielanwendung betreffen, werden erst im Kapitel *Implementierung* betrachtet.

## 4.1 Konkretisierung der Aufgabenstellung

Die Aufgabenstellung umfaßt folgende Teilaspekte:

1. Abbildung von Ressourcenforderungen der Anwendungen auf Ressourcenmanager
2. Reservierung von Ressourcen bei Ressourcenmanagern im Namen der Anwendung
3. Freigabe reservierter Ressourcen
4. Festlegung und Einhaltung von Ressourcenlimits
5. Entwicklung einer einheitlichen Ressourcenmanagementschnittstelle
6. Entwicklung eines einheitlichen Ressourcenbeschreibungsformats

Die Punkte 1-4 beinhalten zu entwickelnde Funktionalität. Sie nutzt die in den Punkten 5 und 6 geforderten Schnittstellen und Formate.

## 4.2 Verteilung der Funktionalität

Der Entwurf der notwendigen Komponenten lehnt sich an das Design an, das für das Ressourcenmanagement von Rialto (siehe Abschnitt 3.3.1) vorgeschlagen wurde. Diese Entscheidung liegt darin begründet, daß sich die Umgebungen von DROPS und Rialto ähneln. Es handelt sich in beiden Fällen um Echtzeitumgebungen mit relativ wenigen beteiligten Ressourcenmanagern. Wie bei Rialto sind auch unter DROPS high-level Ressourcenmanager für die Umsetzung der an sie gerichteten Ressourcenanfragen in von ihnen benötigte Ressourcen verantwortlich (vgl. [HRW+99]). Eine weitere Gemeinsamkeit besteht darin, daß Ressourcenmanager für die Überwachung und Einhaltung einmal abgeschlossener QoS-Verträge verantwortlich sind. Desweiteren eignet sich das einfache Design von Rialto sehr gut für den hier angestrebten Entwurf eines experimentellen Prototyps.

Der als nachteilig dargestellte Mechanismus der Ermittlung des Ressourcenbedarfs durch die Anwendung wird durch ein an GARA (siehe Abschnitt 3.3.2) angelehntes Verfahren ersetzt. Das heißt, Anwendungen übergeben Ressourcenforderungen an das QoS-Managementsystem, welches die eventuell erforderliche Abbildung durch eine Hierarchie von Ressourcenmanagern übernimmt. Diese Vorgehensweise vereinfacht die Implementierung von Anwendungen.

Die Semantik des Betreibens von Anwendungen in verschiedenen Modi wird dadurch nachgebildet, daß für verschiedene Modi benötigte Ressourcenmengen innerhalb einer Anforderung übergeben werden. Das QoS-Management wählt unter Berücksichtigung von Systemrichtlinien eine dieser Alternativen für die Reservierung aus. Das Ressourcenbeschreibungsformat ist entsprechend konzipiert.

Im Hinblick auf ein von COMQUAD (siehe Abschnitt 3.2) angestrebtes verteiltes System mit vielen aktiven Ressourcenmanagern floß ein weiterer Bestandteil aus GARA in diesen Entwurf ein. Das QoS-Managementsystem soll die Möglichkeit des Vorhandenseins verschiedener Ressourcenmanager, welche die gleiche Funktionalität anbieten, berücksichtigen.

Aus Komplexitätsgründen beschränkt sich dieser Entwurf auf die Betrachtung des Einsatzes auf einem Einzelrechner. Ideen für die Erweiterung auf ein Rechnernetz und dafür nötige Änderungen von Komponenten werden im Abschnitt 4.6.3 diskutiert.

Ebenfalls aus Gründen der damit verbundenen Komplexität werden Änderungen bestehender QoS-Verträge hier nicht zugelassen. Eine Diskussion zu diesem Thema erfolgt im Abschnitt 4.6.4.

Für den Entwurf des Mechanismus der Verwaltung von Ressourcenlimits wurde berücksichtigt, daß unter DROPS Echtzeit- und Nicht-Echtzeitanwendungen parallel existieren können.

Die unter den Punkten 1-3 der Aufgabenstellung (siehe Abschnitt 4.1) aufgeführte Funktionalität wird vor bzw. nach der eigentlichen Arbeitsphase

von Anwendungen benötigt. Dabei spielt grundsätzlich keine Rolle, ob es sich dabei um Echtzeit- oder Nicht-Echtzeitanwendungen handelt. Informationen über Zuordnungen von Ressourcenanfragen zu Ressourcenmanagern, die während der Abbildung der Forderung gewonnen werden, sind sowohl für eine Reservierung als auch eine später erfolgende Freigabe notwendig. Daher ist es sinnvoll, diese Funktionalitäten innerhalb einer Komponente zu erfassen. Diese Komponente wird im folgenden QoS-Manager genannt.

Die Funktionalität der Verwaltung von Ressourcenlimits kann im Gegensatz dazu auch während der eigentlichen Arbeitsphase von Anwendungen benötigt werden, z.B. dann, wenn Nicht-Echtzeitanwendungen berücksichtigt werden sollen. Da sich diese Anwendungen ihres Ressourcenbedarfs nicht bewußt sind, können sie ihn während der Reservierungsphase nicht angeben. Somit ist eine endgültige Überprüfung des Ressourcenlimits während dieser Phase nicht möglich. Während ihrer Laufzeit kann ihr Bedarf an Ressourcen steigen, was die Überprüfung des Limits zu solchen Zeitpunkten erfordert.

Deshalb wurde diese Funktionalität in eine eigene Komponente ausgelagert, die als Quota-Manager bezeichnet wird.

Beide Komponenten sind Bestandteil der Steuerungsschicht des Ressourcenmanagements.

Die Kommunikationsbeziehungen zum Quota-Manager wurden denen des JRes-Projektes (siehe Abschnitt 3.4.2) nachempfunden. Ressourcenmanager übermitteln den Betrag ihrer Ressource, der einer Anwendung zugeordnet werden soll, bei jeder Änderung an den Quota-Manager, welcher entscheidet, ob diese Änderung durchgeführt werden darf, oder nicht.

Die Übermittlung an den Quota-Manager erfolgt nicht wie bei JRes ressourcenspezifisch, sondern als Prozentsatz des Gesamtbetrages der verwalteten Ressource. Die Umwandlung in dieses generische Format wurde auch in Rialto angewandt.

Der Vorteil der Darstellung in einem unabhängigen Format besteht in der einfachen Erweiterbarkeit des Quota-Managers um beliebige Ressourcen. Grenzen dieses Ansatzes werden im Abschnitt 4.6.2 analysiert.

Aus diesen Betrachtungen wurden die in Abb. 4.1 schematisch dargestellten Kommunikationsbeziehungen zwischen den verschiedenen Komponenten des QoS-Managementsystems abgeleitet. Als zusätzliche Komponente wird in dieser Abbildung ein „Lader“ eingeführt. Er ist für das Laden und Starten neuer Anwendungen verantwortlich und reserviert für den initialen Betrieb nötige Nicht-Echtzeit-Ressourcen im Namen der zu ladenden Anwendung. Typischerweise handelt es sich bei diesen Ressourcen um einen gewissen Betrag an Arbeitsspeicher und Prozessorzeit. Echtzeitanwendungen würden nach ihrem Start für ihren Echtzeitbetrieb erforderliche, zusätzliche Ressourcen in Eigenverantwortung reservieren (in der Abbildung nicht dargestellt).

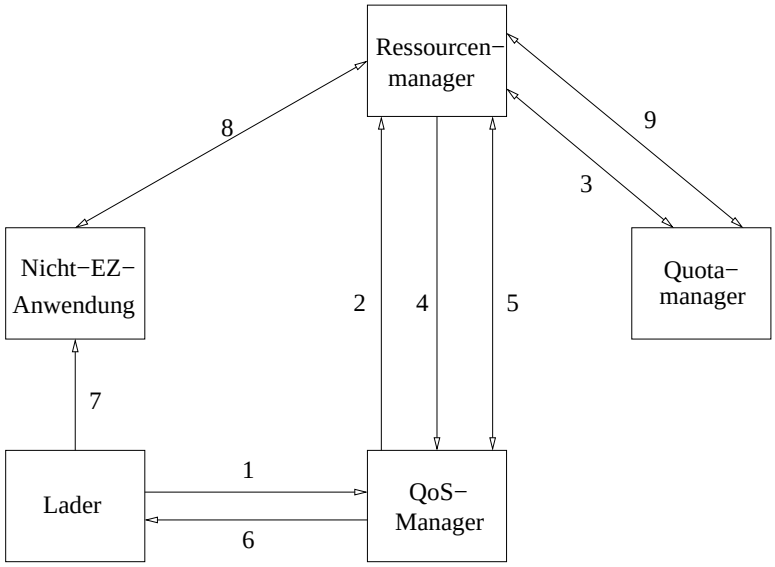


Abbildung 4.1: QoS-Management-Struktur mit Berücksichtigung von Nicht-Echtzeitforderungen; 1. Anfrage d. Laders; 2. Abbildung durch Ressourcenmanager-Hierarchie; 3. Quota-Test; 4. Rückkehr der Anfrage; 5. Reservierung; 6. Übergabe des QoS-Vertrags an Lader; 7. Initialisierung der Anwendung; 8. Nicht-EZ-Betrieb; 9. Anfrage bei Erhöhung des Ressourcenbedarfs

## 4.3 Analyse ausgewählter Szenarien

Zur Identifizierung weiterer Funktionalität sowie der für die Reservierung und Freigabe von Ressourcen benötigten Schnittstellen der Komponenten sollen folgende Szenarien betrachtet werden:

- Aufnahme eines Ressourcenmanagers in das System
- Laden und Start einer Anwendung
- Reservierung von Echtzeitressourcen
- Freigabe von Ressourcen

### 4.3.1 Aufnahme eines Ressourcenmanagers

Eine Aufgabe des Ressourcenmanagements besteht in der Aufnahme von Ressourcenmanagern in das System (siehe Abschnitt 2.4). Ein Ressourcenmanager muß sich dazu beim QoS-Managementsystem registrieren und den von ihm angebotenen Dienst ausreichend beschreiben. Anhand dieser Beschreibung muß das System dann in der Lage sein, eine korrekte Umsetzung

von Ressourcenforderungen durchzuführen und die entsprechenden Ressourcenmanager zu lokalisieren. Diese Funktionalität kann am besten im Rahmen eines Namensdienstes umgesetzt werden.

Im hier vorgestellten Umfeld ist der QoS-Manager die einzige Instanz, die die Funktionalität dieses Namensdienstes benötigt. Deshalb ist es sinnvoll, ihn in den QoS-Manager zu integrieren.

Die Anmeldung eines neuen Ressourcenmanagers erfolgt somit durch die Übergabe der Ressourcenbeschreibung sowie seiner Lokalisierungsinformation an den QoS-Manager.

#### Identifizierte Schnittstellen

Ressourcenmanager → QoS-Manager:

- `register(in: resource description, address)`

### 4.3.2 Laden und Start einer Anwendung

Im Abschnitt 4.2 wurde die Rolle eines Laders im Zusammenhang mit dem Start einer Anwendung aufgezeigt. In der DROPS-Umgebung wird diese Rolle durch den „L4 loader“ wahrgenommen. Die für den ersten Start einer Anwendung benötigten Ressourcen müssen ihr zur Verfügung gestellt werden. Der Lader benutzt dazu den weiter unten im Abschnitt 4.3.3 vorgestellten Mechanismus. Es muß allerdings sichergestellt werden, daß die angeforderten Ressourcen der zu startenden Anwendung angerechnet werden und nicht dem Lader als aufrufende Instanz. Aus diesem Grund ist es notwendig, eine ID der zu belastenden Anwendung beim Reservierungsaufruf mit zu übergeben.

#### Identifizierte Schnittstellen

Lader → QoS–Manager:

- `negotiate(in: resource request, on-behalf-of id, out: QoS-contract)`

### 4.3.3 Reservierung von Echtzeitressourcen

Wie im Abschnitt 2.5 dargelegt, sind sich Echtzeitanwendungen ihres benötigten Ressourcenbedarfs bewußt und übergeben eine Beschreibung dieses Bedarfs (evtl. sind alternative Ressourcenforderungen enthalten) an den QoS-Manager. Dieser führt eine Abbildung der beschriebenen Ressourcen auf Ressourcenmanager durch. Wie in Abbildung 3.2 dargestellt, existiert potentiell eine Hierarchie von Ressourcenmanagern, wobei Anwendungen nur die für sie sichtbare, oberste Ebene spezifizieren.

Bei dieser Abbildung wird in Betracht gezogen, daß eine Ressource von mehr als einem Ressourcenmanager angeboten werden kann, oder anders ausgedrückt, daß ein Dienst durch mehrere Anbieter realisiert wird. Der QoS-Manager muß somit Alternativen sowohl innerhalb der Beschreibung des Ressourcenbedarfs als auch innerhalb der Ressourcenmanager behandeln (siehe Abb. 4.2).

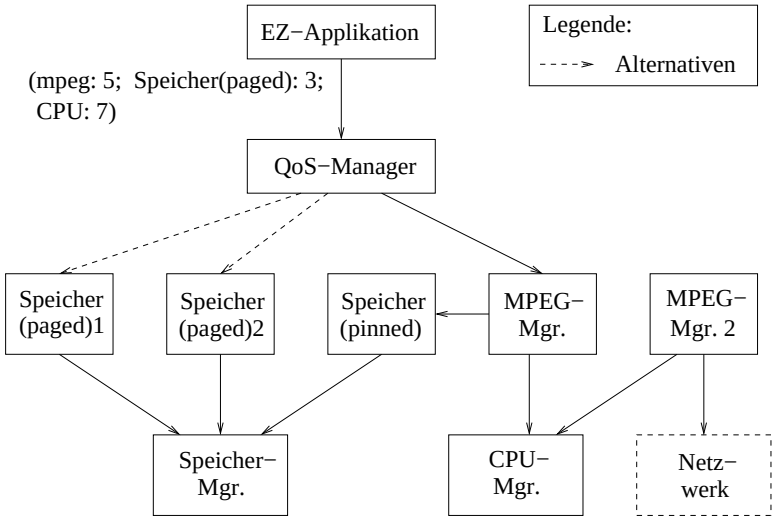


Abbildung 4.2: Alternativen bei der Ressourcenauswahl (Graph)
Der Pfad über „MPEG-Mgr. 2“ stellt keine wirkliche Alternative dar, da die von ihm benötigte Ressource „Netzwerk“ nicht zur Verfügung steht.

Die Reservierungsphase kann in 4 Stufen unterteilt werden:

- Ermittlung möglicher Kombinationen
- Vorreservierung gefundener Kombinationen
- Bewertung gefundener Kombinationen
- Reservierung einer ausgewählten Kombination

### Ermittlung möglicher Ressourcenmanager-Kombinationen

Das Ziel der ersten Stufe des Ressourcenreservierungsprozesses besteht in der Ermittlung von Ressourcenmanagerkombinationen, die die empfangene Ressourcenforderung potentiell erfüllen können. Kombinationen, die nicht auflösbar sind, weil benötigte Ressourcen nicht durch das System zur Verfügung gestellt werden (siehe Abb. 4.2), werden in diesem Schritt herausgefiltert, ohne daß andere Ressourcen reserviert werden müssen.

High-level Ressourcenmanager benötigen für die Durchführung ihrer Dienste andere Ressourcen. Sie müssen in der Lage sein, eine Anforderung in Beträge

von ihnen benötigter Ressourcen umzuwandeln und diesen Bedarf, der auch wieder Alternativen enthalten kann, an den QoS-Manager zurückgeben.

Der QoS-Manager löst die Hierarchie der Ressourcenmanager durch schrittweises Anfragen auf (siehe Abb. 4.3).

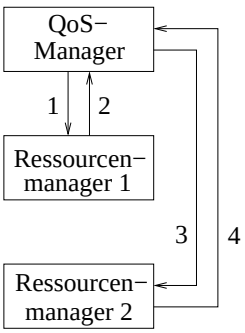


Abbildung 4.3: Aufrufreihenfolge zur Auflösung einer Ressourcen-Hierarchie; Ressourcenmanager 1 benötigt Ressource 2 zur Durchführung seiner Aufgabe.

Abbildung 4.4 veranschaulicht die Situation nach erfolgreicher Auflösung des in Abbildung 4.2 dargestellten Szenarios. Es entstanden 2 alternative Kombinationen (Alternativbäume). Die Existenz von Alternativen innerhalb der Ressourcenmanager-Strukturen kann dazu führen, daß eine Vielzahl von Kombinationen potentiell die Bedingungen einer einfachen Ressourcenforderung erfüllen. Während es in kleinen Systemen noch recht unproblematisch ist, alle Kombinationen zu finden, können Zeit- und Speicherbedarf im allgemeinen Fall unbegrenzt ansteigen. Deshalb muß in Systemen mit einer großen Anzahl von Ressourcenmanagern auf Verfahren zurückgegriffen werden, die mit einer Teilmenge aller Möglichkeiten arbeiten. Um später eine bezüglich der aktuellen Strategie der Ressourcenvergabe optimale Kombination für die Reservierung auswählen zu können, müssen dem QoS-Manager allerdings alle Varianten bekannt sein.

Um den allgemeinen Fall zu unterstützen, wurde ein Algorithmus verwendet, der es dem QoS-Manager erlaubt, Kombinationen unabhängig voneinander aufzufinden. Dadurch ist er in der Lage, den Suchprozeß beliebig abzubrechen, wodurch der Einsatz von sowohl optimalen als auch suboptimalen Auswahlstrategien ermöglicht wird. Dieser Algorithmus wird im Abschnitt 5.2.1 näher vorgestellt.

### Vorreservierung

In Vorbereitung der Reservierung werden im ersten Schritt gefundene Alternativbäume vorreserviert. Im Gegensatz zu der zuvor durchgeführten unverbindlichen Anfrage verpflichten sich Ressourcenmanager in diesem Schritt dazu, die geforderten Ressourcen vorzuhalten, sodaß eine endgültige Reservierung problemlos erfolgen kann.

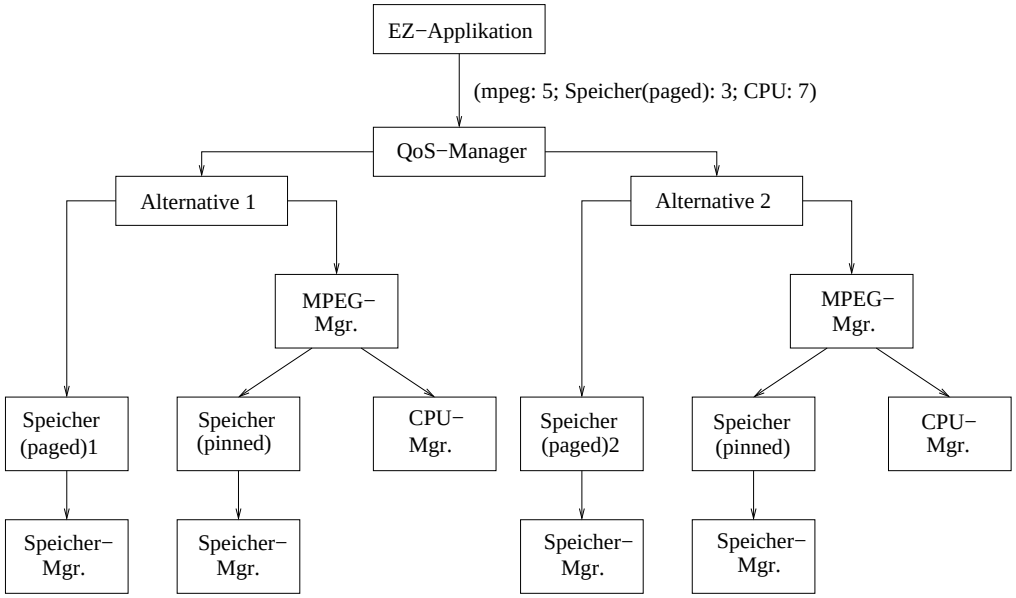


Abbildung 4.4: Alternativen bei der Ressourcenauswahl (Alternativbäume)

Jede erfolgreich vorreservierte Kombination erfüllt die Anforderung der Anwendung.

Die Vorreservierung einer Kombination schlägt fehl, wenn im System keine ausreichenden Ressourcen mit den geforderten Eigenschaften zur Verfügung stehen oder durch die Vorreservierung ein Ressourcenlimit der Anwendung überschritten wird.

Ressourcenlimits werden auf Ebene der low-level Ressourcenmanager überprüft. Eine diesbezügliche Diskussion wird im Abschnitt 4.6.2 geführt. Geprüft wird in diesem Schritt lediglich die *Möglichkeit* der Ressourcenzuteilung. Die endgültige Zuteilung und die damit verbundene Aktualisierung des durch die Anwendung belegten Ressourcenbetrags beim Quota-Manager erfolgt im Schritt *Reservierung*.

Im Hinblick auf eine nachfolgende Reservierung und Freigabe einer Ressourcenforderung ist es nötig, daß Vorreservierungen mit einer eindeutigen ID versehen werden (siehe Abschnitt 4.6.1).

### Bewertung gefundener Kombinationen

Existieren alternative Ressourcenkombinationen für die Erfüllung einer Ressourcenanfrage, hat der QoS-Manager die Aufgabe, eine dieser Alternativen für die endgültige Reservierung auszuwählen.

Ziel ist es, eine möglichst „günstige“ Ressourcenkombination zu ermitteln. Deshalb müssen die vorhandenen Alternativen bewertet werden.

Die Kriterien, die für eine solche Bewertung herangezogen werden (z.B. für Lastverteilung), sind von der Einsatzumgebung des Systems abhängig und werden durch die aktuelle Systemvorgabe bestimmt.

Im Kontext dieses Entwurfs kann eine Bewertung entweder vor oder nach der Vorreservierungsphase erfolgen.

Vor- und Nachteile dieser beiden Ansätze werden nun betrachtet.

**Bewertung vor der Vorreservierungsphase** Bei diesem Ansatz erfolgt die Bewertung einer gefundenen Kombination vor ihrer Vorreservierung.

Alternative Ressourcenkombinationen werden so lange erzeugt und bewertet, bis entweder alle Kombinationen gefunden wurden oder (bei Vorhandensein vieler Alternativen) der Auffindeprozeß unterbrochen wird.

Für die als am „günstigsten“ eingestufte Variante wird nun eine Vorreservierung durchgeführt. Schlägt sie fehl, werden vorreservierte Ressourcen wieder freigegeben und mit der nächsten Kombination fortgefahren.

Verlief die Vorreservierung einer Alternative erfolgreich, wird die endgültige Reservierung der entsprechenden Ressourcen vorgenommen.

Der Nachteil dieses Ansatzes besteht darin, daß Bewertungen auch für solche Kombinationen durchgeführt werden, die letztendlich wegen des Fehlschlagens der Vorreservierung fallengelassen werden müssen.

**Bewertung nach der Vorreservierungsphase** Die Bewertung einer Alternative erfolgt hier erst, nachdem ihre Ressourcen erfolgreich vorreserviert wurden. Dadurch ist sichergestellt, daß eine als „günstig“ und damit für die Reservierung vorgesehene Kombination tatsächlich reserviert werden kann. Somit wird die Bewertung nicht reservierbarer Alternativen vermieden.

Diese Vorgehensweise erfordert allerdings eine erhöhte Komplexität auf Seiten der Ressourcenmanager, da diese sicherstellen müssen, daß die Ressourcen einer beliebigen vorreservierten Alternative reserviert werden können.

**Einschätzung** Aus Sicht eines Ressourcenmanagers stellt die erste Vorgehensweise einen Spezialfall der zweiten dar. Änderungen des QoS-Managers, um die eine oder andere Strategie zu unterstützen, beschränken sich auf das Vertauschen von Vorreservierungs- und Bewertungsphase.

Daher wird im Rahmen dieses Entwurfs der 2. Ansatz betrachtet.

Um eine Grundlage für die Bewertung von Ressourcenkombinationen zu bieten, geben low-level Ressourcenmanager während der Vorreservierungsphase den zu reservierenden Betrag sowie den noch nicht vergebenen Betrag ihrer Ressource als Teil des Gesamtbetrages ihrer Ressource zurück (vgl. Betrachtungen zum Quota-Manager in Abschnitt 4.6.2).

### Reservierung

In dieser Phase werden die Ressourcen der im vorhergehenden Schritt ausgewählten Kombination reserviert, d.h. die zugesagten Dienste müssen soweit eingerichtet werden, daß der Zugriff unter den gestellten Bedingungen erfolgen kann. High-level Ressourcenmanager müssen physischen Zugriff auf die von ihnen benötigten Ressourcen erhalten.

Es ist daher erforderlich, diesen Schritt auf der Stufe der low-level Manager zu beginnen. Diese führen notwendige interne Maßnahmen zur Dienstbereitstellung durch, veranlassen die Aktualisierung der Statistik der durch die aufrufende Anwendung belegten Ressourcen des Quota-Managers und generieren ein „Handle“, das den Zugriff auf diesen Dienst erlaubt. Diese „Handles“ werden den entsprechenden high-level Managern in Form eines QoS-Vertrags übergeben.

Der QoS-Vertrag wird durch den QoS-Manager generiert und enthält neben den benötigten „Handles“ u.a. Informationen darüber, welche der während der Vorreservierungsphase angegebenen Alternativen reserviert wurde. High-level Ressourcenmanager richten daraufhin ihren Dienst ein und generieren ihrerseits ein „Handle“ auf ihre Ressource.

Ein QoS-Vertrag für die von der Anwendung benötigten Ressourcen wird ihr übergeben. Die Freigabe anderer vorreservierter Ressourcen wird vom QoS-Manager veranlaßt.

Die während der Vorreservierung generierte ID dient den Ressourcenmanagern zur Identifizierung der zu reservierenden Ressourcenforderung.

### Identifizierte Schnittstellen

Anwendung → QoS-Manager:

- `negotiate(in: resource request, out: QoS-contract)`

QoS-Manager → Ressourcenmanager:

- `resources_needed(in: resource request, out: needed resources)`
- `pre_reserve(in: resource request, reservation id, out: needed resources)`
- `reserve(in: QoS-contract, reservation id, out: handle to this resource)`
- `release(in: reservation id)`

Ressourcenmanager → Quota-Manager:

- `quota_check(in: resource description, application id, amount, out: (not) allowed)`
- `quota_add(in: resource description, application id, amount, out: (not) allowed)`

### 4.3.4 Freigabe von Ressourcen

Mehrere Ursachen führen dazu, daß Ressourcen freigegeben werden. Beendet beispielsweise eine Anwendung ihren Echtzeitbetrieb, wird sie die nun nicht mehr benötigten Ressourcen wieder an das System zurückgeben.

Eine andere Situation tritt ein, wenn eine Anwendung beendet und damit aus dem System entfernt werden soll. In diesem Fall müssen alle dieser Anwendung zugeordneten Ressourcen, einschließlich derjenigen, die der Anwendung durch den Lader (siehe Abschnitt 4.3.2) zugewiesen wurden, aufgegeben werden.

Diese beiden Fälle werden nun betrachtet.

#### Freigabe durch die reservierende Anwendung

Die Ressourcenfreigabe erfolgt wie die Ressourcenreservierung mit Unterstützung des QoS-Managers. Dabei dient die während der Vorreservierungsphase generierte ID, die Bestandteil des QoS-Vertrags ist, als Identifikator.

Die Anwendung übergibt diese ID an den QoS-Manager. Dieser besitzt Informationen über die an der so gekennzeichneten Reservierung beteiligten Ressourcenmanager und veranlaßt diese Manager zur Freigabe der entsprechenden Ressourcen. Low-level Ressourcenmanager aktualisieren die Ressourcenbelegungsstatistik des Quota-Managers durch entsprechende Aufrufe.

#### Freigabe bei Beendigung einer Anwendung

Das Beenden und damit verbundenen Entfernen einer Anwendung aus dem System kann nicht durch die betroffene Anwendung selbst durchgeführt werden, sondern muß, wie das Laden der Applikation, durch eine dafür verantwortliche Instanz erfolgen. Unter DROPS ist hierfür wiederum der „L4 loader“ zuständig. Diese Komponente übergibt dem QoS-Manager die ID der zu beendenden Anwendung, welcher daraufhin die Freigabe nach dem oben beschriebenen Mechanismus steuert.

#### Identifizierte Schnittstellen

Anwendung → QoS-Manager:

- `release(in: reservation id)`

Lader → QoS-Manager:

- `release_all(in: thread id)`

QoS-Manager → Ressourcenmanager:

- `release(in: reservation id)`

Ressourcenmanager → Quota-Manager:

- `quota_sub(in: resource description, application id, amount)`

## 4.4 Format von Ressourcenforderungen

Die Beschreibung einer Ressourcenforderung sollte folgenden Anforderungen genügen.

1. Der QoS-Manager muß in der Lage sein, eine Abbildung auf Ressourcenmanager vorzunehmen.
2. Es müssen geforderte qualitative und quantitative Eigenschaften einzelner Ressourcen spezifiziert werden können.
3. Die Aufnahme neuer Ressourcen soll ohne Änderungen des QoS-Managers möglich sein.

Diese Ziele können erreicht werden, indem die Beschreibung in einen ressourcenunabhängigen (generischen) und einen ressourcenspezifischen Teil zerlegt wird.

Der ressourcenunabhängige Teil der Beschreibung wird durch den QoS-Manager ausgewertet und enthält Informationen, die für das Auffinden eines die Ressource anbietenden Ressourcenmanagers benötigt werden. Im einfachsten Fall besteht dieser Teil aus einem eindeutigen Ressourcennamen (z.B. „Speicher“).

Diese einfache Beschreibung ist jedoch sehr ungenau. Es ist beispielsweise vorstellbar, daß in einem System verschiedene Anbieter der Ressource „Speicher“ existieren, welche aber jeweils eine andere Qualität (z.B. „pinned“, „paged“, ...) zur Verfügung stellen.

Das Auffinden geeigneter Ressourcenmanager kann durch eine genauere Beschreibung von Ressourcen unterstützt werden. Der hier vorgeschlagene generische Teil der Beschreibung einer Ressourcenforderung besteht deshalb aus dem eindeutigen Ressourcennamen und weiteren qualitativen Eigenschaften der gewünschten Ressource.

Der ressourcenspezifische Teil der Beschreibung enthält konkrete Angaben über die quantitativen Eigenschaften der benötigten Ressource. Da diese Angaben von Ressource zu Ressource verschieden sind, werden sie nicht vom QoS-Manager ausgewertet, sondern an diejenigen Ressourcenmanager weitergegeben, die anhand des generischen Teils der Beschreibung ausgewählt wurden.

Die Beschreibung seines Dienstes, die ein Ressourcenmanager bei seiner Anmeldung an das System übergibt, entspricht dem generischen Teil einer Ressourcenforderung.

Als Trägersprache von Ressourcenbeschreibungen wird XML[1] vorgeschlagen. XML ist eine wohl definierte Sprache und erlaubt es, eigene strukturierte Dokumententypen als DTD[2] zu spezifizieren und durch entsprechende Tools zu validieren. Weiterhin ist die Darstellung gut lesbar und damit benutzerfreundlich. Benötigte Tools wie Parser und Validierer sind frei verfügbar.

Der Nachteil des unter Umständen hohen Aufwandes für Erstellung, Übertragung und Auswertung einer XML-Struktur wird in Kauf genommen.

## 4.5 Eigenschaften der Komponenten

Die Anforderungen an die Komponenten des QoS-Managementsystems werden nachfolgend zusammengefaßt.

### 4.5.1 QoS-Manager

Der QoS-Manager unterstützt Anwendungen bei Reservierung und Freigabe von Ressourcen und stellt einen Mechanismus bereit, der die Zuordnung von Ressourcen zu den benutzenden Anwendungen erlaubt.

Er implementiert folgende Schnittstellen.

- `register(in: resource description, address)`
  Registriert einen Ressourcenmanager mit `address` und der Beschreibung seines Dienstes

- `release(in: reservation id)`
  Veranlaßt die Freigabe der unter `reservation id` reservierten Ressourcen

- `release_all(in: thread id)`
  Veranlaßt die Freigabe aller Ressourcen, welche die durch `thread id` bezeichnete Anwendung reserviert hat

[1] **E**x**t**ensible **M**arkup **L**anguage; `http://www.w3.org/TR/REC-xml`
[2] **D**ocument **T**ype **D**efinition

- `negotiate(in: resource request, out: QoS-contract)`
  Reserviert die in `resource request` geforderten Ressourcen und gibt bei Erfolg einen QoS-Vertrag zurück, der nötige Informationen für den Zugriff auf die reservierten Ressourcen enthält; bei Mißerfolg der Reservierung kommt kein QoS-Vertrag zustande

### 4.5.2 Quota-Manager

Der Quota-Manager ist für die Überwachung von anwendungsspezifischen Ressourcenlimits zuständig. Er bietet die nachfolgend aufgeführten Schnittstellen an. Diese werden von Klienten des Quota-Managers genutzt, um den Quota-Manager mit aktuellen Informationen über vergebene Ressourcen zu versorgen und die eventuelle Überschreitung eines Limits zu erfragen.

Schnittstellen:

- `quota_check(in: resource description, application id, amount, out: (not) allowed)`
  Test, ob einer Erhöhung stattgegeben werden kann.
- `quota_add(in: resource description, application id, amount, out: (not) allowed)`
  Test, ob einer Erhöhung stattgegeben werden kann und Aktualisierung der Statistik, falls keine Überschreitung vorliegt.
- `quota_sub(in: resource description, application id, amount)`
  Aktualisierung der Statistik (Verminderung)

### 4.5.3 Ressourcenmanager

Aus der Betrachtung der vorherigen Abschnitte ergibt sich folgende Managementschnittstelle für Ressourcenmanager.

- `resources_needed(in: resource request, out: needed resources)`
  Umwandlung von `resource request` in `needed resources`
- `pre_reserve(in: resource request, reservation id, out: needed resources)`
  Umwandlung von `resource request` in `needed resources`; Vorhaltung der geforderten Ressourcen für eine mögliche Reservierung; low-level Ressourcenmanager geben in `needed resources` den zu belegenden und den noch freien Anteil der Gesamtressource zurück.
- `reserve(in: QoS-contract, reservation id, out: handle to this resource)`
  Einrichtung des durch `reservation id` identifizierten vorreservierten

Dienstes unter Benutzung der in `QoS-contract` zugänglich gemachten Ressourcen.

- `release(in: reservation id)`
  Freigabe der durch `reservation id` identifizierten, reservierten oder vorreservierten Ressourcen

### Eigenschaften konformer Ressourcenmanager

Zu diesem Entwurf konforme Ressourcenmanager müssen die oben beschriebene Managementschnittstelle implementieren und dürfen nur die zur Erfüllung ihrer Aufgabe nötigen Ressourcen belegen. Sie registrieren sich beim QoS-Manager und übergeben eine Beschreibung ihres Dienstes, welche die im Abschnitt 4.4 beschriebenen Eigenschaften aufweist.

Während der Vorreservierungsphase müssen Ressourcenmanager für eine Anwendung geforderte Ressourcenbeträge verschiedener Alternativen so vorreservieren, daß eine beliebige dieser Alternativen endgültig reserviert werden kann.

Während der Reservierungsphase müssen sie „Handles“ für jede Ressourcenforderung der zu reservierenden Alternative generieren. Der „Speichermanager“ des in Abbildung 4.4 dargestellten Szenarios muß also jeweils ein „Handle“ für den Zugriff auf die für „Speicher(paged)1“ und die für „Speicher(pinned)“ geforderten Beträge generieren, wenn die Ressourcen für „Alternative 1“ reserviert werden.

Nach erfolgreicher Reservierung müssen die Ressourcen mit den geforderten Eigenschaften zur Verfügung stehen.

High-level Ressourcenmanager müssen Ressourcenforderungen in Beträge von ihnen benötigter Ressourcen umwandeln können. Dabei muß sichergestellt sein, daß kein Ressourcenbedarf entsteht, der nicht der aufrufenden Anwendung zugeordnet werden kann. Anwendungsunabhängige Ressourcen (z.B. Speicher für Farbraumkonvertierungstabellen eines MPEG-Dekoders) müssen daher schon beim Start des Ressourcenmanagers belegt werden.

Low-level Ressourcenmanager müssen Ressourcenforderungen in das oben spezifizierte unabhängige Format überführen können. Bevor sie einen Ressourcenbetrag vergeben, stellen low-level Ressourcenmanager mit Hilfe des Quota-Managers sicher, daß die aufrufende Anwendung zur Nutzung dieses Betrags berechtigt ist.

Bei einem Aufruf von `release(reservation id)` müssen alle unter der ID `reservation id` reservierten bzw. vorreservierten Ressourcen freigegeben werden.

Ressourcenmanager sind dafür verantwortlich, daß Klienten keinen höheren als den zugesicherten Ressourcenbetrag nutzen.

Desweiteren müssen Ressourcenmanager die für ihre Ressource definierte Schnittstelle für den Zugriff auf die angebotene Funktionalität implemen-

tieren. Es wird davon ausgegangen, daß Anwendungen diese Schnittstelle kennen. Eine Überprüfung dieser Konformität durch das QoS-Managementsystem ist nicht vorgesehen.

### 4.5.4 Anwendung

Anwendungen können Ressourcen mit Hilfe des `negotiate()`-Aufrufs des QoS–Managers reservieren und geben diese Ressourcen durch einen `release()`-Aufruf wieder frei. Sie übermitteln ihren Bedarf an Ressourcen in einer Form, die den im Abschnitt 4.4 beschriebenen Anforderungen genügt.

## 4.6 Diskussion von Teilaspekten

In diesem Abschnitt werden einige Probleme erläutert, die während der Entwurfsphase auftraten. Nicht alle wurden im Rahmen dieser Arbeit gelöst. Sie werden hier als Anregungen für die weitere Entwicklung des DROPS-QoS-Managements mit aufgeführt.

### 4.6.1 Reservierungs-IDs

Während der Vorreservierungsphase von Ressourcen erzeugt der QoS-Manager eindeutige Identifikatoren, die zusammen mit einer Ressourcenanfrage an beteiligte Ressourcenmanager übergeben werden. Eine Reservierungs-ID hat die in Abbildung 4.5 dargestellten 4 Bestandteile.

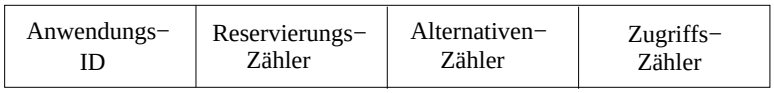


Abbildung 4.5: Aufbau einer Reservierungs-ID

- „Anwendungs-ID“ beinhaltet einen eindeutigen Identifikator für die reservierende Anwendung.
- „Reservierungszähler“ enthält einen eindeutigen Wert für jede Reservierung.
- „Alternativenzähler“ enthält einen eindeutigen Wert für jede Alternative, deren Ressourcen im Zuge einer Vorreservierungsphase (siehe Abschnitt 4.3.3) vorgehalten werden sollen.
- „Zugriffszähler“ enthält einen eindeutigen Wert für jede zu einer Alternative gehörenden Anfrage.

Anhand dieser Identifikatoren ist ein Ressourcenmanager während der Vorreservierungsphase in der Lage, die anfordernde Anwendung zu identifizieren.

Wie im Abschnitt 4.3.3 beschrieben, veranlaßt der QoS-Manager die Vorreservierung verschiedener alternativer Kombinationen, um die für das System „günstigste“ zu ermitteln. Mit Hilfe des „Alternativenzählers“ der Reservierungs-ID kann ein Ressourcenmanager entscheiden, ob verschiedene Anfragen einer Anwendung unterschiedlichen Alternativen angehören. Diese Unterscheidung ist nötig, wenn man bedenkt, daß letztendlich nur Ressourcen für eine Alternative endgültig reserviert werden. Damit ist es ausreichend, wenn ein Ressourcenmanager den Ressourcenbetrag der „teuersten“ Alternative vorreserviert.

Verschiedene Ressourcenforderungen innerhalb einer Alternative müssen durch den Ressourcenmanager unterschieden und korrekt zugeordnet werden können. Im Szenario der Abbildungen 4.2 und 4.4 muß der Manager „Speicher“ für die Manager „Speicher (pinned)“, „Speicher (paged)1“ und „Speicher (paged)2“ angeforderte Ressourcen unterscheiden und die gewünschten Beträge während des endgültigen `reserve()`-Aufrufs reservieren und zugänglich machen.

Ein vorreservierter Ressourcenbetrag wird durch Angabe der zugehörigen Reservierungs-ID im Zuge eines `reserve()`-Aufrufs reserviert und später während der Ausführung der `release()`-Funktion wieder freigegeben.

Die für eine reservierte Ressource vergebene Reservierungs-ID wird als Teil des QoS-Vertrags an die die Ressource benutzende Komponente übergeben. Wird diese ID während der Nutzungsphase als Parameter für Aufrufe der funktionalen Schnittstelle übergeben, kann auf diesem Wege eine Authentifizierung des Klienten und eine Überwachung des QoS-Vertrages erfolgen.

Der „Reservierungszähler“ der Reservierungs-ID dient zur Unterscheidung verschiedener Reservierungen einer Anwendung.

Die L4-Thread-ID erlaubt in der aktuellen L4-Version eine eindeutige Identifizierung von Threads und Tasks. Deshalb wird sie als Identifikator im „Anwendungs-ID“-Feld der Reservierungs-ID verwendet.

Diese Zuordnung von Threads zu Tasks wird durch die zur Zeit in Entwicklung befindliche Version 4 von L4 nicht mehr unterstützt. Bei einer Portierung auf diesen L4-Standard muß daher der Mechanismus der Zuordnung einzelner Threads zu einer Task angepaßt werden.

### 4.6.2 Grenzen des Quota-Managers

Der Quota-Manager erhält seine Informationen in einem generischen Format, welches einen Ressourcenbetrag als Teil des Gesamtbetrages dieser Ressource darstellt. Die Begründung für die Wahl dieser Vorgehensweise wurde im Abschnitt 4.2 gegeben.

Die Umwandlung einer Ressourcenforderung in dieses generische Format kann auf Ebene der low-level Ressourcenmanager einfach erfolgen, da sie eine Ressource anbieten, deren Gesamtkapazität ihnen bekannt sein sollte.

Für high-level Ressourcenmanager ist diese Umwandlung nicht trivial. Ein MPEG-Dekoder kann beispielsweise aus seiner Sicht unbegrenzt viele Anforderungen parallel bearbeiten. Das System muß ihm lediglich die geforderten Ressourcen wie Rechenzeit, Arbeitsspeicher und Eingangsdatenstrom in der benötigten Qualität und Quantität zur Verfügung stellen. Der Anteil einer einzelnen Forderung an seiner Gesamtkapazität ist somit nicht einfach angebbar.

Aus diesem Grunde wird in diesem Entwurf vorgeschlagen, Ressourcenlimits nur auf Ebene der low-level Ressourcen festzulegen. Das Hauptziel zu verhindern, daß einzelne Anwendungen die Arbeit des Gesamtsystems beeinträchtigen, indem sie zu viele Ressourcen belegen, kann auf diese Art erreicht werden.

Für high-level Ressourcen kann ein Quota-Management in abgeschwächter Form durchgeführt werden, indem entweder kein Limit spezifiziert (Anwendung darf diese Ressource benutzen) oder das Limit auf 0 gesetzt wird (Anwendung darf diese Ressource nicht benutzen).

Ressourcenlimits werden ebenfalls als Teil der Gesamtressource spezifiziert, was in einer Umgebung, in der eine low-level Ressource durch mehrere low-level Ressourcenmanager angeboten wird, zu folgendem Problem führt.

Die low-level Ressource „RAM“ mit einer Gesamtkapazität von 128 MB werde zu je 50% durch die Manager „R1“ bzw. „R2“ angeboten. Die Obergrenze an belegbarem Speicher für Anwendung „X“ sei 64 MB also 50% der Gesamtkapazität. Eine Anfrage von „X“ nach 48 MB RAM wird von „R1“ bzw. „R2“ in eine Forderung nach 75% ihrer Kapazität umgewandelt und überschreitet so das als 50% spezifizierte Maximum. Ein ähnliches Szenario läßt sich z.B. für ein Mehrprozessorsystem konstruieren.

Eine Möglichkeit der Behebung dieses Problems wäre eine Wichtung der Managerangaben entsprechend des von ihnen verwalteten Anteils an der Gesamtressource, eine andere Lösung wäre die Einführung eines Managers, der als Klient o.g. Manager fungiert und deren Vorhandensein vor dem QoS- bzw. Quota-Manager verbirgt.

### 4.6.3 Erweiterungen für den Einsatz in Rechnernetzen

Die Beschränkung dieses Entwurfs auf einen Einzelrechner stellt eine schwerwiegende Einschränkung dar, wenn man den Einsatz des Drops-QoS-Managements im Rahmen des COMQUAD-Projektes betrachtet. Wie in Abschnitt 3.2 dargelegt, wird COMQUAD für den Einsatz in einem Rechnernetz entwickelt. Somit sind Erweiterungen des hier vorgestellten Entwurfs nötig, die den Einsatz in einer verteilten Umgebung erlauben.

An dieser Stelle sollen Anregungen gegeben werden, die für eine solche Erweiterung in Betracht gezogen werden können.

Als erster Schritt wird die Einführung eines globalen Namensdienstes vorgeschlagen. Voraussetzung ist das Vorhandensein von über Rechnergrenzen hinaus eindeutigen Adressen. Diese Voraussetzung ist zur Zeit noch nicht gegeben. Zur Erhaltung der Allgemeingültigkeit des Entwurfs sollten diese Adressen einen transparenten Zugriff auf lokale und entfernte Ressourcen erlauben (Ortstransparenz). Das erfordert weiterhin einen Mechanismus, der diese Adressen interpretiert und Nachrichten an die entsprechenden Knoten des Rechnernetzes sendet.

Die Lokalisierungsmechanismen der im Kapitel *Stand der Technik* vorgestellten Projekte GARA und Rialto unterstützen Ortstransparenz.

Der vorgeschlagene globale Namensdienst ermöglicht es einem zentralen QoS-Manager, entfernte Ressourcen zu verwalten. Durch Ausnutzung der Eigenschaft der Ortstransparenz der Adressen, kann die Kommunikation mit entfernten Ressourcenmanagern ohne Änderungen des QoS-Managers erfolgen. Als nachteilig wird betrachtet, daß Namensdienst und QoS-Manager zentrale Ausfallpunkte des Systems darstellen. Weiterhin verlängert sich eine Ressourcenreservierungsphase umso mehr, je mehr entfernte Ressourcenmanager kontaktiert werden müssen. Außerdem ist es dem QoS-Manager nicht möglich, Lastverteilung durchzuführen oder durch entfernte Kommunikation auftretende Latenzen zu berücksichtigen, die beim Zugriff auf solche Ressourcen auftreten und die Einhaltung des abgeschlossenen QoS-Vertrags gefährden können.

Eine Reduzierung des Nachrichtenaufkommens während der Reservierungsphase kann erreicht werden, indem auf jedem Knoten des Systems ein lokales QoS-Management installiert wird. Ein lokales QoS-Management unterscheidet sich von dem im Entwurf vorgestellten QoS-Management nur dadurch, daß es eine Beschreibung aller auf dem entsprechenden Rechner angebotenen Ressourcen an den zentralen QoS-Manager übermittelt. Der zentrale QoS-Manager faßt Reservierungsanfragen, die an ein und denselben Knoten gerichtet sind, zu einer einzigen Nachricht zusammen und sendet diese als Anfrage an den entsprechenden lokalen QoS-Manager, welcher eine Aushandlung für seinen Knoten durchführt und das gesammelte Ergebnis zurücksendet.

Der zentrale QoS-Manager muß jetzt in der Lage sein, die Lokalität einer Ressource zu ermitteln. Neben der Möglichkeit, Nachrichten zusammenzufassen, wenn sie den selben Knoten betreffen, kann er diese Information in Reservierungsentscheidungen einfließen lassen und z.B. bevorzugt Ressourcen reservieren, die auf dem selben Knoten liegen wie die anfordernde Anwendung.

Durch die Einführung lokaler und globaler QoS-Manager wird die Konstruktion von QoS-Management-Hierarchien ermöglicht.

### 4.6.4 Änderung von QoS-Verträgen

Viele existierende QoS-Managementsysteme (vgl. Kapitel *Stand der Technik* unterstützen die Änderung bestehender QoS-Verträge. Durch diese Möglichkeit sind sie z.B. in Situationen von Ressourcenknappheit in der Lage, flexibel zu reagieren.

Unterstützt das System beispielsweise Anwendungen mit verschiedenen Prioritäten, so ist es bei Ressourcenmangel möglich, schon aktive, niederpriore Anwendungen zum Umschalten in einen ressourcensparenderen Modus aufzufordern und die dadurch freigewordenen Ressourcen anderen, höherprioren Applikationen zuzuordnen, die nun zusätzlich aufgenommen werden können.

Im umgekehrten Falle ist es möglich, Anwendungen von freien Ressourcen in Kenntnis zu setzen, was diesen das Umschalten in einen qualitativ höherwertigen Modus erlaubt.

Andere Szenarien, in denen Änderungen von QoS-Verträgen wünschenswert sind, betreffen z.B. das Ausscheiden von Ressourcenmanagern aus dem System, bzw. die Entdeckung von Fehlersituationen innerhalb der Ressourcenmanager-Schicht des QoS-Managementsystems. Möglich wäre es hier, Dienste auf andere Manager zu migrieren, eventuell sogar transparent für die Anwendung.

Die Verwirklichung dieser Mechanismen erfordert allerdings eine recht hohe Komplexität innerhalb des QoS-Managements, da nun auch Abhängigkeiten zwischen verschiedenen Applikationen betrachtet werden müssen. Weiterhin sind zusätzliche Schnittstellen zwischen QoS-Management und Anwendungen erforderlich.

Bei Interaktionen mit einer Anwendung müssen jetzt auch Situationen betrachtet werden, in denen diese gegen das Protokoll verstößt und z.B. auf Forderungen zur Verminderung ihrer Ressourcenansprüche nicht reagiert.

Aus Gründen der zusätzlichen Komplexität werden Änderungen von QoS-Verträgen in diesem Entwurf nicht zugelassen.

### 4.6.5 Ressourcenpools

Insbesondere bei der Speicherverwaltung ist es unter Umständen wünschenswert, daß sich Manager einen Betrag an tieferliegenden Ressourcen reservieren, um diesen bei Bedarf anfragenden Managern oder Anwendungen zur Verfügung zu stellen. Dieses Verfahren erhöht die Flexibilität des betreffenden Managers im Hinblick auf seine Zuteilungsstrategie. Andererseits wirft die Existenz solcher privater Ressourcenpools im hier betrachteten Umfeld Probleme auf, die nun diskutiert werden sollen.

In Abbildung 4.6 ist die Auflösung von Ressourcenanforderungen dargestellt. Im linken Szenario existieren keine privaten Pools. Somit ist der Betrag der Ressourcen, die durch „RM1“ angefordert werden gleich dem benötigten Betrag (10 Einheiten). Die weitere Abbildung ergibt bei Wahl des „RM2(1)“

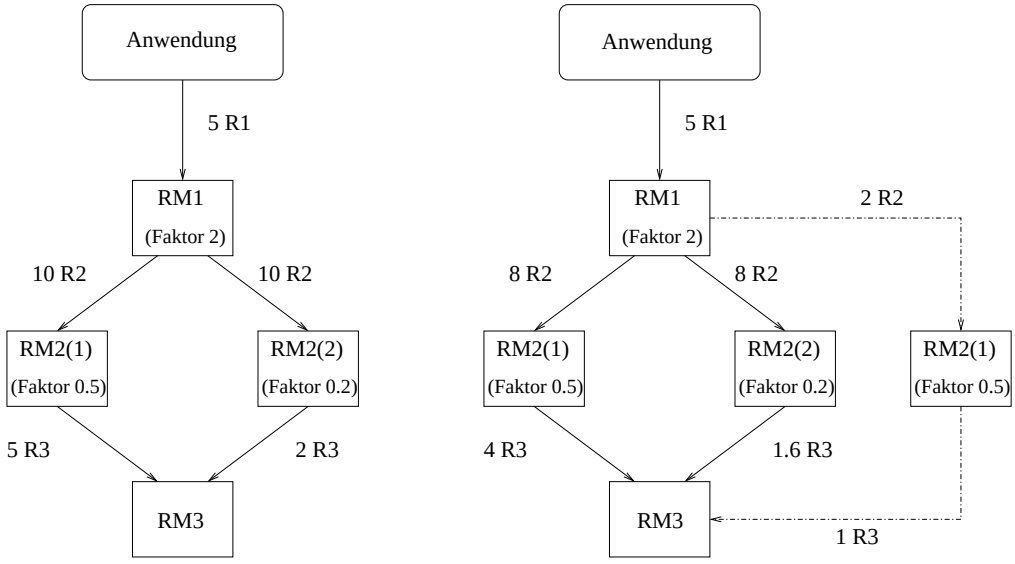


Abbildung 4.6: Auflösung des Ressourcenbedarfs; links ohne Pool, rechts mit Pool; Die Abbildung des Ressourcenbedarfs erfolgt jeweils durch Multiplikation einer Forderung mit „Faktor“.

einen Bedarf von 5 Einheiten der low-level Ressource „R3“ oder bei Wahl des „RM2(2)“ einen Bedarf von 2 Einheiten der Ressource „R3“.

Die rechte Seite der Abbildung zeigt das gleiche Szenario, wobei „RM1“ einen Pool von Ressourcen „R2“ besitzt. Der Betrag an Ressourcen „R2“, der reserviert werden muß, ist hier nur 8 Einheiten, da 2 Einheiten aus dem Pool zugewiesen werden sollen.

Diese 2 Einheiten wurden im Vorfeld reserviert, wobei durch den QoS-Manager der ineffizienter arbeitende „RM2(1)“ ausgewählt wurde. Je nach Wahl der Manager „RM2(1)“ oder „RM2(2)“ ergibt sich hier ein Bedarf von 5 bzw. 2.6 Einheiten der Ressource „R3“.

Als nachteilig erweist sich die erhöhte Komplexität, die durch die Existenz privater Pools auftritt. Bei der Abbildung auf tieferliegende Ressourcen muß zwischen solchen Ressourcen unterschieden werden, die neu reserviert werden müssen und anderen, die aus dem Pool zugewiesen werden sollen. Pools müssen im laufenden Betrieb angepaßt werden. Bei der Reservierung neuer Pool-Einheiten können wiederum verschiedene tieferliegende Manager beteiligt sein.

Der Mechanismus der Überwachung von Ressourcenlimits, wie er oben vorgeschlagen wurde, funktioniert hier nicht, da die Abbildung von Ressourcenforderungen auf Anwendungen nicht ohne weiteres möglich ist. Die Entscheidungsfreiheit des zentralen QoS-Manager in der Ressourcenzuteilung wird durch die Zuweisung der Pool-Ressourcen eingeschränkt. Das kann zur

Folge haben, daß Reservierungsanforderungen fehlschlagen. Wenn das Limit der in der Abbildung dargestellten Anwendung für Ressource „R3“ beispielsweise 2.5 Einheiten beträgt, schlägt die Anfrage im Szenario mit Pool fehl, da wenigstens 2.6 Einheiten benötigt werden, während die Anfrage im Beispiel ohne private Pools nicht fehlschlägt, da hier nur 2 Einheiten der Ressource „R3“ notwendig sind.

Verzichtet man auf diese Pools verringert sich die Komplexität des Reservierungsprozesses beträchtlich. Außerdem bleibt die Abbildung einer Anwendungsforderung auf zu reservierende Ressourcen erhalten. Dafür kann sich aber ein Mehraufwand ergeben, da jede Anfrage nun auf low-level Ressourcen abgebildet werden muß. Weiterhin ist die Granularität der low-level Ressourcen nun ausschlaggebend für die Menge an Ressourcen, die reserviert werden muß. Benötigt eine Anwendung z.B. 1 kB physischen Speicher und die Bereitstellungsgranularität beträgt 4 kB, so müssen diese 4 kB reserviert und dem Ressourcenbetrag der Anwendung angerechnet werden. Das kann zu einer Verschlechterung der Ressourcenausnutzung, bzw. zu einer Ablehnung der Anfrage wegen Ressourcenmangels führen. Wie gezeigt wurde, können ähnliche Anomalien aber auch unter Verwendung von Pools auftreten.

Da private Ressourcenpools eine unverhältnismäßig hohe Komplexität der Reservierungsphase verursachen, werden sie in diesem Entwurf nicht zugelassen.

# Kapitel 5

# Implementierung

In diesem Kapitel wird zunächst gezeigt, wie das im Kapitel *Entwurf* vorgestellte Design der Ressourcenbeschreibung umgesetzt wurde. Danach wird auf den Implementierungsstand von QoS- und Quota-Manager eingegangen. Abschließend wird die im Rahmen der Aufgabenstellung entwickelte Beispielimplementierung vorgestellt.

## 5.1 Umsetzung der Ressourcenbeschreibung

Im Abschnitt 4.4 wurde die Notwendigkeit der generischen Darstellung von Ressourcenbeschreibungen deutlich gemacht und XML als Trägersprache festgelegt. Als Werkzeug zum Parsen der Ressourcenbeschreibungen wurde die Version 1.95.2 der Bibliothek expat [Cla01] verwendet. Bei expat handelt es sich um einen in „C“ geschriebenen, stromorientierten Parser, der die Grundlage vieler anderer XML-Parser bildet und somit praxiserprobt und zuverlässig ist.

Nachfolgend wird das Format der entstandenen XML-Strukturen vorgestellt.

### 5.1.1 `rescoll`

Eine „Resource-Collection“ (`rescoll`) wird von Anwendungen als Parameter des `negotiate()` Aufrufs an den QoS-Manager übergeben und beinhaltet die Beschreibung der benötigten Ressourcen.

High-level Ressourcenmanager übermitteln ihren Ressourcenbedarf als Rückgabeparameter des `resources needed()`- bzw. `pre_reserve()`-Aufrufs ebenfalls in Form einer „Resource-Collection“.

Die DTD einer „Resource-Collection“ ist in Abbildung 5.1 dargestellt. Sie enthält eine oder mehrere Alternativen (`alternative`). Innerhalb einer Alternative werden alle nötigen Ressourcen mit ihren konkreten Attributen beschrieben. Verschiedene Alternativen enthalten üblicherweise unterschiedliche Ressourcenforderungen. So können Forderungen modelliert werden, die

jeweils den Betrieb der Anwendung in einem anderen Modus erlauben. Jeder Alternative wird als Attribut eine natürliche Zahl als Identifikator zugewiesen.

Nach erfolgreicher Reservierung einer Alternative durch den QoS-Manager ist deren Identifikator Bestandteil des QoS-Vertrages (`handlestring`). So wird für die Anwendung ersichtlich, für welche Alternative Ressourcen reserviert wurden.

Die Beschreibung einer Ressourcenforderung (`resource`) gliedert sich in einen generischen Teil (`spec`) und einen ressourcenabhängigen Teil (`attributes`). Der generische Teil wird durch den QoS-Manager ausgewertet, um die Abbildung auf Ressourcenmanager durchzuführen. Eine Ressource wird durch ihren Namen (`rname`) und optional durch weitere Eigenschaften (`property`) beschrieben.

Der ressourcenabhängige Abschnitt einer Beschreibung wird nur von einem entsprechenden Ressourcenmanager ausgewertet und enthält konkrete Forderungen als Name-Wert-Paare (`aname`, `avalue`).

```
<!ELEMENT rescoll (alternative)+ >

<!ELEMENT alternative (resource)+>
<!ATTLIST alternative altnr CDATA #REQUIRED>

<!ELEMENT resource (spec, attributes)>

<!ELEMENT spec (rname,property*)>
<!ELEMENT rname (#PCDATA)>
<!ELEMENT property (#PCDATA)>

<!ELEMENT attributes (aname,avalue)+>
<!ELEMENT aname (#PCDATA)>
<!ELEMENT avalue (#PCDATA)>
```

Abbildung 5.1: DTD einer „Resource-Collection“ (`rescoll`)

### 5.1.2 `resource`

Nach Abbildung einer Ressourcenforderung auf einen konkreten Ressourcenmanager wird diesem die entsprechende Anforderung (`resource` - vgl. 5.1.1) als Parameter eines `resources_needed()`- bzw. `pre_reserve()`-Aufrufs zur weiteren Auswertung übergeben.

### 5.1.3 `spec`

Ressourcenmanager, die in das System aufgenommen werden sollen, müssen sich durch Aufruf der `register()`-Funktion beim QoS-Manager anmelden und den Dienst, den sie anbieten, beschreiben. Die Beschreibung erfolgt durch eine Ressourcenspezifikation (`spec` - vgl. 5.1.1).

Die Implementierung des Quota-Managers arbeitet ebenfalls mit einer Beschreibung der Ressourcen im `spec`-Format.

### 5.1.4 `handlestring`

Nach erfolgter Ressourcenreservierung faßt der QoS-Manager die von den entsprechenden Ressourcenmanagern zurückgegebenen „Handles“ auf die reservierten Ressourcen (siehe Abschnitt 4.3.3) als QoS-Vertrag in einer XML-Struktur (`handlestring`) zusammen und übergibt diesen als Parameter eines `reserve()`-Aufrufs an den fordernden Ressourcenmanager weiter, bzw. als Rückgabewert eines `negotiate()`-Aufrufs an die aufrufende Anwendung zurück.

Ein `handlestring` (siehe Abb. 5.2) enthält zunächst den Identifikator für die reservierte Alternative (`resalt`) (siehe Abschnitt 5.1.1) gefolgt von der Handles-Sektion (`handles`). Für jeden an der Reservierung beteiligten Manager (`mgr`) werden die für den Zugriff notwendigen Informationen aufgeführt. `mgrid` enthält die Thread-ID des Managers, `resnr` die QoS-Reservierungsnummer der Anfrage.

Da Binärdaten nicht ohne weiteres innerhalb einer XML-Struktur übertragen werden können, erfolgt die Übermittlung von `mgrid` und `resnr` als Base64[1] kodierte Zeichenkette.

```
<!ELEMENT handlestring (resalt, handles)>

<!ELEMENT resalt (#PCDATA)>

<!ELEMENT handles (mgr)+>

<!ELEMENT mgr (spec, mgrid, resnr)>
<!ELEMENT mgrid (#PCDATA)>
<!ELEMENT resnr (#PCDATA)>
```

Abbildung 5.2: DTD eines QoS-Vertrags (`handlestring`) ; vgl. Abschnitt 5.1 für die Definition von `spec`

### 5.1.5 `resinfo`

Informationen über zu belegende Ressourcen bilden die Grundlage für die Bewertung von Ressourcenkombinationen (siehe Abschnitt 4.3.3).

Low-level Ressourcenmanager übermitteln diese Information als „Resource-Information“ (`resinfo`). Übertragen wird der Anteil des angeforderten Ressourcenbetrags (`requested`) und der Anteil des noch nicht belegten Ressourcenbetrags (`available`) am Gesamtbetrag der Ressource (siehe Abb. 5.3).

[1] Dieses Verfahren wandelt beliebige Bitfolgen in ein Format um, das keine durch XML definierten Steuerzeichen enthält (vgl. RFC 1113).

```
<!ELEMENT resinfo (requested,available)>

<!ELEMENT requested (#PCDATA)>
<!ELEMENT available (#PCDATA)>
```

Abbildung 5.3: DTD einer „Resource-Information“ (`resinfo`)

## 5.2 QoS-Manager

Im Rahmen der hier vorgestellten Arbeit wurde der im Kapitel Entwurf spezifizierte QoS-Manager in „C“ implementiert. Er enthält die zur Durchführung der Abbildung von Ressourcen auf Ressourcenmanager notwendige Funktionalität (Namensdienst) sowie Mechanismen, die Reservierung und Freigabe von Ressourcen im Namen einer Anwendung unterstützen. Es entstanden Bibliotheken für die Kapselung des Kommunikationscodes und Bibliotheken, die u. a. Funktionen zur Erleichterung der Auswertung und Erstellung von XML-Dokumenten enthalten. Abbildung 5.4 zeigt einen Überblick.

In den folgenden Abschnitten werden Teilaspekte des QoS-Managers näher betrachtet.

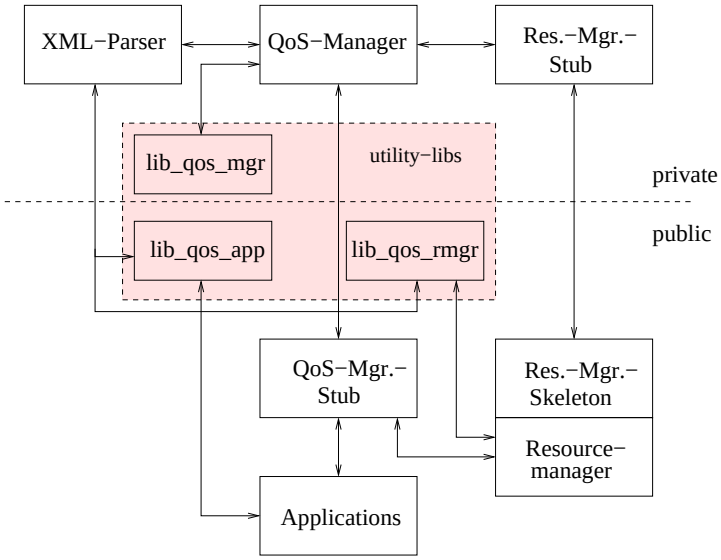


Abbildung 5.4: QoS-Manager und Bibliotheken; die Pfeile symbolisieren Kommunikationsbeziehungen.

### 5.2.1 Ermittlung von Ressourcenmanager-Kombinationen

Der Algorithmus zur Auflösung der Anforderungen wurde so gewählt, daß Alternativen unabhängig voneinander ermittelt werden und somit der

Auflösungsprozeß beliebig abgebrochen werden kann (siehe Abschnitt 4.3.3). Seine Funktionsweise soll nun anhand des in Abbildung 5.5 gezeigten Szenarios beschrieben werden.

Das Verfahren benutzt zwei Stacks. Auf einem, dem „Pflicht-Stack“, werden Informationen abgelegt, die für die erfolgreiche Auflösung einer Anfrage noch ausgewertet müssen. Auf dem anderen Stack, dem „Alternativ-Stack“, werden Informationen abgelegt, die zur Konstruktion alternativer Kombinationen benötigt werden.

Einträge des „Pflicht-Stack“ werden ausgewertet und in einen Baum eingefügt, der die Hierarchie der Nutzungsabhängigkeiten zwischen Ressourcenmanagern repräsentiert.

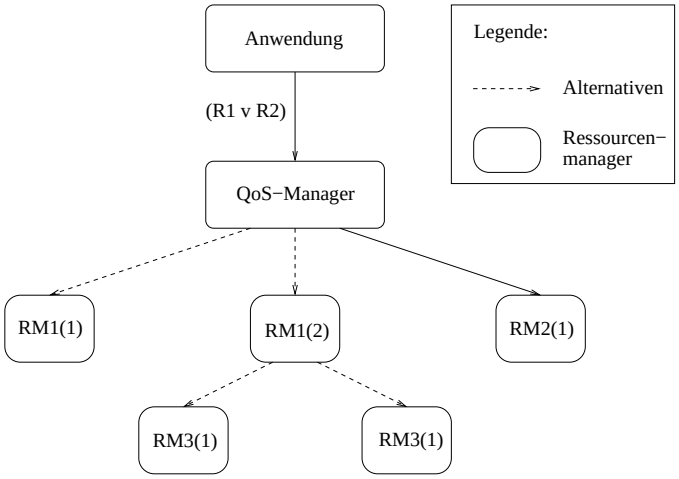


Abbildung 5.5: mögliche Ressourcenmanager-Kombinationen (Beispielszenario)

Die Anwendung in Abbildung 5.5 möchte entweder Ressource „R1“ oder Ressource „R2“ reservieren und übergibt diese Anfrage an den QoS-Manager. Zu diesem Zeitpunkt sind alle Stacks leer. Die Anfrage wird zuerst in ihre Alternativen zerlegt und diese auf dem „Alternativ-Stack“ abgelegt. Der oberste Eintrag des „Alternativ-Stack“ wird in die Ressourcen zerlegt, die er beinhaltet und diese auf dem „Pflicht-Stack“ abgelegt. Im betrachteten Szenario enthält er nun das Element „R1“. Zur Erfüllung der Anwendungsforderung muß für jede Ressource des „Pflicht-Stack“ eine Reservierung erfolgen.

Solange der „Pflicht-Stack“ also Elemente enthält, werden diese ausgewertet.

Das oberste Element des „Pflicht-Stack“ ist eine Ressource, deshalb wird eine Abbildung auf Ressourcenmanager vorgenommen. Diese liefert zwei Alternativen, „RM1(1)“ und „RM1(2)“. „RM1(1)“ wird als erstes Element in einen neuen Ressourcenbaum („B1“) eingefügt, „RM1(2)“ wird mit einem Vermerk, daß es „RM1(1)“ gleichgeordnet ist, auf dem „Alternativ-Stack“

abgelegt. Eine Anfrage bei „RM1(1)“ ergibt, daß keine weiteren Ressourcen benötigt werden.

Der „Pflicht-Stack“ ist nun leer. Das bedeutet, daß eine Anforderung erfolgreich aufgelöst wurde.

An dieser Stelle kann entweder abgebrochen, oder alternative Kombinationen gesucht werden.

Zur Ermittlung der nächsten Kombination wird wieder das oberste Element des „Alternativ-Stack“ („RM1(2)“) ausgewertet. Ein neuer Baum („B2“) wird erzeugt und „RM1(2)“ eingefügt. Eine Anfrage bei „RM1(2)“ ergibt, daß weitere Ressourcen benötigt werden („R3“). Diese als „Resource-Collection“ übergebene Ressourcenanfrage wird in ihre Alternativen zerlegt und diese auf dem „Alternativ-Stack“ abgelegt. Der oberste Eintrag des „Alternativ-Stack“ wird (wie oben gezeigt) ausgewertet, seine Ressourcen („R3“) auf dem „Pflicht-Stack“ abgelegt und für die oberste der Ressourcen eine Auflösung vollzogen, die in diesem Fall zwei in Frage kommende Manager, „RM3(1)“ und „RM3(2)“, liefert. „RM3(1)“ wird unterhalb von „RM1(2)“ in „B2“ eingefügt und „RM3(2)“ mit einem Verweis auf „RM3(1)“ auf dem „Alternativ-Stack“ abgelegt. „RM3(1)“ benötigt keine weiteren Ressourcen, damit ist der „Pflicht-Stack“ leer und ein weiterer Baum fertiggestellt.

Als oberster Eintrag dieses Baumes besitzt „RM1(2)“ einen Verweis auf ein Element in einem vorher konstruierten Baum. Bäume mit diesem Element (im konkreten Fall „B1“) werden kopiert. In der Kopie wird der Teilbaum mit Element „RM1(1)“ als Wurzel durch den neu konstruierten Baum mit „RM1(2)“ als Wurzel ersetzt. Somit wurde eine zweite Kombination gefunden.

Auf diese Weise kann fortgefahren werden, bis beide Stacks leer sind. Tritt dieser Fall ein, wurden alle Kombinationen ermittelt. Abbildung 5.6 stellt den Ablauf graphisch dar.

Einträge in Bäumen bestehen aus Referenzen auf Objekte. Kopieraktionen beschränken sich damit auf das Kopieren von Referenzen. Objekte werden nach dem Löschen der letzten Referenz zerstört.

### 5.2.2 Bewertung von Ressourcenkombinationen

Im Abschnitt 4.3.3 wurden zwei Ansätze für den Zeitpunkt der Bewertung alternativer Ressourcenkombinationen vorgestellt. Die Implementierung des QoS-Managers unterstützt beide Ansätze. Durch entsprechendes Setzen eines Parameters kann die gewünschte Variante zur Übersetzungszeit ausgewählt werden.

Damit eine Bewertung vor der Vorreservierungsphase erfolgen kann, müssen low-level Ressourcenmanager auch im Rückgabeparameter des `resources_needed()`-Aufrufs eine gültige `resinfo`-Struktur liefern.

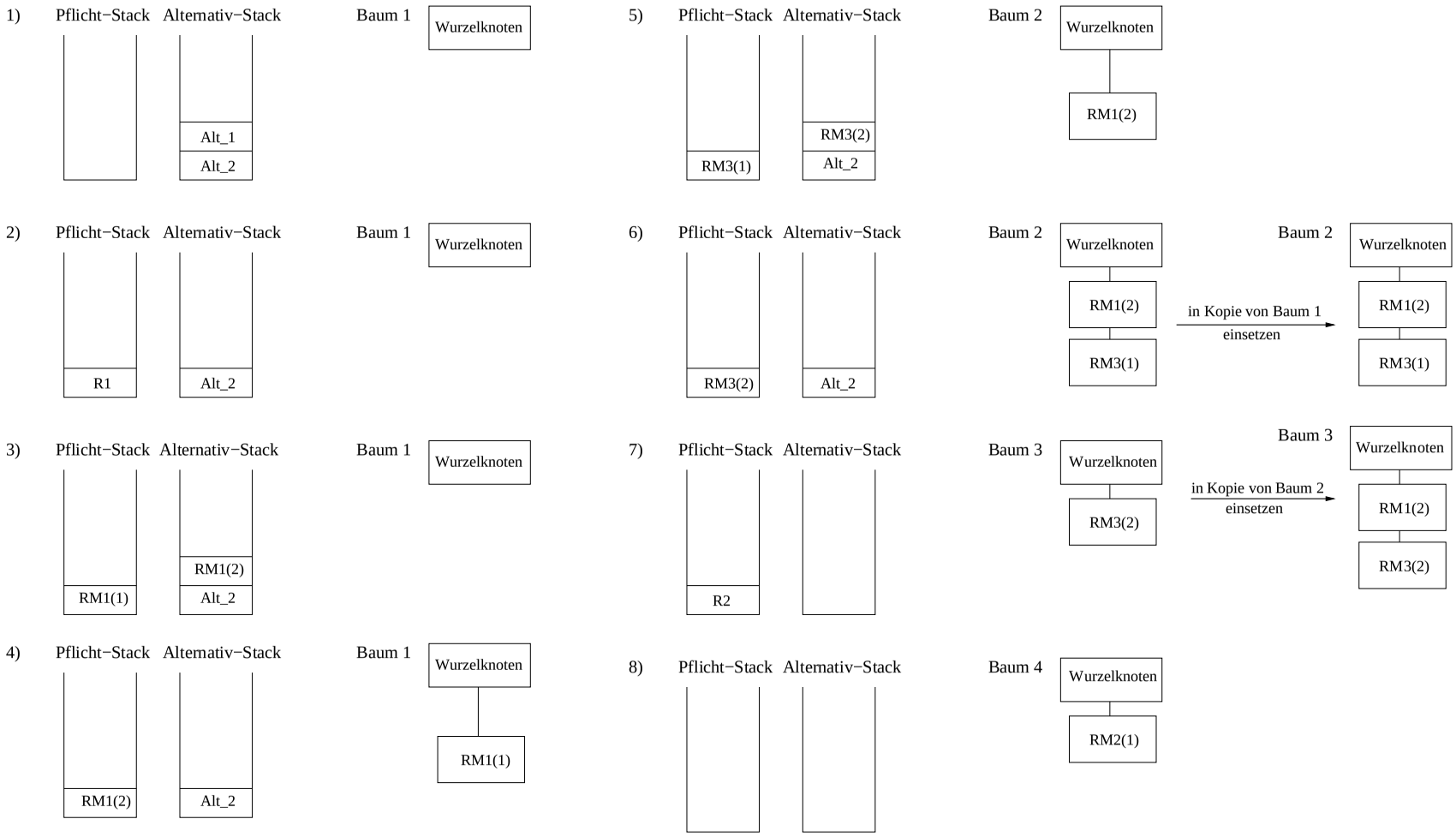


Abbildung 5.6: Konstruktion von Alternativen

Für die Durchführung von Bewertungen wurden drei einfache Strategien implementiert. Dabei wurde die Tatsache berücksichtigt, daß der QoS-Manager alternative Forderungen einer Anwendung in der Reihenfolge ihres Auftretens innerhalb der Anfragestruktur bearbeitet.

Diese Strategien werden nun kurz erklärt.

**1. Fifo:** Kombinationen werden in der Reihenfolge ausgewählt, in der sie erzeugt wurden. Durch die oben genannten Eigenschaften des Algorithmus zur Ressourcenauflösung können Anwendungen durch Plazierung bevorzugter Alternativen an den Anfang der Ressourcenforderung die Wahrscheinlichkeit erhöhen, daß Ressourcen für diese Alternativen reserviert werden.

**2. Ressourcenbedarf:** Diese Methode wertet die Information aus, die low-level Ressourcenmanager im `requested`-Feld ihrer `resinfo`-Struktur übergeben (siehe Abschnitt 5.1.5). Die „günstigste“ Kombination ist diejenige mit dem geringsten Bedarf an low-level Ressourcen.

**3. Zufällig:** Hier werden Kombination nach dem Zufallsprinzip ausgewählt.

Die gewünschte Strategie kann beim Start des QoS-Managers durch Angabe eines Kommandozeilenparameters ausgewählt werden (siehe Tabelle 5.1).

| Parameter | Bedeutung |
|---|---|
| -m 0 | Auswahl gemäß Fifo (Standard) |
| -m 1 | Auswahl nach Ressourcenbedarf |
| -m 2 | Auswahl nach Zufallsprinzip |

Tabelle 5.1: Kommandozeilenparameter des QoS-Managers

## 5.3 Quota-Manager

Die Implementierung des Quota-Managers bietet eine eingeschränkte Funktionalität an. Für die Verwaltung von Ressourcenlimits wurde daß in UNIX-Systemen für Festplattenplatz angewandte Quota-Prinzip eingesetzt (siehe Abschnitt 3.4.3). Limits werden auf der Ebene von L4-Tasks verwaltet. Das bedeutet, daß mit einer Portierung auf die Version 4 der L4-Spezifikation eine Anpassung erfolgen muß.

Derzeit ist es nicht möglich, Ressourcenlimits zu spezifizieren, die für ausgewählte Anwendungen gelten. Wird für eine Ressource ein Limit festgelegt, so gilt dieses Limit für jede Anwendung. Ist für eine Ressource kein Limit spezifiziert, so wird angenommen, daß keine Begrenzung für diese Ressource erfolgen soll.

Die Initialisierung des Quota-Managers mit zu überwachenden Ressourcen erfolgt beim Start des Managers.

Für die Auswertung von Ressourcenbeschreibungen verwendet der Quota-Manager die im Rahmen der Implementierung des QoS-Managers entstandene `lib_qos_rmgr`-Bibliothek.

Abbildung 5.7 stellt die Beziehungen des Quota-Managers zu anderen Komponenten des QoS-Managementsystems dar.

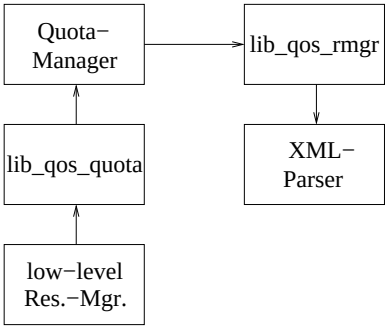


Abbildung 5.7: Beziehungen des Quota-Managers zu anderen Komponenten der Ressourcenverwaltung

## 5.4 Implementierung der Beispielumgebung

Die Funktionsfähigkeit von QoS- und Quota-Manager sollte im Rahmen der Implementierung einer Beispielumgebung gezeigt werden, die die Wiedergabe eines MPEG-Videostroms in Echtzeit ermöglicht.

Dazu war es notwendig, verschiedene Ressourcenmanager so anzupassen, daß sie die im Abschnitt 4.5.3 angegebenen Anforderungen erfüllen. Da die Änderungen an den betroffenen Ressourcenmanagern recht umfangreich waren, wurden zunächst nur zwei beteiligte Manager angepaßt bzw. neu entwickelt. Diese arbeiten mit anderen DROPS-Komponenten zusammen, welche die Kriterien der QoS-Umgebung noch nicht erfüllen.

Desweiteren wurden 2 Anwendungen implementiert, die die Mechanismen des QoS-Managers für die Anforderung und Freigabe von Ressourcen nutzen (siehe Abb. 5.8).

Die Komponenten der QoS-Umgebung werden nun vorgestellt.

### 5.4.1 QoS-Speichermanager

Der Speichermanager der QoS-Umgebung heißt `dm_phys_qos` und stellt physischen Arbeitsspeicher zur Verfügung. Als low-level Ressourcenmanager der QoS-Umgebung implementiert er neben der im Abschnitt 4.5.3 geforderten Funktionalität auch die im Punkt 5.2.2 spezifizierte Erweiterung.

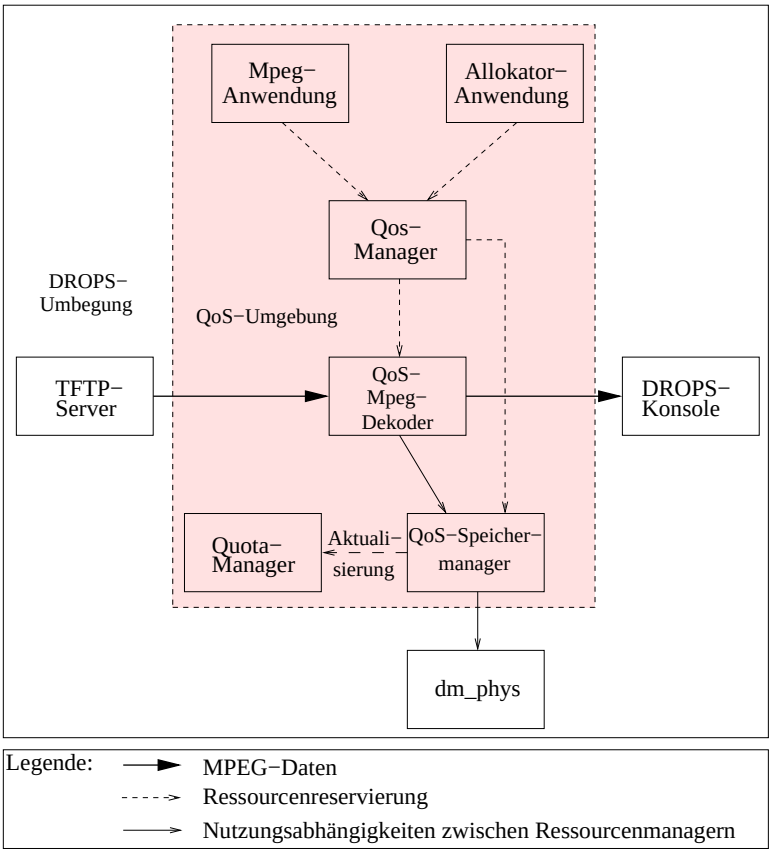


Abbildung 5.8: Komponenten der Beispielumgebung

```
<?xml version="1.0" standalone="no"?>
 <!DOCTYPE spec SYSTEM "qos_xml.dtd">
<spec>
  <rname>mem_qos</rname>
  <property>pinned</property>
</spec>
```

Abbildung 5.9: Ressourcenbeschreibung des QoS-Speichermanagers

```
<?xml version="1.0" standalone="no"?>
 <!DOCTYPE resinfo SYSTEM "qos_xml.dtd">
<resinfo>
  <requested>100</requested>
  <available>1000</available>
</resinfo>
```

Abbildung 5.10: Beispiel einer `resinfo`-Struktur des QoS-Speichermanagers; die Forderung entsprach 100 Promille der Gesamtressource bei einer noch freien Kapazität von 1000 Promille

Er bezieht seinen Speicher vom Speichermanager der DROPS-Umgebung (z.Z. `dm_phys`).

`dm_phys` unterstützt Speicherpools. Speicherpools verwalten jeweils exklusiv einen bestimmten Bereich des physischen Speichers. Die Größe von Pools kann beim Start von `dm_phys` spezifiziert werden. Für die hier beschriebene QoS-Umgebung wurde ein 20 MB großer Pool definiert, der ausschließlich von `dm_phys_qos` benutzt wird.

`dm_phys` bietet „pinned“ Speicher an, wodurch sichergestellt ist, daß beim Zugriff auf erhaltenen Speicher keine Seitenfehler auftreten. Die Granularität des Speichers entspricht der Seitengröße (4 kB).

Der durch `dm_phys_qos` zugewiesene Speicher besitzt die gleichen Eigenschaften wie der von `dm_phys` vergebene.

`dm_phys_qos` erwartet während der Ressourcenaushandlung den gewünschten Speicherbetrag in Byte. Dieser Betrag wird, wenn nötig, entsprechend der Granularität (4 kB) aufgerundet. Die Vergabe erfolgt in Absprache mit dem Quota-Manager.

Klienten von `dm_phys_qos` können die Bibliothek `lib_qos_malloc` für eine einfachere Handhabung ihres Speichers verwenden. Diese Bibliothek bietet eine `malloc()` / `free()`-ähnliche Schnittstelle an. Die Verwaltung des Speichers erfolgt mit Hilfe der „lmm“-Implementierung des OS-Kits [The99].

Abbildung 5.9 zeigt die Ressourcenbeschreibung, mit der sich `dm_phys_qos` beim QoS-Manager anmeldet. Abbildung 5.10 zeigt eine mögliche `resinfo`-Struktur, die `dm_phys_qos` an den QoS-Manager zurückgibt.

### 5.4.2 QoS-MPEG-Dekoder

Der MPEG-Dekoder der QoS-Umgebung implementiert die im Abschnitt 4.5.3 geforderte Funktionalität.

Die Dekodierung des Videostroms wird durch eine angepaßte Version der am Lehrstuhl Betriebssysteme entwickelten Smart-Mpeg-Bibliothek durchgeführt. Die Anpassung bestand darin, daß Aufrufe zur Allokation bzw. Freigabe dynamischen Speichers durch Aufrufe der entsprechenden Funktionen der Bibliothek `lib_qos_malloc` ersetzt wurden, um den Zugriff auf den von `dm_phys_qos` bereitgestellten Speicher zu ermöglichen.

Die Videodaten werden durch einen DROPS-TFTP-Server bereitgestellt, die Ausgabe erfolgt über die DROPS-Konsole.

Während der Ressourcenreservierungsphase erhält der MPEG-Dekoder Zugriff auf den zur Realisierung der Forderung benötigten Speicher. Dieser wird durch den QoS-Speichermanager zur Verfügung gestellt. Der QoS-MPEG-Dekoder erzeugt einen neuen Arbeits-Thread, der so initialisiert wird, daß er die geforderte Dekodierung vornehmen kann und auf dem zur Verfügung gestellten Speicherbetrag arbeitet. Das dem QoS-Manager zurückgegebene Ressourcen-Handle entspricht der Thread-ID des erzeugten Arbeits-Threads.

Da ein solcher Arbeits-Thread für jede zu reservierende Anfrage erzeugt wird, ist die parallele Dekodierung mehrerer Videoströme möglich.

Der zum Dekodieren eines bestimmten Videostroms benötigte Speicherbetrag wurde im Vorfeld ausgemessen und zusammen mit dem betreffenden Video abgelegt.

Der Videodekoder erwartet im Rahmen der Ressourcenaushandlung den Namen des gewünschten Videos als Parameter. Abbildung 5.11 zeigt die Ressourcenbeschreibung, mit der sich der MPEG-Dekoder beim QoS-Manager anmeldet.

```
<?xml version="1.0" standalone="no"?>
 <!DOCTYPE spec SYSTEM "qos_xml.dtd">
<spec>
  <rname>mpeg_qos</rname>
</spec>
```

Abbildung 5.11: Ressourcenbeschreibung des QoS-Videodekoders

### 5.4.3 MPEG-Anwendung

Die MPEG-Anwendung reserviert die Ressource „mpeg_qos“ unter Angabe des gewünschten Videostroms mit Hilfe des QoS-Managers. Nach erfolgreicher Reservierung startet sie nach Auswertung des QoS-Vertrages die Videowiedergabe durch Aufruf der entsprechenden Funktion des Videodekoders. Nach Beendigung der Wiedergabe gibt sie die belegten Ressourcen unter Benutzung der `release()`-Funktion des QoS-Managers wieder frei. Abbildung 5.12 zeigt das Beispiel einer Ressourcenforderung, die die MPEG-Anwendung an den QoS-Manager übergibt. In Abbildung 5.13 wird ein möglicher QoS-Vertrag dargestellt, den sie nach einer erfolgreicher Reservierung erhält.

### 5.4.4 Allokator-Anwendung

Diese Anwendung der QoS-Umgebung reserviert unter Zuhilfenahme des QoS-Managers einen durch den Benutzer angebbaren Betrag an Speicher und gibt ihn auf Anweisung des Nutzers wieder frei. So kann eine „Systemlast“ erzeugt werden, die es erlaubt, das Verhalten der anderen Komponenten in Situationen der Ressourcenknappheit zu testen.

Abbildung 5.14 zeigt das Beispiel einer Ressourcenforderung, die die Allokator-Anwendung an den QoS-Manager übergibt.

```
<?xml version="1.0" standalone="no"?>
 <!DOCTYPE rescoll SYSTEM "qos_xml.dtd">
<rescoll>
  <alternative altnr="1">
    <resource>
      <spec>
        <rname>mpeg_qos</rname>
      </spec>
      <attributes>
        <aname>videoname</aname>
        <avalue>python_352</avalue>
      </attributes>
    </resource>
  </alternative>

  <alternative altnr="2">
    <resource>
      <spec>
        <rname>mpeg_qos</rname>
      </spec>
      <attributes>
        <aname>videoname</aname>
        <avalue>starwars</avalue>
      </attributes>
    </resource>
  </alternative>
</rescoll>
```

Abbildung 5.12: Ressourcenforderung der QoS-MPEG-Anwendung

```
<?xml version="1.0" standalone="no"?>
 <!DOCTYPE handlestring SYSTEM "qos_xml.dtd">
<handlestring>
  <resalt>1</resalt>
  <handles>
    <mgr>
      <spec>
        <rname>mpeg_qos</rname>
      </spec>
      <mgrid> b64-codierter l4_threadid_t </mgrid>
      <resnr> b64-codierter qos_res_nr_t </resnr>
    </mgr>
  </handles>
</handlestring>
```

Abbildung 5.13: QoS-Vertrag der QoS-MPEG-Anwendung

```
<?xml version="1.0" standalone="no"?>
 <!DOCTYPE rescoll SYSTEM "qos_xml.dtd">
<rescoll>
  <alternative altnr="1">
    <resource>
      <spec>
        <rname>mem_qos</rname>
        <property>pinned</property>
      </spec>
      <attributes>
        <aname>size</aname>
        <avalue>500</avalue>
      </attributes>
    </resource>
  </alternative>
</rescoll>
```

Abbildung 5.14: Ressourcenforderung der Allokator-Anwendung

# Kapitel 6

# Zusammenfassung und Ausblick

In diesem Kapitel werden die Ergebnisse dieser Arbeit zusammengefaßt. Anschließend werden Anregungen für die weitere Entwicklung des DROPS-QoS-Managementsystems gegeben.

## 6.1 Zusammenfassung

Im Rahmen dieser Arbeit wurden Komponenten entworfen, die Anwendungen bei der Reservierung und Freigabe von Ressourcen unterstützen (QoS-Manager) sowie die Festlegung und Durchsetzung von Ressourcenlimits für Anwendungen ermöglichen (Quota-Manager).

Für die Kommunikation zwischen Komponenten des DROPS-QoS-Managementsystems während der Ressourcenreservierungsphase wurden einheitliche Schnittstellen festgelegt, die die einfache Erweiterung des Systems um neue Ressourcen erlauben.

Für die Formulierung von Ressourcenbeschreibungen und Ressourcenforderungen wurde ein generisches Format entworfen, welches es dem QoS-Manager einerseits ermöglicht, beliebige Ressourcen auf geeignete Ressourcenmanager des Systems abzubilden und andererseits die Übermittlung ressourcenspezifischer Attribute erlaubt.

In den Entwurf flossen zahlreiche Anregungen aus anderen existierenden Projekten ein, die im Kapitel *Stand der Technik* vorgestellt wurden.

Die Ideen des Entwurfs wurde im Rahmen einer Beispielimplementierung umgesetzt. Damit durchgeführte Tests zeigten, daß es dieser Ansatz ermöglicht, die angestrebten Ziele zu erreichen.

## 6.2 Ausblick

Erste Tests mit der Implementierung des vorgestellten Systems haben dessen Funktionsfähigkeit demonstriert, aber auch seine Grenzen aufgezeigt.

Der QoS-Manager verfügt über keine Mechanismen für eine Interaktion mit Anwendungen. Solche Interaktionen sind z.B. dann nötig, wenn Änderungen von QoS-Verträgen durchgeführt werden sollen. Desweiteren werden Abhängigkeiten zwischen Ressourcenmanagern, die durch das Operieren dieser Manager auf demselben Datenstrom auftreten können, nicht berücksichtigt.

Derzeit gibt es in der DROPS-Umgebung keine Instanz, die Privilegien an Komponenten vergibt. Das ist ein Grund dafür, daß der Quota-Manager derzeit keine Abstufung zwischen verschiedenen Applikationen vornimmt und damit jede Anwendung das selbe Ressourcenlimit besitzt.

Ebensowenig können Komponenten, die mit dem QoS-Manager kommunizieren auf ihre Rechte geprüft werden, wodurch sich ein Problem für die Systemsicherheit ergibt, da sich z.B. eine beliebige Komponente als Ressourcenmanager anmelden kann.

Weiterführende Untersuchungen sollten sich mit der Lösung dieser Probleme beschäftigen.

Eine weitere Aufgabe besteht darin, die Funktionalität der Ressourcenmanager der Testumgebung zu erweitern, bzw. weitere DROPS-Ressourcenmanager anzupassen, um die Testumgebung zu vergrößern. Im Mittelpunkt sollte dabei die Frage der Berechnung des benötigten Ressourcenbedarfs stehen.

Im Hinblick auf den Einsatz dieser Umgebung im Rahmen des COMQUAD-Projektes müssen Änderungen vorgenommen werden, die eine Verwaltung verteilter Ressourcen in Rechnernetzen erlauben.

# Literaturverzeichnis


[AH99] Au, Alan ; Heiser, Gernot: *L4 User Manual, Version 1.14*. The University of New South Wales, Sydney 2052, Australia: School of Computer Science and Engineering, März 1999

[BBH+98] Baumgartl, Robert ; Borriss, Martin ; Härtig, Hermann ; Hamann, Claude-Joachim ; Hohmuth, Michael ; Reuther, Lars ; Schönberg, Sebastian ; Wolter, Jean: Dresden Real-time Operating System. In: *First Workshop on System Design Automation (SDA'98)*. TU Dresden, Fakultät Informatik, D-01062 Dresden, März 1998, S. 205–212

[BH99] Borriss, Martin ; Härtig, Hermann: Design and Implementation of a Real-Time ATM-Based Protocol Server / Dresden University of Technology, Operating Systems Group. 1999. – Forschungsbericht

[CCF+00] Chandra, Prashant ; Chu, Yang-hua ; Fisher, Allan ; Gao, Jun ; Corey, Kosak ; NG, T. S. E. ; Steenkiste, Peter ; Takahashi, Eduardo ; Zhang, Hui: Darwin: Customizable Resource Management for Value-Added Network Services / Carnegie Mellon University. 2000. – Forschungsbericht

[CE98] Czajkowski, Grzegorz ; von Eicken, Thorsten: JRes: A Resource Accounting Interface for Java, 1998, S. 21–35

[CFK+98] Czajkowski, Karl ; Foster, Ian ; Karonis, Nick ; Kesselman, Carl ; Martin, Stuart ; Smith, Waren ; Tuecke, Steven: A Resource Management Architecture for Metacomputing Systems. In: *The 4th Workshop on Job Scheduling Strategies for Parallel Processing*, 1998

[Cla01] Clark, James: *Expat, Release 1.95.2*. `http://expat.sourceforge.net`, Juli 2001

[FKL+99] Foster, I. ; Kesselman, C. ; Lee, C. ; Lindell, R. ; Nahrstedt, K. ; Roy, A.: A Distributed Resource Management Architecture that Supports Advance Reservations and Co-Allocation. In: *Proceedings of the International Workshop on Quality of Service*, 1999

[Gos91] Goscinski, Andrzej: *Distributed Operating Systems – The Logical Design*. Addison-Wesley Publishing Company, 1991

[HLR98] Heiser, G. ; Lam, F. ; Russel, S. *Resource Management in the Mungi Single-Address-Space Operating System*. 1998

[HLR+01] Hamann, Claude-Joachim ; Löser, Jork ; Reuther, Lars ; Schönberg, Sebastian ; Wolter, Jean ; Härtig, Hermann: Quality-Assuring Scheduling – Using Stochastic Behavior to Improve Resource Utilization / Dresden University of Technology, Department of Computer Science. 2001. – Forschungsbericht

[HRW+99] Härtig, H. ; Reuther, L. ; Wolter, J. ; Boriss, M. ; Paul, T.: Cooperating Resource Managers / Dresden University of Technology, Department of Computer Science. 1999. – Forschungsbericht

[JLD95] Jones, Michael B. ; Leach, Paul J. ; Draves, Richard: Support for User-Centric Modular Real-Time Resource Management in the Rialto Operating System. In: *Network and Operating System Support for Digital Audio and Video*, 1995, S. 53–63

[JR99] Jones, Michael B. ; Regehr, John: CPU Reservations and Time Constraints: Implementation Experience on Windows NT. In: *Proceedings of the 3rd USENIX Windows NT Symposium*, 1999, S. 93–102

[LHR01] Löser, J. ; Härtig, H. ; Reuther, L.: A Streaming Interface for Real-Time Interprocess Communication / Dresden University of Technology, Department of Computer Science. 2001. – Forschungsbericht

[Liu00] Liu, Jane W.: *Real-time systems*. Prentice-Hall, Inc., 2000

[SHS99] Sullivan, David G. ; Haas, Robert ; Seltzer, Margo I.: Tickets and Currencies Revisited: Extensions to Multi-Resource Lottery Scheduling. In: *Workshop on Hot Topics in Operating Systems*, 1999, S. 148–152

[The99] The Flux Research Group: *Flux Operating System Toolkit Version 0.97*. Department of Computer Science, University of Utah : `http://www.cs.utah.edu/projects/flux/`, Januar 1999

[WW94] Waldspurger, Carl A. ; Weihl, William E.: Lottery Scheduling: Flexible Proportional-Share Resource Management. In: *Operating Systems Design and Implementation*, 1994, S. 1–11