in: Proc. GI/ITG-Fachtagung "Kommunikation in verteilten Systemen", Mannheim, Feb. 1991, S. 216-231

# Ein Client/Server-System als Basiskomponente für ein kooperierendes Datenbanksystem

*Christoph Hübel, Wolfgang Käfer, Bernd Sutter*
*Universität Kaiserslautern*
*Fachbereich Informatik*

**Überblick**

Konventionelle Datenbanksysteme besitzen erhebliche Leistungsschwächen bei der Abarbeitung aufwendiger Verarbeitungsalgorithmen auf komplexen Datenstrukturen. Ein Lösungsansatz wird in der "semantischen Dekomposition", also der Zerlegung einer einzelnen komplexen Datenbankoperation in logisch unabhängige Teilschritte, gesehen. Unter Berücksichtigung der Operationssemantik können die unabhängigen Teilschritte parallel abgewickelt werden, wodurch insgesamt eine Reduktion der Antwortzeit erwartet wird. Die praktische Erprobung und Validierung dieses Ansatzes erfordert die Entwicklung eines verteilten, kooperierenden Datenbanksystems.

Im vorliegenden Beitrag wird eine Basiskomponente, das sog. Remote-Cooperation-System, zur Realisierung asynchroner Auftragsbeziehungen als Ablaufumgebung eines kooperierenden Systems vorgestellt. Dabei stehen insbesondere Konzepte zur Beschreibung von Auftragsparametern, deren Übertragung zu anderen Systemkomponenten und die flexible Abbildung des verteilten Systems auf unterschiedliche Hardwareplattformen im Vordergrund. Das in einer ersten Version vollständig implementierte Remote-Cooperation-System dient derzeit als Implementierungsgrundlage bei der Realisierung eines kooperierenden Datenbank-Kernsystems.

## 1. Einleitung

Datenbanksysteme (DBS) für neuere Anwendungsbereiche, speziell für das Gebiet der Ingenieuranwendungen, bilden eine zentrale Herausforderung für die gegenwärtige Datenbankforschung. Neben dem Problem der Datenmodellierung, das sich im wesentlichen aus der strukturellen Vielfalt und der Komplexität der in diesen Anwendungsbereichen relevanten Objektstrukturen ergibt, ist ein Hauptproblem in der mangelnden Leistungsfähigkeit konventioneller DBS, insbesondere bei der Verarbeitung dieser komplexen Objektstrukturen, zu sehen. Das hieraus resultierende, überwiegend schlechte Antwortzeitverhalten führt bei den meisten interaktiven Anwendungen aus dem Ingenieurbereich zu einer Inakzeptanz solcher DB-gestützter Ingenieursysteme.

Einen vielversprechenden Ansatz zur Behebung dieses Mißstandes wird in der Nutzung von Parallelität bei der Bearbeitung einzelner Datenbankoperationen gesehen /HSS88/. Die auf eine Erhöhung des Durchsatzes ausgerichteten Konzepte für den Mehrbenutzerbetrieb bieten hierzu keinerlei Grundlage. Im Gegensatz zu dieser "konkurrierenden" Parallelität zwischen DB-Operationen verschiedener Benutzer, wie sie von konventionellen DBS angeboten wird, sind hier für die notwendige Reduktion der Antwortzeit Konzepte erforderlich, die eine "kooperierende" Parallelität innerhalb einzelner DB-Operationen unterstützen. Dies setzt ihre Zerlegung in einzelne Teilschritte voraus, die dann möglichst parallel abgewickelt werden können / AS83, Le89/.

Eine "zeitparallele" Abarbeitung der Teilschritte erfordert eine aus mehreren Prozessoren bestehende Hardware-Umgebung /HSS89/. Daher bilden die Modularisierung der DBS-Software und die Einbettung dieser Module in eine adäquate Mehrprozessor-Ablaufumgebung eine zentrale Voraussetzung für die Nutzung von Parallelität bei der DB-Verarbeitung. /HMMS88/ enthält hierzu einen Vorschlag für die Architektur eines kooperierenden DBS, der gegenwärtig im Rahmen des PRIMA-Projektes /Hä88/ in einer Prototypentwicklung

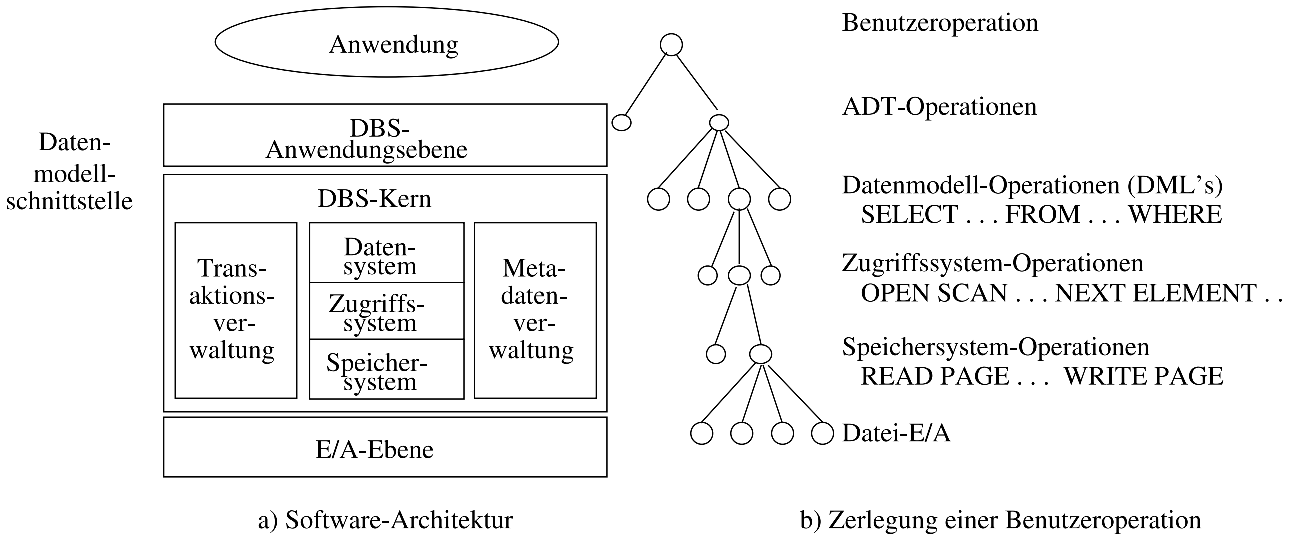


Bild1: Software-Architektur eines kooperierenden DBS

konkret umgesetzt wird. Das PRIMA-System bildet dabei eine Art Testumgebung u.a. für die Realisierung und Validierung verschiedener, paralleler DB-Verarbeitungsstrategien /HSS88/. Bild 1a skizziert hierzu die PRIMA-Grobarchitektur: Sie setzt sich aus der Ein-/Ausgabeebene, dem DBS-Kern und der sog. Anwendungsebene zusammen. Der auf den Zugriffsprimitiven der Ein-/Ausgabeebene aufsetzende DBS-Kern ist weiter unterteilt in Daten-, Zugriffs- und Speichersystem sowie in Transaktions- und Metadatenverwaltung. Die Transaktionsverwaltung besteht selbst wiederum aus Komponenten zur Durchführung der Synchronisation (Locking), der Protokollierung (Logging) und der Restauration (Recovery). Der DBS-Kern realisiert ein speziell entwickeltes Datenmodell /Mi88/, das die Beschreibung und die Handhabung komplex-strukturierter Objekte erlaubt. Auf diesem Kern aufbauend findet sich die Anwendungsebene, durch die das gesamte DBS auf die Unterstützung einer bestimmten Anwendungsklasse hin ausgerichtet wird. Hierzu bietet die Anwendungsebene an ihrer Schnittstelle bereits anwendungsnahe Objekte mit den entsprechenden Operationen an, was beispielsweise in Form Abstrakter Datentypen (ADT's) erfolgen kann.

Neben der Software-Architektur illustriert Bild 1b die hierarchische Zerlegung einer Benutzeroperation in einen Operationsbaum, dessen Ebenen das zugrundeliegende Architekturmodell widerspiegeln. Jeder Knoten entspricht einem Aufruf aus der übergeordneten Ebene und ist wiederum zerlegt in Suboperationen, die dann ihrerseits Aufrufe der darunterliegenden Ebene bewirken. Wie in /HHM86/ verdeutlicht, muß dieser Operationsbaum keineswegs strikt sequentiell abgearbeitet werden. Vielmehr können unter Ausnutzung der jeweiligen Operationssemantik die Suboperationen so zusammengefaßt werden, daß eine parallele Abwicklung möglich ist.

Die in unserem Fall vorherrschende Hardware-Umgebung ist insbesondere durch heterogene, leistungsfähige Arbeitsplatzrechner (Workstation) bestimmt, die über ein lokales Netz untereinander und mit einigen wenigen zentralen Dienstleistungsrechnern verbunden sind. Das Spektrum der zur Verfügung stehenden Rechner erstreckt sich von leistungsfähigen Einzelrechnern bis hin zu einem speicher-gekoppelten Mehrprozessor- bzw. Mehrrechnersystem mit einer hohen Speicher- und Verarbeitungskapazität. Neben der inhomogenen Hardware ist diese Umgebung durch die Heterogenität der vorhandenen Betriebssysteme bzw. Betriebssystemderivate bestimmt.

Die Erprobung paralleler DB-Verarbeitungsstrategien erfordert die Realisierung eines verteilten DBS bestehend aus kooperierenden Systemkomponenten. Eine wichtige Frage betrifft die Abbildung eines solchen verteilten Systems auf eine konkrete Hardware- und Betriebssystemumgebung. Aufgrund der Vielfalt und der Heterogenität dieses Umfeldes bietet sich eine zweistufige Abbildung an. Ein erster Schritt besteht hierbei in der Abbildung auf eine möglichst von Hardware und Betriebssystem unabhängige "Zwischenschicht", auf ein sog. **Basis-Kooperationssystem**. Dies bietet zum einen den Vorteil, daß die Programmierung eines kooperierenden Anwendungssystems weitestgehend unabhängig von der aktuellen Verteilung in einer realen Ablaufumgebung erfolgen kann, und daß zum anderen eine Ausrichtung der Zwischenschicht auf spezielle Kooperationsbedürfnisse möglich ist. In einem Basis-Kooperationssystem sollen damit wichtige Kooperationsfunktionen zusammgefaßt und dann in einem zweiten Schritt effizient auf die jeweils existierenden Hardware- und Betriebssystemgegebenheiten abgebildet werden.

Eine geeignete Unterstützung bei der Gestaltung verteilter Systeme muß eine Zuordnung der einzelnen Architekturbausteine eines Anwendungssystems (vgl. Bild 1a) zu Softwarekomponenten vorsehen. Darüber hinaus müssen Abhängigkeiten zwischen den Architekturbausteinen innerhalb des Basis-Kooperationssystems nachgebildet werden können. Eine der aus unserer Sicht vordringlichsten Aufgaben eines solchen Kooperationssystems ist darin zu sehen, eine für die *Realisierung des kooperierenden DBS möglichst transparente und flexible Abbildung* der benötigten *Systemkomponenten* auf die vorhandene *Hardware- und Betriebssystem-Umgebung* zu ermöglichen. Nur so können für die Evaluierung und die Erprobung paralleler DB-Verarbeitungsstrategien verschiedene Grade real möglicher Parallelität leicht eingestellt und die tatsächlich erreichte Parallelitätsausnutzung beobachtet und bewertet werden. Es sollte für die Anwendungsprogrammierung keinen Unterschied machen, ob nun ein einzelner Rechner oder ein ganzes Rechnernetz als reale Ablaufumgebung dient. Andererseits sollten aber auch die speziellen Eigenschaften der jeweils verwendeten Hardware (z.B. vernetzte Workstations, gemeinsamer Speicher in einem Rechner-Cluster, etc.) genutzt werden können.

Aufgrund allgemeiner Überlegungen aber auch spezieller Beobachtungen ergeben sich die im folgenden zusammengestellten Anforderungen (siehe auch /Schö90/):

- **Transparenz und Flexibilität** bzgl. der Zuordnung von Systemkomponenten zu Rechnern.
- Das Zusammenspiel der Komponenten, also deren Kooperation, erfolgt überwiegend nach dem **Client/ Server-Prinzip**. Jede Systemkomponente kann dabei die Rolle eines Servers für die übergeordnete Komponente übernehmen und gleichzeitig als Client für weitere Systemkomponenten fungieren.
- Parallele Aktivitäten zwischen den Systemkomponenten setzen die Möglichkeit einer **asynchronen Auftragserteilung** voraus.
- Um die logische Zerlegung in Suboperationen (insbesondere deren genaue Anzahl) unabhängig von der real verfügbaren Hardware zu halten, sollte jeder Server in der Lage sein, eine prinzipiell **beliebige Anzahl von Aufträgen** entgegenzunehmen und möglichst unabhängig voneinander zu bearbeiten. Im Zusammenhang hiermit ergibt sich die Forderung, die Auftragsabarbeitung durch eine entsprechende **Prioritätenvergabe** zu beeinflussen. Z.B. sollen Aufträge, die ein Transaktionsende einleiten, i. allg. vor Aufträgen bearbeitet werden, die den Beginn einer neuen Transaktion bewirken.
- Obwohl die Datenmodelloperationen in semantisch unabhängige Suboperationen zerlegt werden, können die durch sie ausgelösten Aufrufe auf tieferer Systemebene voneinander abhängig werden. Z.B. können aus Anwendungssicht unabhängige Daten auf zusammenhängende Datenstrukturen, beispielsweise eine

gemeinsame Speicherseite, abgebildet werden, was zwangsläufig zu **wechselseitigen Abhängigkeiten** bei den entsprechenden Synchronisationsaufrufen führt. Ein geeigneter Kooperationsmechanismus muß daher die Handhabung solcher Abhängigkeiten zwischen Aufträgen erlauben.

- Die DB-Verarbeitung ist typischerweise äußerst datenintensiv. In der Regel werden große und komplex-strukturierte Datenmengen bearbeitet. Daher muß es möglich sein, neben dem Kontrollfluß in Form der Auftragserteilung, den Datenfluß in Form **komplex-strukturierter Parameter** zu beschreiben und möglichst effizient umzusetzen. In Anbetracht der anstehenden Parametergröße muß ein evtl. vorhandener gemeinsamer Hauptspeicher vom Kooperationssystem genutzt werden.
- Aufgrund der heute bereits vorherrschenden Heterogenität der Hardware- und Betriebssystem-Ausstattung und der allgemein sehr raschen Entwicklung neuer Systeme besteht ein vitales Interesse an der **Portabilität** des Basis-Kooperationssystems.

Die von komerziell verfügbaren Betriebssystemen angebotenen Kooperationsmöglichkeiten sind in aller Regel nachrichtenorientiert, bieten nur geringfügigen Programmierkomfort oder erlauben lediglich synchrone Auftragsbeziehungen; ein evtl. vorhandener gemeinsamer Speicher wird nicht konsequent für Kooperationszwecke genutzt. Neuere Betriebssysteme, die als Prototypen in zahlreichen Forschungseinrichtungen entstehen, bieten zumindest teilweise die geforderte Unterstützung, sind aber i. allg. nicht auf so heterogener Hardware-Plattform verfügbar und können, als Experimentalsysteme ausgelegt, nicht die Zuverlässigkeit und Stabilität kommerzieller Betriebssysteme erreichen, die für eine Neuentwicklung komplexer Anwendungssysteme erforderlich ist. Daher wurde im PRIMA-Projekt eine Eigenentwicklung durchgeführt, die unter Ausnutzung möglichst allgemein verfügbarer Betriebssystemkonzepte den oben angesprochenen Randbedingungen genügt. Die generelle Zielsetzung dabei war, eine möglichst einfache zusätzliche Systemschicht auf bestehenden Betriebssystemen zu realisieren, die die geforderte Funktionalität bereitstellt. Obwohl für ein ganz spezielles Umfeld entwickelt, ist das im weiteren als **RCS** (Remote Cooperation System) bezeichnete Kooperationssystem durchaus von allgemeinerem Interesse. So sind die zentralen Anforderungen an Mechanismen zur parametrisierbaren auftragsbezogenen Kooperation und der parallelen bzw. asynchronen Auftragsabwicklung keineswegs exotisch; sie stellen vielmehr generelle Randbedingungen für die Gestaltung verteilter und parallel ablaufender Anwendungssysteme dar.

Der Aufsatz ist folgendermaßen gegliedert: In Kapitel 2 werden zunächst die grundlegenden Fragen der Abbildung und der Einbettung in eine konventionelle Betriebssystemumgebung diskutiert. In Kapitel 3 sind dann die Grobarchitektur und die einzelnen RCS-Bausteine beschrieben. Insbesondere werden die RCS-Anwendungsschnittstelle vorgestellt und die wichtigsten Realisierungskonzepte erläutert. Kapitel 4 enthält schließlich eine Zusammenfassung und einen Ausblick auf weitere Arbeiten.

## 2. RCS-Einbettung in eine existierende Betriebssystemumgebung

Entsprechend den oben aufgeführten Anforderungen stellt das RCS ein Hilfsmittel zur Realisierung verteilter Anwendungssysteme dar. Es unterstützt ortstransparente und asynchrone Auftragsbeziehungen zwischen einzelnen Komponenten eines verteilten Systems und erlaubt damit explizit eine zeitparallele Abwicklung von Aufträgen. Die Auftragsbearbeitung durch eine Systemkomponente kann durch die (parallele) Erteilung von Subaufträgen und das entsprechende Aufsammeln und Weiterverarbeiten ihrer Ergebnise erfolgen. Die in Bild 1a skizzierte Software-Architektur unserer DBS-Entwicklung kann somit auf vielfältige Weise auf eine unterschiedliche Anzahl von kooperierenden Systemkomponenten abgebildet werden, wobei jede Systemkomponente jeweils die Funktionalität eines entsprechenden Architekturbausteins (z.B. Datensystem, Zugriffssystem, etc.) realisiert. Der in Bild 1b illustrierte Operationsbaum wird aus Sicht des RCS in einer Auftragshierarchie nachgebildet, die dynamisch zwischen den einzelnen Systemkomponenten entsteht.

Die hierbei unterstellte Systemsicht, insbesondere die damit verbundenen Aufgaben des RCS, üben einen wesentlichen Einfluß auf Aspekte der RCS-Einbettung in eine konventionelle Betriebssystemumgebung aus. So sind zunächst betriebssystemseitige Ablaufeinheiten erforderlich, in die Systemkomponenten eingebettet werden können. Aufgrund der generellen Verfügbarkeit und der relativ einfachen und flexiblen Verteilbarkeit auf unterschiedliche Rechner, bietet sich hierzu das Prozeßkonzept an. Prozesse bilden aus Betriebssystemsicht die Einheiten der Isolation und der Zuteilung von Ressourcen (Prozessorzeit, Speicherplatz) und können i. allg. eindeutig einem Prozessor als physische Ablaufumgebung zugeordnet werden; sie besitzen damit bereits die wesentlichen Eigenschaften, die den Komponenten eines verteilten Systems zuzuschreiben sind. Für Systemkomponenten, die auf **einem** Prozessor ausgeführt werden, kommt jedoch auch das Konzept der *Threads* oder *Lightweight Processes* /Sun89/ in Betracht. Hierbei würden die auf einem Prozessor ablaufenden Komponenten eines verteilten Systems auf verschiedene Threads eines einzigen Prozesses abgebildet werden. Da Threads keine Isolationseinheiten bilden und in einem gemeinsamen Adreßraum ausgeführt werden, ergeben sich sehr effiziente Kommunikationsmöglichkeiten. Daneben ist ein Wechsel bei der Thread-Ausführung mit weitaus weniger Overhead verbunden, als dies bei einem herkömmlichen Prozeßwechsel der Fall ist. Diesen Vorteilen steht allerdings der Nachteil gegenüber, daß in Abhängigkeit von der konkreten Rechnerzuordnung der Systemkomponenten unterschiedliche Betriebssystemeinbettungen vorzunehmen sind. Ein weiterer Nachteil ist in der heute noch geringen Verbreitung und Allgemeingültigkeit der erforderlichen Konzepte zu sehen. Im Rahmen der RCS-Entwicklung ist daher die Einbettung von Systemkomponenten in Betriebssystemprozesse realisiert.

Eine weitere wichtige Abbildungsfrage ergibt sich aus der Forderung nach der Entgegennahme und der **unabhängigen Bearbeitung mehrerer Aufträge durch eine einzige Systemkomponente**. Hierzu kommt das Konzept des *Multi-Processing* sowie das Konzept des *Multi-Tasking* in Frage. Im Falle des Multi-Processing wird die Funktionalität einer Server-Komponente nicht nur durch einen einzelnen Prozeß erbracht, vielmehr sind je nach Auftragseingang eine beliebige Anzahl von Server-Prozessen zu erzeugen. Jeder Prozeß bearbeitet dabei nur einen Auftrag (Single Tasking), was sich ohne Zweifel positiv auf die Einfachheit und Natürlichkeit der Server-Programmierung auswirkt. Dem gegenüber steht allerdings der relativ hohe Verwaltungsaufwand durch das Betriebssystem, was sich insbesondere bei einer großen Anzahl von notwendigen Prozessen bemerkbar macht. Zudem wird bei auftretenden Abhängigkeiten zwischen einzelnen Aufträgen eine zusätzliche Kommunikation der jeweils zuständigen Server-Prozesse erforderlich. Wir haben uns daher bei der Realisierung des RCS für den Einsatz des Multi-Tasking-Konzeptes entschieden, bei dem ein einzelner Prozeß (Single Processing) mehrere unabhängige Aufträge verzahnt bearbeitet. Für jeden Auftrag wird dabei prozeßintern eine geeignete Datenstruktur angelegt, durch die der aktuelle Bearbeitungszustand des jeweiligen Auftrages festgehalten wird. Durch diese explizit durchgeführte Auftragsverwaltung wird zudem die Realisierung einer flexiblen und anwendungsorientierten Prioritätensteuerung bei der Auftragsabwicklung erleichtert.

Weniger durch die speziellen Anforderungen als aufgrund allgemeiner Effizienzüberlegungen bestimmt, besitzt das RCS selbst eine verteilte Systemstruktur. Es gibt keine zentrale Instanz des RCS, die quasi als "Kooperationszentrale" die Verwaltung der erforderlichen Systeminformationen übernimmt, vielmehr sind alle

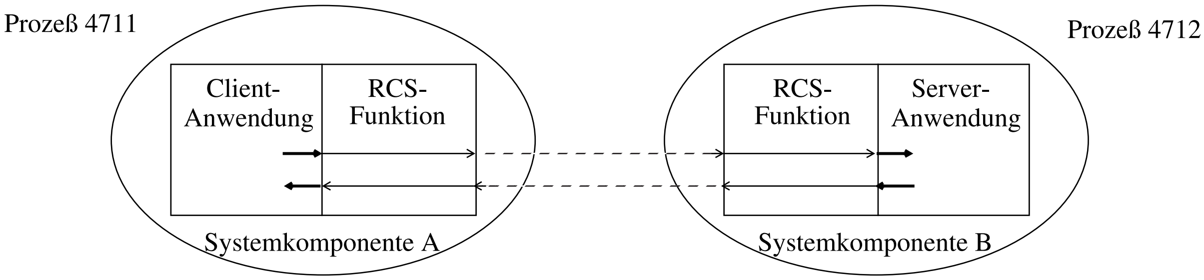


Bild 2: Abbildung der RCS-Systemkomponenten auf allgemeine Betriebssystemkonzepte

Verwaltungsdaten auf die entsprechenden Server-Prozesse verteilt. Zusammen mit den jeweils erforderlichen Daten sind alle RCS-Funktionen unmittelbar an die Client- oder Server-Anwendungsprogramme gebunden, sie werden also "inlinked" in einem gemeinsamen Adreßraum ausgeführt. Zwischen auftraggebender und auftragnehmender Systemkomponente besteht eine "direkte" Verbindung, wodurch in der Regel das durch die Kooperation verursachte Kommunikationsaufkommen reduziert wird. Allerdings tritt dabei eine Gefährdung sensitiver RCS-Daten durch fehlerhafte Anwendungsprogramme auf. Dies kann in dem von uns speziell betrachteten Fall eines kooperierenden DBS in Kauf genommen werden, da es sich bei den RCS-Anwendungsprogrammen um DBS-Module und damit um systemnahe und "vertrauenswürdige" Programme handelt.

Im folgenden sind die wichtigsten Einbettungsaspekte nochmals zusammengefaßt (vgl. Bild 2): Systemkomponenten werden auf Prozesse des Betriebssystems abgebildet und setzen sich aus einem anwendungsbezogenen Teil und einem RCS-spezifischen Teil zusammen. Der anwendungsbezogene Teil, das RCS-Anwendungsprogramm, erfüllt die eigentliche Aufgabe der betroffenen Systemkomponente. Der RCS-spezifische Teil realisiert die lokale RCS-Funktionalität und verwaltet alle die jeweilige Komponente betreffenden RCS-Daten. Hierzu zählen u.a. die durch das Multi-Tasking-Konzept bedingten auftragsbezogenen Verwaltungsdaten sowie weitere wichtige Systeminformationen, wie beispielsweise die Ansprechstellen und die Verbindungsinformation für die Nutzung der durch andere Server bereitgestellten Funktionen.

## 3. Das Remote-Cooperation-System

Im folgenden wollen wir nun das von uns realisierte System etwas genauer vorstellen. Bild 3 zeigt die Grobarchitektur des RCS. Neben der RCS-Anwendungsschnittstelle, über die den Komponenten eines verteilten Systems die RCS-Funktionalität bereitgestellt wird, besteht das RCS aus einer Auftragsverwaltung, einem Modul zur Auftrags- und Parameterübertragung sowie aus einer Speicherverwaltung und einem Kommunikationsdienst. Zunächst werden die RCS-Funktionen vorgestellt und an einem Beispiel erläutert. Insbesondere werden Aspekte des Datenaustausches über Auftragsparameter diskutiert. Dabei stehen die beschreib-

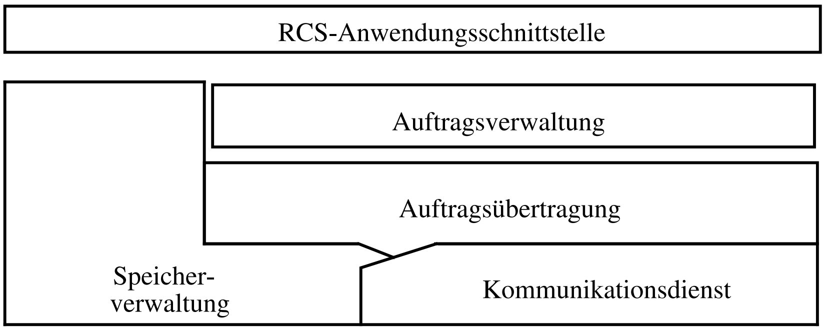


Bild 3: Grobarchitektur des Remote-Cooperation-Systems

bare Parametersemantik und die Art der Parameterübertragung im Vordergrund. Im Anschluß daran werden dann die weiteren RCS-Module beschrieben.

### 3.1 Die RCS-Anwendungsschnittstelle

Die an der RCS-Schnittstelle zur Verfügung stehenden Operationen unterstützen eine auftragsbezogene Kooperation nach dem Client/Server-Konzept, wobei jede Komponente sowohl als Client als auch als Server auftreten kann. Die Kooperation zwischen den einzelnen Komponenten erfolgt dabei stets in den folgenden Schritten:

(1) Anstoßen einer Server-Funktion durch den Client (Auftragserteilung)

(2) Entgegennahme eines entsprechenden Auftrages durch den Server (Auftragsentgegennahme)

(3) Ausführung des Auftrages durch den Server (Auftragsabwicklung)

(4) Übermittlung von Ergebnissen an den Client (Antworterzeugung)

(5) Abholen der Ergebnisse durch den Client (Antwortentgegennahme)

Um die Möglichkeit einer parallelen Verarbeitung zwischen Client und Server zu bieten, erfolgt die Auftragserteilung asynchron nach dem Konzept des *Remote-Server-Invocation* /SE86/, d.h., Auftragserteilung und Antwortentgegennahme sind als unabhängige RCS-Funktionen konzipiert. Durch sukzessives Anstoßen von Server-Funktionen kann damit auch Parallelität zwischen der Auftragsabwicklung verschiedener Server erreicht werden. Für die Auftragsabwicklung auf Server-Seite wird wie oben begründet ein Multi-Tasking-Mechanismus angeboten, der eine verzahnte Abarbeitung von Aufträgen durch einen Server-Prozeß vorsieht. Im einzelnen ergibt sich auf Server-Seite das folgende schrittweise Vorgehen:

(1) Auftragsentgegennahme

(2) Aufschlüsselung des Bearbeitungskontextes

(3) Ausführung weiterer Teilschritte zur Auftragsabwicklung

(4) Falls Auftrag vollständig bearbeitet: Antworterzeugung und Ergebnisübermittlung;
Sonst Sicherung des aktuellen Verarbeitungskontextes und Unterbrechung der momentanen Auftragsbearbeitung; Bearbeitung eines nächsten Auftrags.

Dabei sollten durch die Server-Komponente jeweils möglichst viele Teilschritte zur Auftragsabwicklung durchgeführt werden und eine Bearbeitungsunterbrechung nur dann eingeleitet werden, wenn "externe" Abhängigkeiten auftreten, wenn also auf das Eintreten bestimmter Ereignisse gewartet wird, z.B. auf die Ergebnisübermittlung durch einen weiteren selbst beauftragten Server oder auf die Änderung eines globalen Datums durch andere Aufträge des gleichen Servers (z.B. das Freigeben einer Sperre).

In Bild 4 sind die einzelnen RCS-Operationen im Überblick zusammengefaßt. Mittels "Rc_Init" bzw. "Rc_Terminate" kann sich eine Anwendungskomponente jederzeit als Client und/oder als Server einem verteilten System hinzufügen bzw. sich daraus entfernen. Voraussetzung für eine Kooperation zwischen einem Client und einem Server ist eine logische Verbindung zwischen den entsprechenden Prozessen, die durch "Remote_Server_Start" und "Remote_Server_End" dynamisch auf- und abgebaut werden kann. Nach dem Aufbau einer solchen logischen Verbindung kann ein Client beliebige Dienstleistungen eines Servers veranlassen ("Remote_Server_Initiation"). Die RCS-Client-Seite übernimmt dabei den Auftrag und sorgt für dessen Weiterleitung an die entsprechende Server-Seite. Unmittelbar anschließend an die Weiterleitung erhält die Client-Anwendung die Kontrolle zurück, so daß es sich tatsächlich um eine asynchrone Auftragserteilung handelt.

Das Ergebnis eines derartigen Auftrags kann dann zu einem späteren Zeitpunkt über die RCS-Operation "Get_Result" abgerufen werden. Dabei können sowohl bei "Remote_Server_Initiation" als auch bei "Get_Result" eine beliebige Anzahl komplex-strukturierter Parameter angegeben werden. Der Aufruf von "Get_Result" setzt voraus, daß der betreffende Auftrag bereits beendet wurde. Daher bietet die Operation "Look_For_Server_Termination" die Möglichkeit, den aktuellen Bearbeitungszustand eines Auftrags bzw. einer Liste von Aufträgen zu erfragen, während mit "Wait_For_Server_ Termination" explizit auf das Ende der Bearbeitung eines in einer Auftragsliste enthaltenen Auftrags gewartet werden kann. Die Operation "Abort_Server_Processing" ermöglicht dem Client den Abbruch eines aus seiner Sicht noch aktiven Auftrags (d.h., er hat noch kein "Get_Result" ausgeführt). Dies bewirkt, daß der Auftrag auf der entsprechenden Server-Seite entweder überhaupt nicht bearbeitet wird, oder aber, daß seine Bearbeitung zum frühest möglichen Zeitpunkt beendet wird. Die eventuell durchgeführten Änderungen werden nicht automatisch revidiert, vielmehr kann anwendungsseitig eine spezielle Abortbehandlung definiert werden, die in den entsprechenden Fällen durch das RCS aktiviert wird.

Fordert auf Server-Seite ein Auftragnehmer mittels "Accept_Task" einen Auftrag zur Bearbeitung an und liegt aktuell kein Auftrag vor, wird er in einen Wartezustand versetzt. Ansonsten erhält er Informationen über den auszuführenden Auftrag. Dazu gehört insbesondere die sogenannte Kontext-Information, die bei einem Multi-Tasking-Betrieb den momentanen Bearbeitungskontext eines Auftrags enthält. Diese Kontext-Information ist zu Beginn eines Auftrags zunächst undefiniert. Sie muß im Verlaufe der Auftragsbearbeitung durch die jeweiligen Server-Anwendungen in geeigneter Weise initialisiert und gewartet werden. Mit "Break_Task" kann die Bearbeitung des aktuellen Auftrags unterbrochen werden. Dabei kann eine Liste von eigenen Subaufträgen angegeben werden. Der unterbrochene Auftrag wird in diesem Fall erst dann weiterbearbeitet, wenn mindestens einer dieser Subaufträge ein Ergebnis zurückgeliefert hat. Am Ende einer

| Operation | Beschreibung |
|---|---|
| Rc_Init | macht eine Komponente, die als Client und/oder Server arbeitet, dem RCS bekannt. |
| Rc_Terminate | zeigt dem RCS an, daß ein Server nicht mehr zur Verfügung steht. |
| Remote_Server_Start | stellt eine logische Verbindung zwischen einem Client und einem Server her. |
| Remote_Server_End | beendet eine logische Verbindung zwischen einem Client und einem Server. |
| Remote_Server_Initiation | startet einen Auftrag, der vom betreffenden Server asynchron ausgeführt wird. |
| Get_Server_Result | ruft das Ergebnis eines beendeten Auftrags ab. |
| Wait_For_Server_Termination | blockiert einen Client bis einer der angegebenen Aufträge beendet ist. |
| Look_For_Server_Termination | liefert den aktuellen Bearbeitungszustand der angegebenen Aufträge. |
| Abort_Server_Processing | bricht die Bearbeitung eines Auftrags ab. |
| Accept_Task | übergibt einen Auftrag an einen Server. |
| Break_Task | unterbricht die Bearbeitung eines Auftrags, bis einer der angegebenen Subaufträge beendet ist. |
| Reply_Task | übergibt am Ende eines Auftrags das ermittelte Ergebnis an den betreffenden Client. |
| Wait_For_Event | blockiert einen Auftrag, bis ein bestimmtes Ereignis eingetreten ist. |
| Signal_Event | signalisiert das Eintreten eines bestimmten Ereignisses. |

Bild 4: Die wichtigsten Operationen des RCS

Auftragsbearbeitung kann das Resultat des Auftrags mittels "Reply_Task" an den Client weitergeleitet werden.

Für die Handhabung von wechselseitigen Abhängigkeiten zwischen Aufträgen werden zwei zusätzliche Operationen, "Wait_For_Event" und "Signal_Event", zur Verfügung gestellt. "Wait_For_ Event" unterbricht dabei einen Auftrag solange, bis ein bestimmtes Ereignis eintritt, das von einem weiteren Auftrag mittels "Signal_Event" angezeigt wird. Als Beispiel für eine derartige Abhängigkeit zwischen Aufträgen kann der oben bereits angedeutete Fall der Synchronisation von Datenzugriffen dienen.

Zur Verdeutlichung der Wirkungsweise einzelner RCS-Operationen ist in Bild 5 das Zusammenwirken in einer einfachen Client/Server-Stuktur dargestellt. Die in Bild 5 grau hinterlegte Aufrufsequenz beschreibt eine parallel zur Client-Anwendung abgewickelte Auftragsbearbeitung durch eine Server-Anwendung. Sowohl bei der Auftragserteilung als auch bei dem Ergebnisabruf können Parameter übergeben werden. Durch die Übergabe wechselt die Kontrolle über den entsprechenden Parameter von der einen Systemkomponente zur anderen. Aufgrund der Systemverteilung und der parallelen Abwicklung ergeben sich hierbei eine Reihe

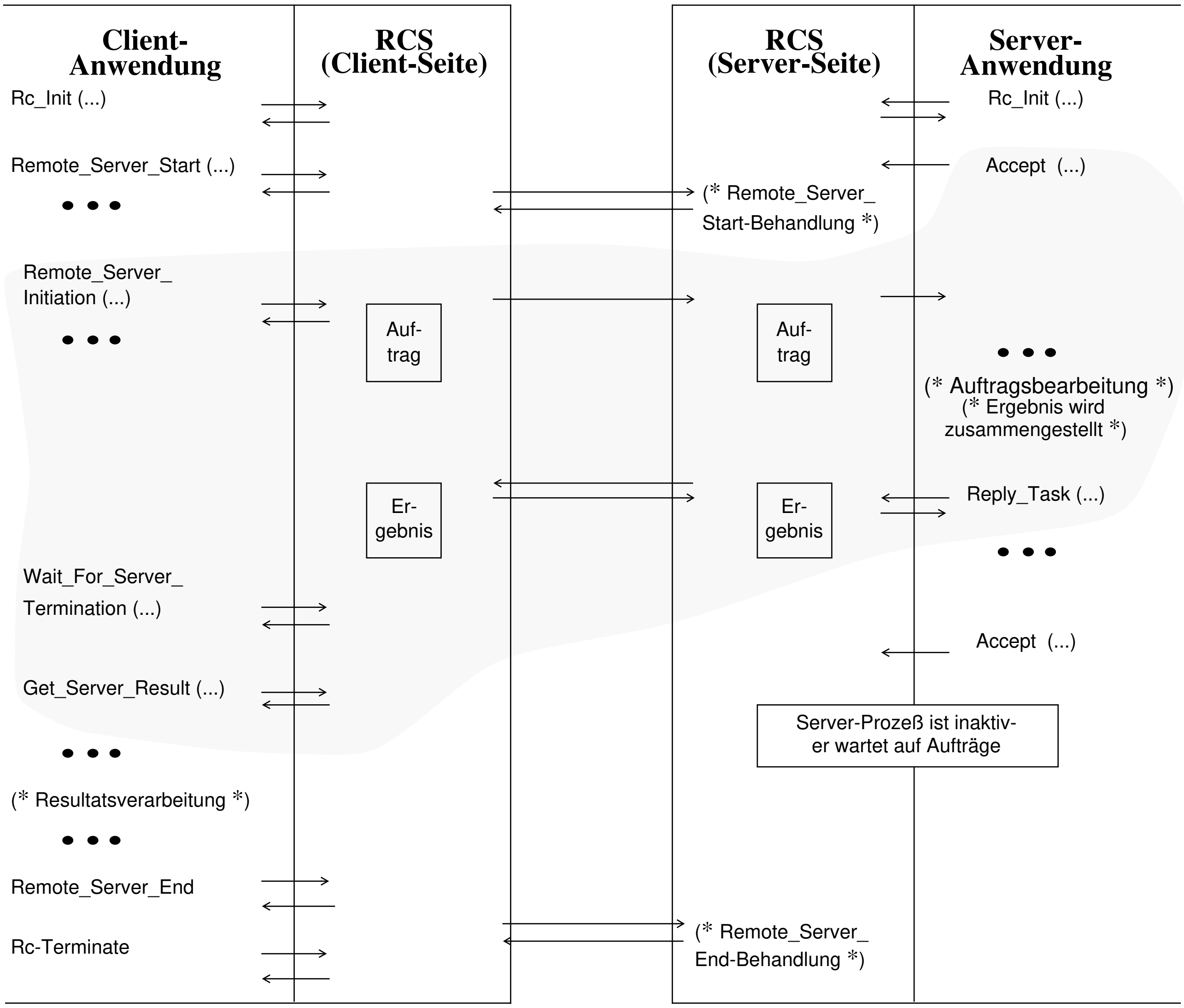


Bild 5: Beispielkooperation zwischen Client- und Server-Komponente

von Problemen, deren Lösung spezielle Absprachen bzgl. der Semantik der Parameterübergabe erforderlich machen.

### 3.2 Parametersemantik

Ähnlich wie im Fall synchroner Prozeduraufrufe können bei der asynchronen Auftragserteilung und Ergebnisübernahme mehrere Parametersemantiken unterschieden werden. Diese lassen sich analog zum synchronen Fall zunächst in die

- **Kontrollsemantik**
  (entspricht "call by reference"), d.h., die Kontrolle über den Parameter wird für die Dauer des Auftrags an den Auftragnehmer abgegeben, und die
- **Kopiersemantik**
  (entspricht "call by value"), d.h., der Parameterwert wird vom Auftraggeber zum Auftragnehmer kopiert,

unterteilen. Zusätzlich ergibt sich jedoch aufgrund der asynchronen Aufrufe und der damit verbundenen parallelen Verarbeitung zwischen Auftraggeber und Auftragnehmer eine weitere Parametersemantik, die

- **Behaltesemantik:**
  Die Aufteilung des Auftrags in zwei getrennte Aufrufe für Auftragserteilung und Ergebnisübermittlung erlaubt eine unterschiedliche Parametrisierung beider Operationen: ein Parameter, der bei der Auftragser-

teilung übergeben wurde, muß nicht notwendigerweise bei der Ergebnisübermittlung zurückgegeben werden; er dient dann lediglich als Eingabeparameter und kann damit vollständig in der Zuständigkeit des Auftragnehmers verbleiben, d.h., bei der Ergebnisübernahme erfolgt für diesen Parameter keine Angabe. Analoges gilt für Parameter, die nur bei der Ergebnisübernahme, nicht aber bei der Auftragserteilung auftreten.

Damit ergibt sich die in Bild 6 dargestellte Semantik der Parameterübergabe. Ein Parameter, der bei Auftragserteilung übergeben und bei Ergebnisübermittlung zurückgenommen wird (der "klassische" Fall), kann jeweils kopiert werden (**in-out** --> "call by value"), wenn der Parameter von Auftraggeber und Auftragnehmer gebraucht wird, oder es kann jeweils die Kontrolle über den Parameter übergeben werden (**exclusive** --> "call by reference"), wenn der Parameter nur vom Auftraggeber oder nur vom Auftragnehmer benötigt wird. Wird der Parameter nach der Auftragserteilung zwar weiterhin vom Auftraggeber benötigt, aber bei der Ergebnisübernahme nicht mehr zurückgenommen (-> Behaltesemantik), so muß er kopiert werden (**copy-in**), bzw., wenn er vom Auftraggeber nicht mehr benötigt wird, kann er unter die Kontrolle des Auftragnehmers gestellt werden (**control-in**). Analog sind bei der Ergebnisübernahme und der Behaltesemantik **copy-out** und **control-out** zu unterscheiden. Die Kombination der Kontroll- und Kopiersemantik (in Bild 6 durch * gekennzeichnet) ist nicht sinnvoll, da der Zusammenhang zwischen Eingabe- und Ausgabeparameter verlorengeht. Beispielsweise würde im Falle von **copy-in/control-out** die Kontrolle über die **Kopie** des Parameters zurückgegeben werden.

Zusätzlich zu diesen unterschiedlichen Parametersemantiken sind vom RCS auch umfangreiche und komplex-strukturierte Parameter (Bäume oder Netze) zu behandeln. Wird hierbei die Parametersemantik **copy-in** oder **copy-out** verwendet, so muß das RCS in der Lage sein, auch diese komplex-strukturierten Parameter zu kopieren. Da im RCS die Struktur des Parameters nicht bekannt ist, muß dem RCS anwendungsseitig eine entsprechende Prozedur übergeben werden, die den Kopiervorgang übernimmt.

Diese **naheliegendste** Lösung, nämlich eine Prozedur zum Kopieren des Parameters, löst zwar das geschilderte Problem, ist aber ungeeignet, um den Parameter über ein Netzwerk zu übertragen. Hierzu muß der Parameter **linearisiert**, d.h. zu einem zusammenhängenden Bytestring transformiert werden können. Zusammen mit einer Prozedur zum **Entlinearisieren** der Parameter, d.h. der Erzeugung der Verweisstruktur eines Parameters, ist damit auch das Kopieren des Parameters möglich. Aus diesem Grund müssen beim Verbindungsaufbau zwischen zwei RCS-Komponenten dem RCS zwei Prozeduren übergeben werden, die das Linearisieren und das Entlinearisieren der für diese Verbindung definierten Parameter erlauben.

### 3.3 Parameterübertragung

Bild 7 zeigt die zur Parameterübertragung grundsätzlich notwendigen Schritte auf. Wenn die hierarchisch- bzw. netzwerkartig-strukturierten Parameter über ein Netzwerk übertragen werden sollen, so müssen sie li-

| Ergebnisübergabe<br>Auftragserteilung | keine Ergebnisübergabe | Kopie | Kontrolle |
|---|---|---|---|
| keine Auftragsparameterübergabe | — | copy-out | control-out |
| Kopie | copy-in | in-out | * |
| Kontrolle | control-in | * | exclusive |

Bild 6: Mögliche Parameteroptionen bei Client/Server-Beziehungen

nearisiert bzw. entlinearisiert werden (Schritte (1) und (5)). Bei der Linearisierung wird ein zusammenhängender Bytestring erzeugt, wozu in der Regel ein Kopiervorgang notwendig ist. Da die Übertragung der Parameter asynchron zur Anwendung erfolgen muß (Auftragserteilung und Ergebnisübernahme erfolgen asynchron zur eigentlichen Anwendung), ist eine zusätzliche Entkopplung der Anwendung vom RCS notwendig (Schritte (2) und (4)). Letztendlich beschreibt Schritt (3) die (physische) Übertragung der Nachricht. Prinzipiell bedeutet jeder dieser fünf Schritte das Kopieren der Nachricht.

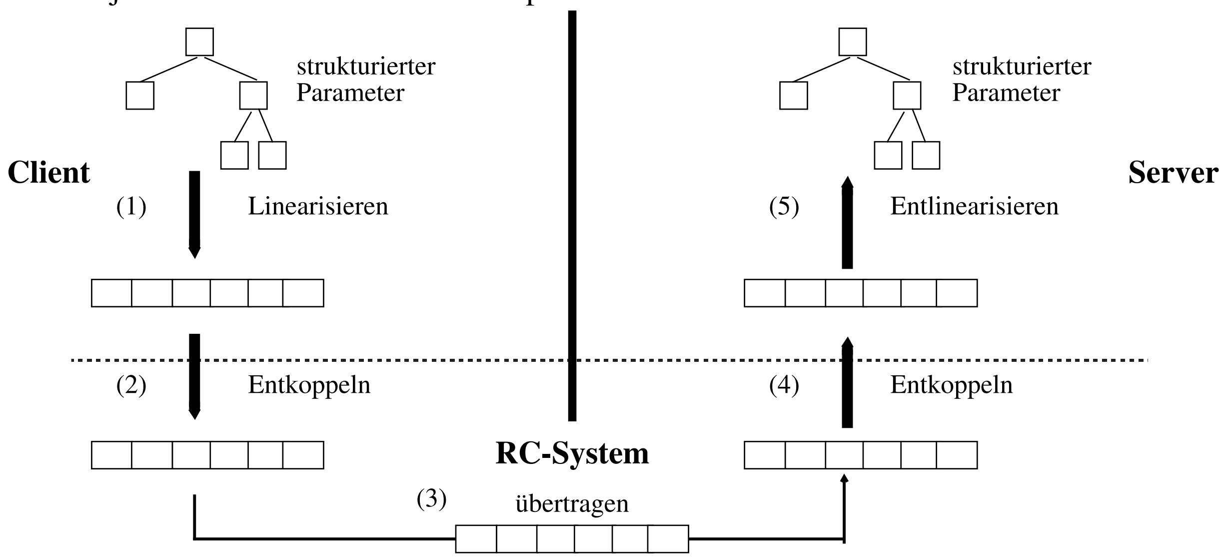


Bild 7: Allgemeines Szenarium der Parameterübertragung bei Auftragserteilung

Die Art und Weise, wie Parameter zwischen Auftraggeber und Auftragnehmer übertragen werden, hängt im wesentlichen von zwei Faktoren ab:

(1) der Parametersemantik und

(2) der Hardware-Umgebung.

Das RCS garantiert eine Übertragung der Parameter gemäß der angegebenen Semantik unabhängig von der zur Verfügung stehenden Hardware. Eventuell vorhandener gemeinsamer Hauptspeicher soll zur Parameterübertragung optimal ausgenutzt werden. Hierzu ist es notwendig, daß die Parameter bereits im gemeinsamen Speicher allokiert wurden. Dies ist automatisch erfüllt, wenn nur gemeinsamer Speicher zur Verfügung steht, wie dies üblicherweise bei speicher-gekoppelten Mehrprozessorsystemen der Fall ist. Häufig steht jedoch gemeinsamer Speicher nur in beschränkter Größe und mit erhöhten Zugriffskosten zur Verfügung (z.B. in SunOS 4.0 zwischen Prozessen eines Prozessors oder in speziellen Hardware-Architekturen /En89, Sy88/, bei nahe gekoppelten Rechnersystemen). Deshalb enthält das RCS eine Komponente zur Speicherverwaltung, die generell zwei Speicherkategorien unterscheidet:

- **Lokale Speicherbereiche** sind Bereiche, die nur in einer einzigen Anwendungskomponente verwendet werden und deren Kontrolle stets bei der jeweiligen Komponente verbleibt.
- Für **globale Speicherbereiche** kann dagegen die Kontrolle, also alle Rechte und Pflichten (Zugriffsrecht/ Freigabepflicht entsprechend der Parametersemantik), an eine andere Komponente übergeben werden.

*Das RCS garantiert, daß diese Speicherkategorien unabhängig von der zugrundeliegenden Hardware und vollständig transparent für die Anwendung realisiert werden.* Die Anwendung muß alle Parameter zur Auftrags- oder Ergebnisübermittlung in globalem Speicher, alle sonstigen Daten in lokalem Speicher allokieren. Unter dieser Voraussetzung zeigt Bild 8 die Anzahl der notwendigen Kopiervorgänge bei der Auftragsübermittlung bzw. Ergebnisentgegennahme in Abhängigkeit von der Parametersemantik und der zur Verfügung stehenden Hardware.

| Parametersemantik | copy-out | control-out | copy-in | control-in | in-out | exclusive |
|---|---|---|---|---|---|---|
| gemeinsamer Speicher | 1+1 | **0** | 1+1 | **0** | 2+2 | **0** |
| Netzwerk | 2+1 | 2+1 | 2+1 | 2+1 | 4+2 | 4+2 |

Bild 8: Notwendige Kopier- bzw. Linearisierungs- und (+) Entlinearisierungsvorgänge bei der Parameterübertragung abhängig von der Parametersemantik und der Hardware-Umgebung

Im Falle der **Behaltesemantik** (der Parameter ist also nur in eine Richtung zu übertragen) muß ein Parameter nur einmal kopiert bzw. linearisiert und einmal entlinearisiert werden. Falls kein gemeinsamer Speicher vorhanden ist, muß zusätzlich eine Übertragung über ein Netzwerk erfolgen. Wenn im Falle der Kontrollsemantik (**control-***) kein gemeinsamer Speicher vorhanden ist, so muß dieser mittels eines Kopiervorganges über das Netzwerk simuliert werden. **In-out** und **exclusive** erfordern jeweils den doppelten Aufwand, da der Parameter hin und zurück übertragen werden muß.

### 3.4 Auftragsverwaltung

Die Auftragsverwaltung des RCS gliedert sich in die Verwaltung der abgesetzten Aufträge eines Clients und das Scheduling der erhaltenen Aufträge eines Server. Der Client erhält bei Erteilung eines Auftrages eine Auftragsnummer zurück, die bei weiteren Aufrufen des RCS zur eindeutigen Identifizierung des Auftrags verwendet wird. Die Auftragsnummern können in sogenannten Auftragslisten organisiert werden, zur deren Handhabung das RCS Funktionen zum Erzeugen, Konkatenieren und Löschen von Auftragslisten sowie das Einfügen und Löschen von Auftragsnummern in bzw. aus Auftragslisten bereitstellt.

Die Steuerung der Auftragsabwicklung auf der Server-Seite gestaltet sich ungleich schwieriger, da die Auswahl eines Auftrags zur Bearbeitung durch den Server das gesamte Systemverhalten maßgeblich beeinflußt. Dies wird besonders am Beispiel eines Synchronisations-Servers deutlich, der durch die Vergabe von Lese- bzw. Schreibsperren den Zugriff auf gemeinsame Daten serialisiert. Ein solcher Server muß bestrebt sein, die Aufträge zur Freigabe von Sperren immer zuerst zu bearbeiten, da dadurch sowohl die Blockierungszeiten als auch die Konfliktwahrscheinlichkeit positiv beeinflußt werden. Da die Scheduling-Strategie offensichtlich server-spezifisch ist (Freigabe von Sperren), muß sie von außen in das RCS eingebracht werden können.

Für jeden Auftrag wird durch das RCS eine Gesamtauftragspriorität ermittelt, nach der ein Auftrag beim Neu- bzw. Wiedereintritt in die Liste der bearbeitbaren Aufträge des Servers eingeordnet wird. Der Auftrag mit der höchsten Priorität wird als nächster Auftrag durch den Server bearbeitet. Die Gesamtpriorität eines Auftrags setzt sich aus fünf Bestimmungsgrößen zusammen, nämlich der auszuführenden Funktion, dem auftraggebenden Client, dem Zustand des Auftrags, die vom Auftraggeber mitgelieferte Priorität für den Auftrag und dem Verhältnis von Wartezeit und Verweilzeit des Auftrags.

### 3.5 Auftragsübertragung

Der Modul zur Auftragsübertragung stellt Primitive zur auftragsbezogenen Kommunikation zwischen RCS-Anwendungskomponenten zur Verfügung. Er verwaltet alle Daten bzgl. der laufenden Aufträge (und Ergebnisse) sowie alle Daten über bestehende Verbindungen zu anderen RCS-Anwendungskomponenten, insbesondere auch die Art der Verbindung (Netzwerk oder gemeinsamer Speicher). Zu den auftragsbezogenen Daten zählen der Zustand des Auftrags, der Kontext des Auftrags und die Priorität des Auftrags. Ein Auftrag kann neu, unterbrochen, abgebrochen oder beendet sein. Für unterbrochene Aufträge müssen der aktuelle Verarbeitungskontext und ggf. Ereignisse, auf die der Auftrag wartet, verwaltet werden. Weiterhin müssen nach jeder Auftragsbearbeitung die Priorität des Auftrags neu berechnet und die Aufträge anhand ihrer Priorität (und ihres Zustandes) neu geordnet werden. Die Auftrags- und Parameterübertragung zu anderen RCS-Komponenten erfolgt mit Hilfe des Kommunikationsdienstes bzw. der Speicherverwaltung.

### 3.6 Kommunikationsdienst

Wenn zwischen zwei Prozessen kein gemeinsamer Hauptspeicher zur Verfügung steht, so muß der Datenaustausch nachrichtenbasiert über ein Netzwerk erfolgen. Im Fall von UNIX-Betriebssystemderivaten stehen rechnerübergreifende Mechanismen zur Interprozeßkommunikation zur Verfügung. Die Kontaktstellen solcher Kommunikationspfade werden als *Sockets* bezeichnet /CS87/. Auf Sockets werden Operationen zum Senden und Empfangen von Daten mit einer ähnlichen Semantik wie das Schreiben und Lesen auf Dateien zur Verfügung gestellt. Es werden zwei Typen von Sockets unterschieden, Stream-Sockets und Datagram-Sockets (vgl. /Sun87/). Zur Realisierung des RCS wurden Datagram-Sockets gewählt, da sie die auftretenden Anforderungen besser erfüllen /HKS90/.

### 3.7 Speicherverwaltung

Die Aufgabe der Speicherverwaltung besteht in der Bereitstellung einer einheitlichen Schnittstelle zum Anfordern und Freigeben von lokalem bzw. von globalem Speicher, wobei letzterer zur Aufnahme von Auftragsparametern benötigt wird. Lokaler Speicher wird dabei stets innerhalb des lokalen Adreßraums desjenigen Prozesses beschafft, der die gerade anfordernde Systemkomponente ausführt. Bei Anforderung von globalem Speicher ist von der Speicherverwaltung zunächst zu klären, ob im konkreten Fall tatsächlich gemeinsamer Speicher vorhanden ist oder nicht, d.h., ob es von Betriebssystem- bzw. Hardware-Seite überhaupt die Möglichkeit gibt, prozeß- bzw. prozessorübergreifend auf gemeinsame Daten zuzugreifen. Ist dies der Fall, so kann die globale Speicheranforderung unmittelbar auf gemeinsamen Speicher abgebildet werden. Die entsprechend abgelegten Parameter können dann relativ einfach von einer Systemkomponente zur anderen übertragen werden (vgl. Abschnitt 3.3). Ist dagegen kein gemeinsamer Speicher verfügbar, so wird die Anforderung von globalem Speicher ebenfalls aus dem prozeßlokalen Adreßraum erfüllt. Parameter, die in diesem Bereich abgelegt sind, müsen dann allerdings bei einer Übertragung durch das RCS kopiert und ggf. linearisiert werden. Aus Sicht der RCS-Anwendung macht sich dies lediglich in einer weniger effizienten Parameterübertragung bemerkbar.

## 4. Zusammenfassung

In diesem Aufsatz haben wir das Remote-Cooperation-System vorgestellt, das als Basiskomponente zur Realisierung eines koopierenden Datenbanksystems entwickelt wurde und derzeit erfolgreich in der Implementierungs- und Testphase des PRIMA-Prototypsystems eingesetzt wird. Das RCS bietet eine Ablaufumgebung an, die die Gestaltung eines verteilten Systems, bestehend aus kooperierenden Systemkomponenten, ermöglicht und somit die Realisierung von parallelen DB-Abarbeitungsstrategien unterstützt. Das RCS ist in einer heterogenen Hardware-Umgebung lauffähig. Die Kooperation zwischen den Komponenten erfolgt überwiegend nach dem Client/Server-Prinzip durch asynchrone Auftragsbearbeitung. Jede Komponente kann gleichzeitig als Client- und als Server-Komponente auftreten. Die Ausführung paralleler Aktivitäten zwischen den Komponenten setzt die Fähigkeit von asynchronen Server-Aufrufen voraus. Jeder Server kann beliebig viele Aufträge im Multi-Tasking-Prinzip bearbeiten, so daß die Anzahl der Aufträge unabhängig von der konkreten Hardware-Umgebung ist.

Besonderes Interesse gilt der Parameterübergabe bei der asynchronen Auftragserteilung und Ergebnisübernahme. Dazu haben wir eine Parametersemantik für Client/Server-Beziehungen definiert. Das RCS garantiert eine Übertragung der Parameter gemäß der angegebenen Semantik unabhängig von der zur Verfügung stehenden Hardware.

Aus Gründen der Portabilität ist das RCS in der Sprache C unter ausschließlicher Verwendung von Standardfunktionen des Betriebssystems UNIX (BSD 4.2) entworfen und implementiert /Hu89/. Im Rahmen des PRIMA-Projektes wird gegenwärtig ein über die drei Rechnerarten (Siemens, Sun, Apollo) verteiltes Anwendungssystem erprobt. Dabei laufen Systemkomponenten mit der Funktionalität des Zugriffs- und Speichersystems auf dem Siemens-Mainframe, Systemkomponenten für die Transaktions- und die Metadatenverwaltung sowie das Datensystem auf Sun-Rechnern und schließlich die DBS-Anwendungsebene und die PRIMA-Anwendung selbst auf Apollo-Workstations.

Nachdem die Funktionalität des RCS durch den Einsatz im PRIMA-Prototypsystem weitestgehend validiert wurde, ist es in weiteren Arbeiten geplant, die generelle Leistungsfähigkeit des RCS zu untersuchen und zu verbessern. Von besonderem Interesse ist dabei die Frage, in welchem Maß der evtl. vorhandene gemeinsame Speicher zur Effizienzsteigerung bei der Auftrags- und Parameterübertragung genutzt werden kann. Aus Sicht der PRIMA-Prototypentwicklung liegt das Augenmerk auf einer Bewertung der mit Hilfe des RCS umgesetzten DB-Verarbeitungsstrategien. Daher wurde das RCS um Werkzeuge zur Durchführung von Messungen und Analysen sowie zur Darstellung der Systemdynamik erweitert. Die bereits realisierten Analysewerkzeuge sollen es ermöglichen, Aussagen über den erzielten Paralleltätsgrad abzuleiten und die konkret durchgeführte Zerlegung von DB-Operationen in Suboperationen, insbesondere deren Granularität und Ausführungshäufigkeiten, zu bewerten /HKSS91/.

## 5. Literatur


AS83 Andrews, G.R., Schneider, F.B.: Concepts and Notations for Concurrent Programming, in: ACM Computer Surveys, Vol. 15, No. 1, March 1983.

CS87 Coffild, D., Shepard, D.: Tutorial Guide to Unix Sockets for Network Communications, in: Computer Communications, Vol. 10, No. 1, Feb. 1987.

En89 Encore Computer Cooperation: Multimax Technical Summary, 1989.

Hä88 Härder, T.: The PRIMA Project - Design and Implementation of a Non-Standard Database System, SFB-Bericht 26/88, Universität Kaiserslautern, 1988.

HHM86: Härder, T., Hübel, C., Mitschang, B.: Use of Inherent Parallelism in Database Operations, in: Proc. of the Conference on Algorithms and Hardware for Parallel Processing, Springer-Verlag, Lecture Notes in Computer Science, Vol. 237, 1986.

HKS90 Hübel, C., Käfer, W., Sutter, B.: Ein Client/Server-System als Basiskomponente für ein kooperierendes Datenbanksystem (Langfassung), SFB-Bericht 26/90, Universität Kaiserslautern, Mai 1990.

HKSS91 Hübel, C., Käfer, W., Schöning H., Sutter, B. : Leistungsbewertung in einem kooperierenden Datenbanksystem - Anforderungen, Methoden, Analysewerkzeuge, (in Vorbereitung) Universität Kaiserslautern, 1991.

HMMS88 Härder, T., Meyer-Wegener, K., Mitschang, B., Sikeler, A.: PRIMA - A DBMS Prototype Supporting Engineering Applications, in: Proceedings 13th International Conference on Very Large Databases, pp. 433-442, Brighton, 1987.

HSS88 Härder, T., Schöning, H., Sikeler, A.: Parallelism in Processing Queries on Complex Objects, in: Proc. of the International Symposium on Databases in Parallel and Distributed Systems, Austin, Texas, 1988.

HSS89 Härder, T., Schöning, H., Sikeler, A.: Evaluation of Hardware Architectures for Parallel Execution of Complex Database Operations, in: Proc. 3rd Annual Parallel Processing Symposium Fullerton, CA, USA 1989, pp 564-578.

Hu89 Huber, H.-P.: RCS - Ein Basissystem zur Realisierung verteilter Anwendungssysteme, Diplomarbeit, Universität Kaiserslautern, 1989.

Le89 Lee, P.-A.: Why Parallel Processing, Technical Report, Computing Laboratory, University of Newcastle upon Tyne, 1989.

Mi88 Mitschang, B. : Ein Molekül-Atom-Datenmodell für Non-Standard-Anwendungen - Anwendungsanalyse, Datenmodellentwurf und Implementierungsaspekte, Informatik-Fachberichte 185, Springer-Verlag, Berlin, 1988.

Schö90 Schöning, H.: Realisierung von Parallelität bei der Bearbeitung von Anfragen von komplexen Objekten, in: Härder, T., Wedekind, H., Zimmermann, G. (Hrsg): Entwurf und Betrieb verteilter Systeme, Informatik-Fachberichte, IFB264, Springer, September 1990.

SE86 Seifert, M., Eberle H.: Remote Service Call (RSC): A Network Operating System Kernel for Heterogenous Distributed Systems, in: NTG-Fachberichte Nr. 92, VDE-Verlag 1986.

Sun89 Sun Microsystems: verschiedene Benutzerhandbücher, 1989.

Sy88 Sequent Computer Systems, Inc.: Symmetry Technical Summary, 1988.