Prüfer:	Prof. Dr. A. Reuter
Betreuer:	Friedemann Schwenkreis

Beginn am:	15.12.1994
Ende am:	14.06.1995

CR-Nummer:	H.2.4, D.2.5

Studienarbeit-Nr. 1440

**Komponenten-
management
für
APRICOTS**

Anett Frank

Fakultät für Informatik
Institut für Parallele und verteilte Höchstleistungsrechner (IPVR)
Universität Stuttgart
Breitwiesenstraße 20-22
70567 Stuttgart

Einleitung

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

In immer mehr kommerziellen, wissenschaftlichen und kulturellen Bereichen werden Informationen in computer-basierten Informationssystemen gespeichert, verwaltet, abgefragt und aktualisiert. Um zuverlässig mit diesen Informationssystemen zu arbeiten, wurde in den 70er Jahren das Konzept der ACID-Transaktionen entwickelt. Eine Transaktion ist eine sequentielle Ausführungseinheit auf einem computer-basierten Informationssystem. Die Buchstaben A, C, I, D stehen für die folgende vier Eigenschaften:

- Atomicity
  Eine Transaktion wird entweder ganz oder gar nicht ausgeführt.

- Consistency
  Das Informationssystem enthält vor und nach der Transaktion keine Widersprüche.

- Isolation
  Die Transaktion läuft isoliert von anderen Transaktionen auf Teilen des Informationssystems ab.

- Durability
  Die Änderungen der Transaktion im Informationssystem müssen dauerhaft gespeichert sein.

Dieses Konzept eignet sich sehr gut für Anwendungen, deren Ausführungseinheiten klein sind, die wenig Datenzugriffe haben und nur eine kurze Zeit im System zubringen, wie z.B. für Lager- und Buchhaltungssysteme.
Für Anwendungen in großen, verteilten Systemen, in denen eventuell Migration möglich ist, für Systeme, die eine komplexe Flußkontrolle benötigen, viele Daten anfassen und zeitaufwendig sind, wie z.B. Steuerungsprozeße, CAD-Systeme, ist das klassische ACID-Transaktionskonzept unzureichend.

Beispiel:

In einem Reisebüro will ein Kunde eine Reise nach Los Angeles buchen. Es soll ein Flug einer bestimmten Fluggesellschaft, ein Zimmer eines beliebigen Hotels und ein Auto von irgendeiner Autoverleihfirma gebucht werden. Nur wenn alle drei Buchungen möglich sind, soll die Reise gebucht werden.
Die Reisekauffrau im Reisebüro informiert sich in einem computer-basierten Informationssystem über freie Kapazitäten der Fluggesellschaft. Es ist ein Platz frei. Dieser wird von ihr reserviert.
Das Hotel, indem die Reisekauffrau ein Zimmer reservieren will, ist ausgebucht. Sie muß versuchen in einem anderen Hotel ein Zimmer zu reservieren. Dies gelingt ihr dann auch.

Als letztes muß ein Auto gemietet werden. Doch keine Autoverleihfirma in Los Angeles verfügt über freie Kapazitäten. Kein Auto kann gemietet werden.
Da der Kunde die Reise nur buchen will, wenn alle drei Einzel-Buchungen möglich sind, muß die Flug- und Hotelreservierung rückgängig gemacht werden.

Im Beispiel bestimmt und steuert die Reisekauffrau die Reihenfolge der Aktionen, der Transaktionen. Es entstand die Idee diese Tätigkeiten zu automatisieren. Kurze Transaktionen in einen Kontrollfluß einzubetten und darin zu steuern.

In [Reut89] wird das Konzept des ConTracts entwickelt, das einen solchen Kontrollfluß ermöglicht. Dort wird der formale Hintergrund für die Definition und Steuerung langlebiger, komplexer Anwendungen aufgezeigt. Ein ConTract wird durch den Kontrollfluß (Skript) einzelner Aktivierungseinheiten (Steps) bezeichnet.
Dieses Programmiermodell garantiert Dauerhaftigkeit, Konsistenz, Recovery, Synchronisation und Kooperation.

1.1 Das ConTract-Modell

Einem ConTract (*Con*troled Activities Controled *Tr*ansa*ct*ion Compound) liegt das ConTract-Modell von [WäRe91] zu Grunde. Er wird durch ein *Skript* und den darin enthaltenen *Steps* (Aktionen) beschrieben.

> Definition eines ConTracts nach [WäRe91]:
>
> > Ein ConTract ist eine konsistente und fehlertolerante Ausführung einer beliebigen Sequenz vordefinierter Aktionen (*Steps*), gemäß einer expliziten spezifizierten Kontrollflußbeschreibung (*Skript*).

Steps sind einzelne oder in einer Sequenz zusammengefaßte Aktionen, z.B. Flugbuchung, Hotelbuchung.

Das *Skript* definiert den Kontrollfluß einer Anwendung in graphischer und in sprachlicher Form.

Die *graphische Darstellung* gleicht einem Petri-Netz. Es werden Teilschritte (Steps/ Stellen), Bedingungen (Prädikate/Conditions), Ereignisse (Events/Transitionen) und das Weiterleiten der Ereignisse (Triggern/Feuern) festgelegt. Die Transaktionsgrenzen können über einen Step oder über mehrere Steps bestimmt werden. Dies ist ganz dem Anwender überlassen.

Durch die *sprachliche Darstellung* erhält der Skript-Programmierer die Möglichkeit:

- einen Kontext für persistente Variablen zu definieren,
- Transaktionsgrenzen und -abhängigkeiten festzulegen,
- die Synchronisation parallel laufender ConTract-Steps durch Konsistenzprüfung zu ermöglichen,
- zu jedem Teilschritt eine Kompensationsaktion bereitzustellen, um bei Abbruch die Konsistenz zu gewährleisten.

1.2 APRICOTS

APRICOTS (**A PR**ototype **I**mplementation of a **ConTr**act **S**ystem) ist die Realisierung des ConTract-Systems. Es wird am IPVR der Universität Stuttgart entwickelt. Bei der Entwicklung und Realisierung von APRICOTS steht hohe Zuverlässigkeit, hohe Fehlertoleranz und leichte Erweiterbarkeit im Vordergrund.

In Abb. 1 auf Seite 9 sind die funktionalen Zusammenhänge von den Modulen *ConTractManager*, *StepServer*, *ResourceManager* und dem *TransactionManager* repräsentiert. Das *CommunicationSystem*-Modul sorgt für die zuverlässige Kommunikation zwischen den Modulen.

Der Benutzer kann mit dem in [Zell92] entwickelten *User-Interface* zum einen Komponenten statisch konfigurieren und zum anderen die dynamische Konfiguration der Komponenten zur Laufzeit des ConTracts überwachen.
Mit Hilfe des User-Interface definiert, aktualisiert und eliminiert der Benutzer statisch vor Laufzeit der Komponenten, die Dienste und ResourceManager der APRICOTS-Komponente.
Zur Laufzeit bietet das User-Interface dem Benutzer die Möglichkeit bei Ausfall oder Überlastung einzelner Komponenten sie neu zu starten, zu stoppen oder zu migrieren.

Der *ContractManager* (CM) ist für den korrekten und zuverlässigen Ablauf des im Skript definierten Kontrollflusses zuständig. Er initiiert die im Skript festgelegten Steps, indem er die StepServer aufruft, die die Aktionen implementieren.

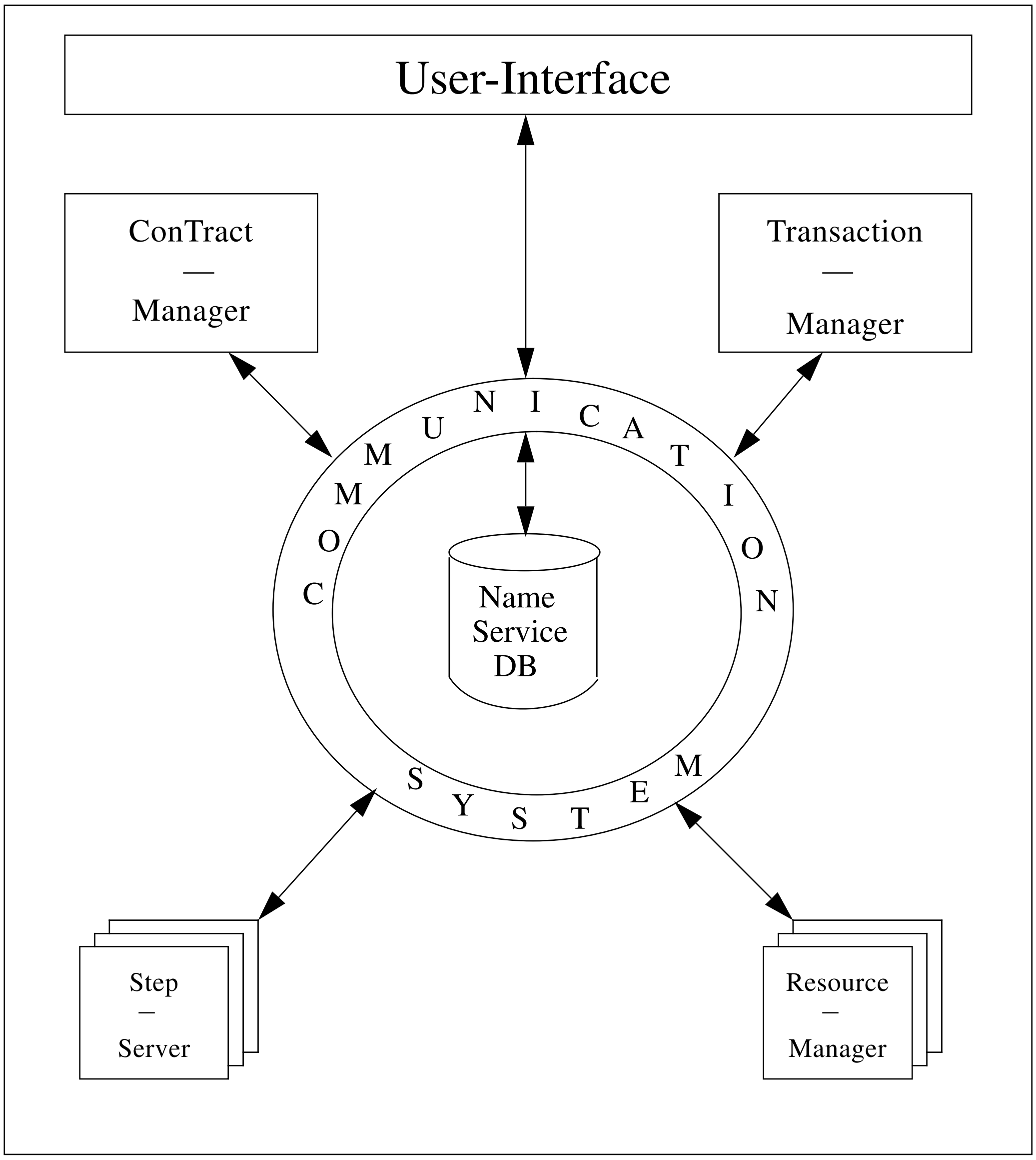

Abb. 1 Aktuelle Architektur von APRICOTS nach [Schü93]

Im *StepServer* (SS) sind die vom ConTractManager initiierten Aktionen implementiert. Der StepServer wiederum initiiert die ResourceManager.

Die *ResourceManager* (RM) verwalten die nötigen Ressourcen, wie z.B. Datenbanken, Roboter oder Drucker.

Der *TransactionManager* (TM) führt die Transaktionen nach dem „2-Phase-Commit-Protokoll“ aus. Er kontrolliert und verwaltet die im Skript definierten Beziehungen zwischen den Transaktionen.

Das *CommunicationSystem* (CS) stellt den Modulen Kommunikationsmechanismen zum Versenden und Empfangen von Nachrichten zur Verfügung.

1.3 APRICOTS-Komponente

In APRICOTS werden Teile der Modulfunktionalitäten des *ConTractManager*, *StepServer*, *ResourceManager* zu lauffähigen Programmen zusammengebunden und als *Komponente* bezeichnet, siehe Abb. 2 auf Seite 11. Jede Komponente läuft in einem Prozeß.

Die von Komponenten angebotene Funktionalität wird *Dienst* bzw. *service* genannt. Jede Komponente hält Dienstinformationen in einer prozeßlokalen Einsprungtabelle und im prozeßlokalen Cache.
In der prozeßlokalen Einsprungtabelle sind Dienstinformationen gespeichert, die die Komponente „selbst“ anbietet.
Im lokalen Cache befinden sich Informationen über die Dienste, die seit Komponentenstart von ihr adressiert wurden.

Alle Dienste einer Inselgruppe [Häup93] werden in einer relationalen Datenbank, dem *NameService*, persistent gespeichert. (Das Datenbanksystem garantiert die ACID-Eigenschaften der auf ihm stattfindenden Transaktionen.) Eine Inselgruppe ist ein Verbund von mehreren Rechnerknoten, die auf ein gemeinsamen stabilen Speicher zugreifen.

Dienste anderer Inselgruppen werden durch Aufruf eines NameServers adressierbar. Auf jedem Rechner gibt es mindestens einen *NameServer*, der Verbindung mit dem NameService „seiner“ Inselgruppe hat. Ein NameServer „kennt“ die NameServer anderer Inselgruppen, die Verbindung mit deren NameService haben. So können Komponenten alle Dienste, der im System vorhandenen Inselgruppen, adressieren, wenn die entsprechenden NameServer bekannt sind.

Den Komponenten steht über ein Verwaltungssystem eine Anzahl von Funktionalitäten zur Verfügung, wie z.B. Einfügen, Löschen, Aktualisieren einer Liste von Diensten, als auch Adressenrecherchen nach gesuchten Diensten in der Einsprungtabelle, im Cache und in NameService.

Um „Dienste“ anderen Komponenten zur Verfügung zu stellen und sie von anderen zu nutzen, erhält **jede** Komponente Teile des *CommunicationSystem*. Durch die Verteilung der Kommunikationsfunktionalitäten auf die Komponenten wird die Ausfallsicherheit des CommunicationSystems erhöht und damit für die Komponente zuverlässig.

Jeder Komponente, d.h. jedem Prozeß, wird eine Mailbox zugeordnet.
Eine Mailbox ist eine prozeßunabhängige, persistente Messagequeue. Die indirekte Kommunikation zwischen den Komponenten findet über diese Mailbox statt.

Mit Hilfe des CommunicationSystem können die Komponenten Nachrichten von anderen Komponenten über die Mailbox empfangen. Mit einem für APRICOTS entwickelten *resource_manager_call()* (RMC) werden die Dienste prozeßlokal, rechnerlokal und rechnerübergreifend aktiviert. Bei jedem *resource_manager_call()* wird die Empfängeradresse ermittelt, eine Berechtigungsprüfung durchgeführt und schließlich der Dienst initiiert.

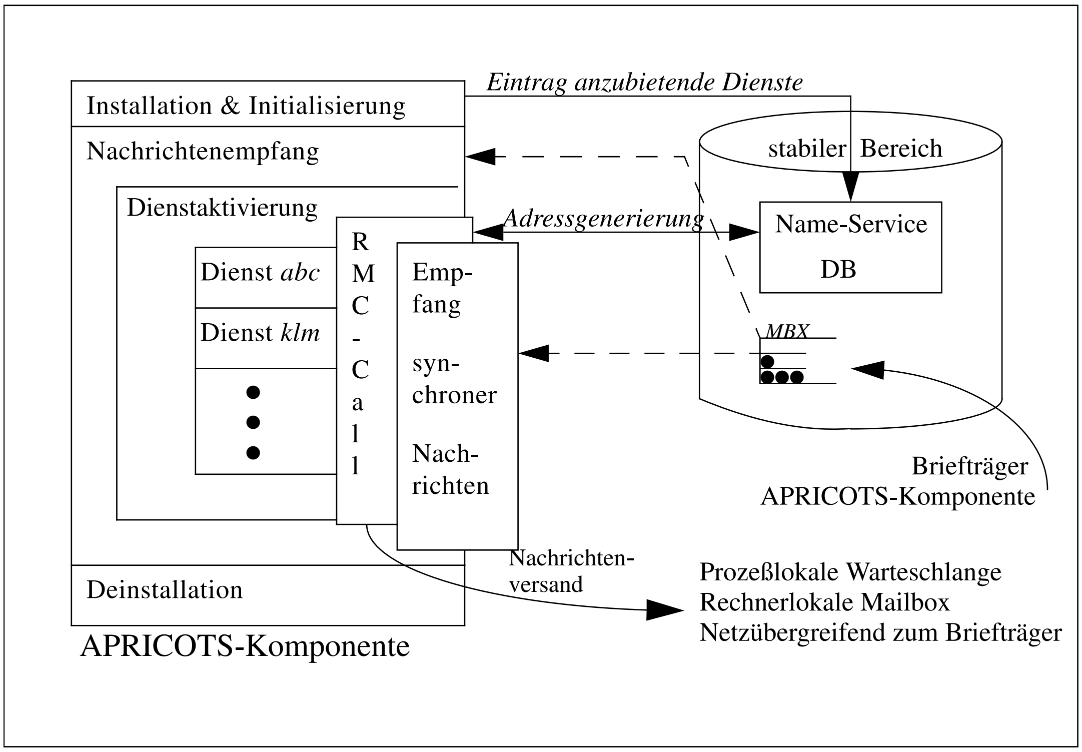

Abb. 2 APRICOTS-Komponente nach [Schü93]

1.4 Aufgabenstellung der Studienarbeit

APRICOTS soll ein zuverlässiges, fehlertolerantes und ausfallsicheres System sein.
Da Systemfehler (Leitungsbruch, Rechnerausfall und Prozeßabsturz) unausweichlich sind, sollen sie zuverlässig durch ein Überwachungsmodul erkannt und entsprechend auf sie reagiert werden.
Der Benutzer des Systems ist dann in der Lage entsprechend darauf zu reagieren.

Ziel dieser Arbeit ist: Ein Konzept für die Fehlererkennung und -behandlung in APRICOTS zu entwickeln und zu realisieren.

2 Kommunikation in APRICOTS

Mit der Kommunikationsschnittstelle *resource_manager_call()* (RMC) ist es jeder Komponente möglich, Dienste anderer Komponenten synchron oder asynchron zu initiieren.

Der *synchrone RMC* ähnelt dem remote procedure call (RPC) client/server-basierter Systeme. Die auftraggebende Komponente (Client) initiiert bei der anbietenden Komponente (Server) den Dienst und wartet auf die Antwort. Da einige Komponenten, wie z.B. der StepServer, sowohl die Rolle des Clients (Dienstinitiierung der ResourceManager) als auch die Rolle des Servers (Dienstbearbeitung für den ContractManager) einnehmen, konnte der RPC im Prototyp nicht verwendet werden. Deswegen wurde in [Laub92] der *resource_manager_call*() eingeführt, der es möglich macht, daß die Komponenten beide Rollen einnehmen.
Es wird in diesem Zusammenhang nicht von Client und Server gesprochen, sondern von *Auftraggeber* (AG) und *Auftragnehmer* (AN).

Der *asynchrone RMC* erlaubt dem Auftraggeber nach der Dienstabsetzung die Fortführung seiner Arbeit. So wird es möglich Dienste parallel auszuführen.

Bei jedem Dienstaufruf, mittels RMC, bestimmt das Kommunikationssystem die Empfehlungsadresse, führt die Berechtigungsprüfung durch und aktiviert den aufgerufenen Dienst beim Aufragnehmer.

Weitere Unterschiede gegenüber dem RPC sind:

- Transparente Dienstadressierung
- Indirekte Kommunikation durch den Mailboxmechanismus. Die indirekte Kommunikation erlaubt den Komponenten auch während des WAIT-Modus‘ Nachrichten zu empfangen.
- Stornieren der unbearbeiteten Nachricht aus der Mailbox.

Für die Nachrichtenübermittlung werden drei unterschiedlichen Kommunikationsmechanismen verwendet. Entsprechend wo der Dienst angeboten wird, wählt das Kommunikationssystem einen anderen Kommunikationsmechanismus.
Wird der Dienst prozeßlokal angeboten, wird dis Nachricht *prozeßlokal* übermittelt.
Wird der Dienst von einer Komponente auf demselben Rechner angeboten, wird die Nachricht *rechnerlokal* übermittelt.
Wird der Dienst auf einem anderen Rechner angeboten, wird die Nachricht *rechnerübergreifend* übermittelt.

2.1 Dienstaktivierung

Den Komponenten stehen zwei Arten der Dienstaktivierung zur Verfügung:

- der *synchrone* Dienstaufruf *(RM_call_sync())*
- der *asynchrone* Dienstaufruf *(RM_call_async())*

2.1.1 Synchroner RMC

Der Aufraggeber eines *synchronen* Dienstaufrufes wartet solange auf die Antwort des Auftragnehmers. Das heißt: die Komponente ist blockiert und bearbeitet keine anderen Aufträge.

Aufträge anderer Komponenten können während dessen in der Mailbox empfangen werden.

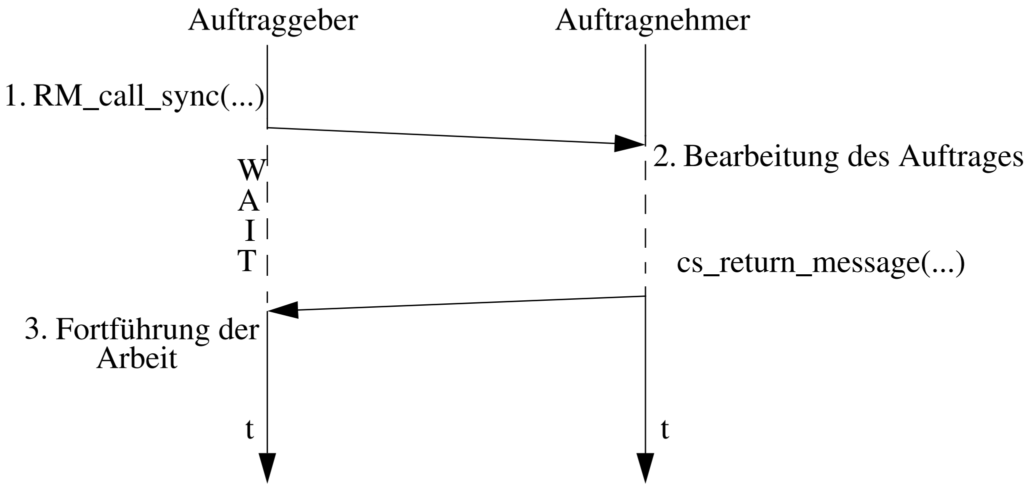

Abb. 3 Ablauf eines synchronen RMC

In Abbildung 3 ist der Ablauf eines synchronen RMC in drei Schritten dargestellt:

1. Der synchrone RMC wird vom Auftraggeber mit der Funktion *RM_call_sync()* aufgerufen.
   Nach der Empfängerermittlung und der Berechtigungsprüfung versendet der Auftraggeber die Nachricht an die Auftragnehmerkomponente.

Der Auftraggeber geht in den Warte-Modus.
Wenn lange Zeit keine Antwort vom Auftragnehmer kommt, wird der Auftraggeber durch einen Timeout unterbrochen. Der Auftraggeber hat dann die Möglichkeit die Nachricht aus der Mailbox des Auftragnehmers zu entfernen und den RMC abzubrechen.

2. Der Auftragnehmer bearbeitet den synchronen RMC und schickt die Antwort mit der Funktion *cs_return_message()* an den Auftraggeber zurück.
3. Sobald die Antwort eintrifft, setzt der Auftraggeber seine Arbeit fort.

Der synchrone RMC ermöglicht die sequentielle Initiierung der ResourceManager und die zuverlässige Kommunikation mit dem TransactionManager.

2.1.2 Asynchroner RMC

Der Aufraggeber eines *asynchronen* Dienstaufrufs wartet nach der Dienstinitiierung nicht bis die Antwort eintrifft, sondern bearbeitet die vorhandenen Aufträge weiter.

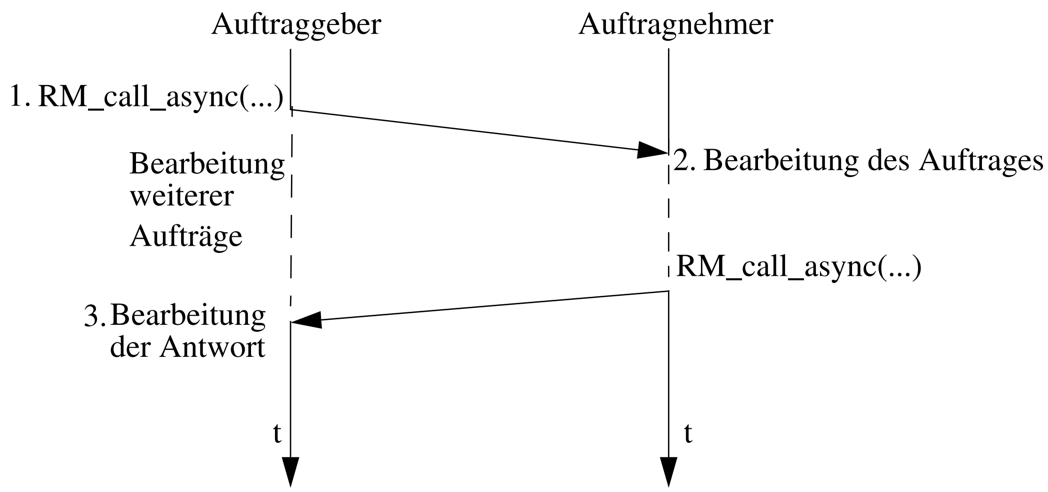

Abb. 4 Ablauf eines asynchronen RMC

In Abbildung 4 ist der Ablauf eines asynchronen RMC in drei Schritten dargestellt:

1. Der asynchronen RMC wird mit der Funktion RM_call_async() aufgerufen.
   Der Aufraggeber wartet nach dem Versenden des Auftrages nicht auf die Antwort des Auftragnehmers, sondern setzt seine Arbeit fort.

2. Der Auftragnehmer bearbeitet den Auftrag und schickt die Antwort mit einem asynchronen RMC zum Auftraggeber zurück.

3. Der Arbeitgeber nimmt die Nachricht entgegen und bearbeitet sie.

Mit dem asynchronen RMC läßt sich die im ConTract-Modell gewünschte Parallelität der Aktionen realisieren.

2.2 Adressenermittlung

Die Dienstadresse wird nach einem dreistufigen Scheduling-Verfahren vom Auftraggeber ermittelt. Findet sich die gesuchte Adresse nicht prozeßlokal, wird rechnerlokal im NameService der Inselgruppe und dann im entfernten NameService gesucht. Die Beschreibung der Einsprungtabelle, des Caches und des NameService findet sich in Kapitel 3.

- Prozeßlokal

  In der Einsprungtabelle und im lokalen Cache der Komponente wird nach der Dienstadresse gesucht.
  Die Einsprungtabelle verwaltet Dienstadressen, die von der Komponente selbst angeboten werden.
  Im lokalen Cache befinden sich Dienstadressen aus dem lokalen und dem entfernten NameService, die seit Start der Komponente adressiert wurden.

- NameService der Inselgruppe

  Der lokale NameService enthält alle Dienstadressen, die in der Inselgruppe verfügbar sind. Eine Inselgruppe ist ein Verbund von Rechnerknoten, die auf ein gemeinsamen stabilen Speicher, dem NameService, zugreifen.

- Entfernter NameService

  Den Komponenten einer Inselgruppe steht mindestens ein Name-Server, der NameServices anderer Inselgruppen kennt, zur Verfügung. Dadurch können Adressenrecherchen auch auf diesen NameServices durchgeführt werden.
  Dienstdaten anderer Inselgruppen werden bei Adressierung in den lokalen NameService eingetragen.

Nach der Ermittlung der Dienstadresse prüft der Auftraggeber vor dem Senden der Nachricht, ob er berechtigt ist den Dienst beim Auftragnehmer zu initiieren. Der Auftragnehmer prüft nach Empfang der Nachricht, ob der Auftraggeber berechtigt ist bei ihm den Dienst aufzurufen.

2.3 „Distanzabhängige“ Nachrichtenübermittlung

In APRICOTS werden drei Kommunikationsmechanismen verwendet. Der prozeßlokale, rechnerlokale und rechnerübergreifende Kommunikationsmechanismus. Mit dem prozeßlokalen Kommunikationsmechanismus wird eine Funktion aufgerufen. Rechnerlokal wird eine Nachricht innerhalb des selben Rechners übermittelt und rechnerübergreifend wird der Briefträger auf dem anderen Rechner beauftragt die Nachricht rechnerlokal zu übermitteln.

2.3.1 Prozeßlokale Kommunikation

Der gewünschte Dienst wird vom Auftraggeber selbst angeboten. Die Auftraggeberadresse ist mit der Dienstadresse identisch. Auftraggeber und Auftragnehmer befinden sich im selben Prozeßraum.

Der Dienstaufruf entspricht einem *Funktionsaufruf*.

In Abbildung 5 ist die prozeßlokale Dienstaktivierung dargestellt.

Komponente A ruft prozeßlokal die Funktion „Dienst xyz“ auf.

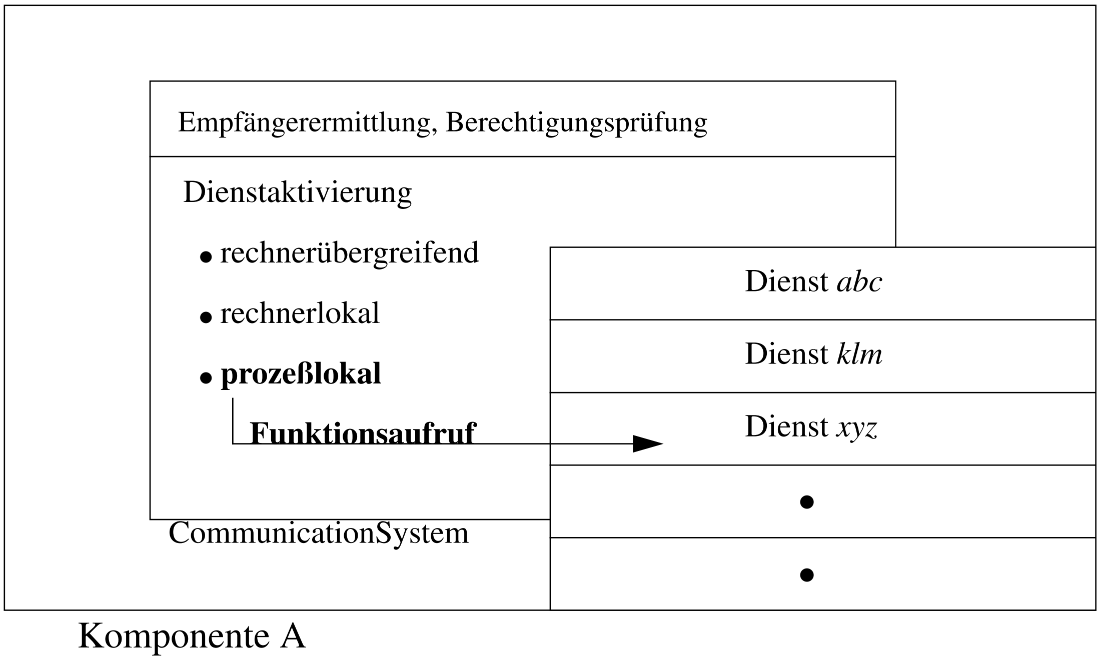

Abb. 5 Prozeßlokale Dienstaktivierung

Es ist vorgesehen, daß jeweils ein Dienst von einem „Thread“ bearbeitet wird. Ein Thread ist eine Unterklasse eines Prozesses. Ein Thread verfügt lediglich über einen eigenen Registersatz (Stack) und einem Programmzähler. Der Daten- und Programmspeicher ist für alle Threads in einem Prozeß gleich. Jeder Prozeß hat mindestens einen Thread. Bei einem *thr_create*-Aufruf wird nicht wie bei einem *fork*-Aufruf ein neuer Speicherplatz im Hauptspeicher erzeugt, sondern nur ein Stack im Prozeß für den Thread erzeugt. Man spricht auch von leichtgewichtigen Prozessen („light-weight process“).

Vorteile von Threads: Sie sparen:

- Zeit für die Erzeugung der notwendigen Prozeßdaten
- Speicherplatz für redundante Daten und
- Kommunikation zwischen den Prozessen

Es gibt noch keinen endgültigen Standard für eine Thread-Schnittstelle zu C-Programmen, aber im POSIX-Standard P1003.1c (früher 4a) Draft 10 unter „Portable Operating System Interface for Computer Environments, Threads Extensions“ sind bereits die wesentlichen und notwendigen Techniken festgelegt. Die Realisierungen dieser Techniken unterscheiden sich noch stark, wodurch die Portabilität solcher Thread-Techniken stark eingeschränkt wird.

2.3.2 Rechnerlokale Kommunikation

Der gewünschte Dienst wird von einer Komponente angeboten, die sich auf dem gleichen Rechner wie die aufzurufende Komponente befindet. Die Kommunikation findet indirekt zwischen der Komponente und der Mailbox statt.

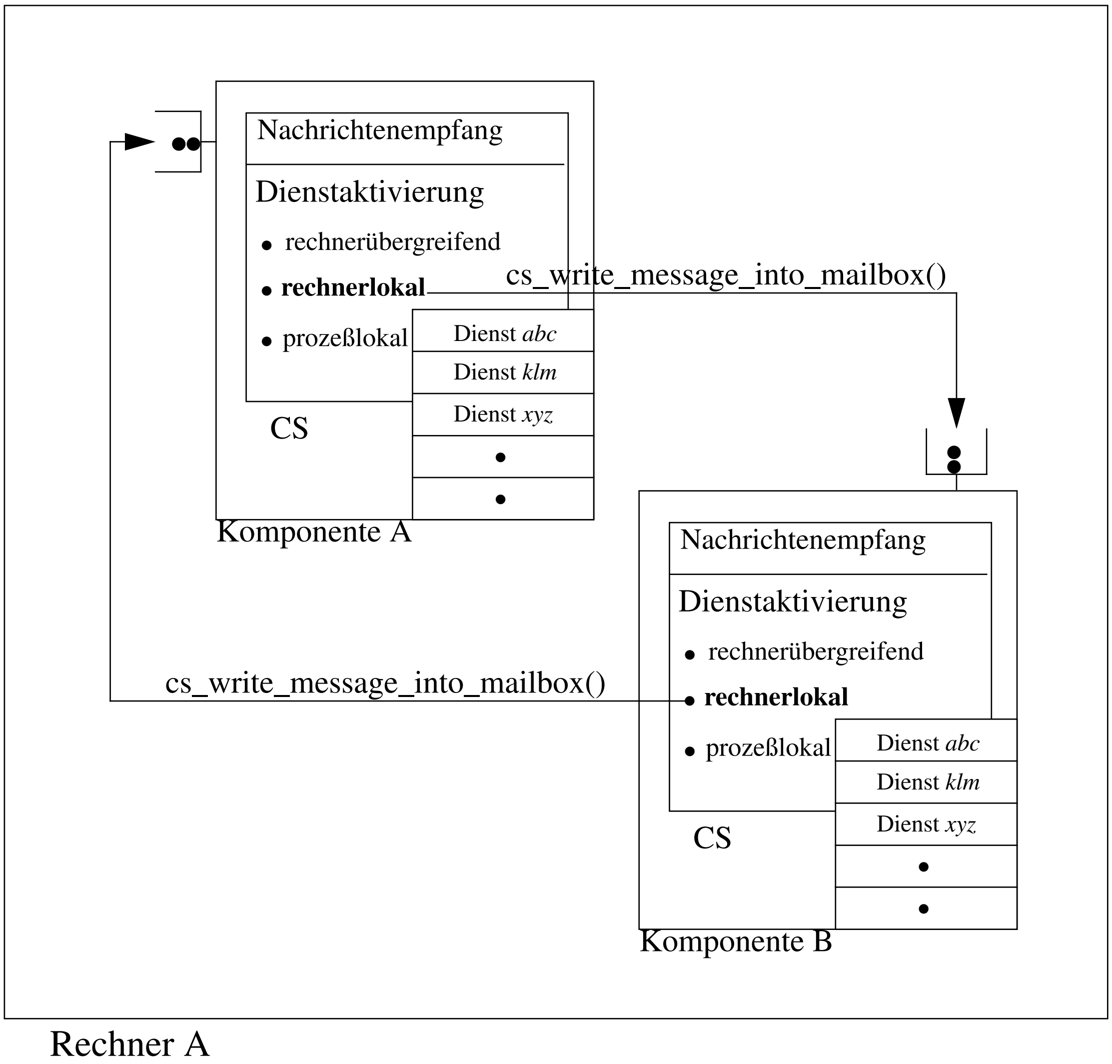

Abb. 6 Rechnerlokale Dienstaktivierung

In Abbildung 6 wird die rechnerübergreifende Kommunikation dargestellt. Komponente A ruft einen Dienst auf, den Komponente B anbietet.
Mit der Mailboxverwaltungsroutine *cs_write_message_into_mailbox()* wird die Nachricht in die Mailbox von Komponente B geschrieben.
Komponente B empfängt, bearbeitet und löscht die Nachricht aus der Mailbox.
Nach der Dienstbearbeitung schickt Komponente B die Antwort an die Mailbox von Komponente A.
Diese empfängt die Antwort und bearbeitet sie.

2.3.3 Rechnerübergreifende Kommunikation

Der gewünschte Dienst wird von einer Komponente, die auf einem anderen Rechner läuft, angeboten.

In [Häup93] wurde der Kommunikationsmechanismus für den rechnerübergreifenden Betrieb entwickelt und realisiert. Jeder Rechner erhält einen Briefträger, der ankommende Nachrichten in die Empfängermailbox der Komponenten schreibt.

Die rechnerübergreifende Dienstübermittlung ist in Abbildung 7 dargestellt.

Komponente A auf Rechner A will Komponente B auf Rechner B eine Nachricht übermitteln. Komponente A verwendet die Funktion *remote_mesage(SEND,...),* um die Nachricht vom Briefträger auf dem Rechner B in die Mailbox der Komponente B zu schreiben.
Der Briefträger von Rechner B empfängt den Auftrag vom Komponente A und schreibt die Nachricht mit der Funktion *cs_write_message_into_mailbox*() in die Mailbox der Komponente B.

Komponente B schickt ihre Antwort an den Briefträger auf Rechner A, der sie dann in die Mailbox von Komponente A schreibt.

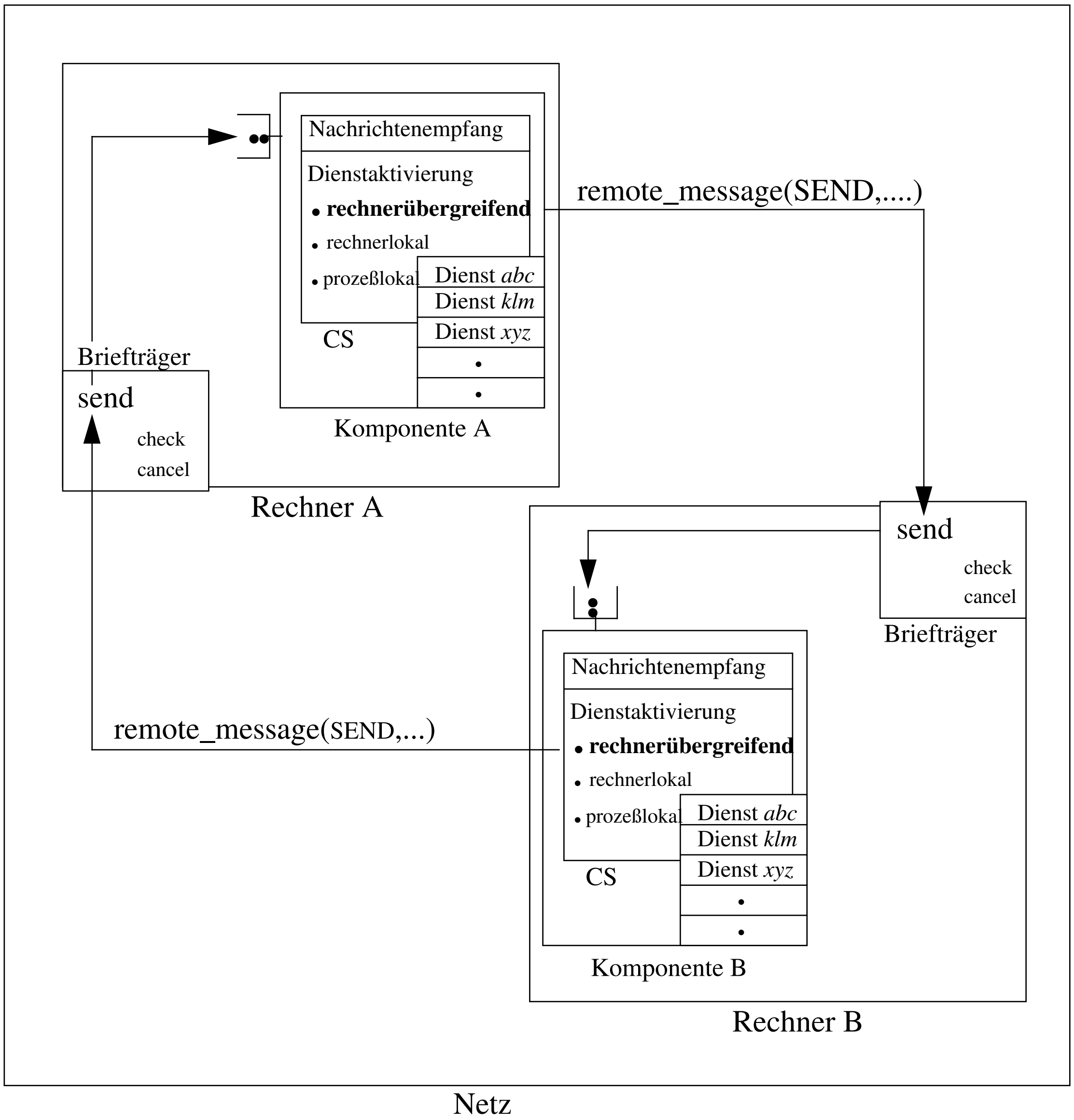

Abb. 7 Rechnerübergreifende Dienstaktivierung

Durch die Kombination dieser beiden Kommunikationsarten ist die zuverlässige rechnerübergreifende Dienstaktivierung sichergestellt. Die aufrufende Komponente erhält vom Briefträger eine Bestätigung (*Acknowledgement*), ob das Schreiben in die Mailbox erfolgreich war. Im erfolgreichen Fall steht die Nachricht auf jeden Fall in der Mailbox. Wenn zu lange[1] kein Acknowledgement bei der aufrufenden Komponente eintrifft, wird die Dienstinitiierung abgebrochen.

Die Kommunikation der Komponenten mit dem Briefträger ist verbindungsorientiert und basiert auf dem TCP/IP-Protokoll der Internet-Protokolle. Die Briefträger-Kommunikation mit der Mailbox basiert auf dem Mailbox-Mechanismus der rechnerlokalen Kommunikation, die in Abschnitt 2.3.2 auf Seite 19 beschrieben wird.

1. In [Häupt93] wurde 1s gewählt.

3 Basismechanismen des Kommunikationssystems

Informationen über Dienste werden in der Einsprungtabelle und im Cache der Komponente und in der NameService-Datenbank der Inselgruppe gespeichert.
Die Einsprungtabelle enthält Adress- und Zustandsinformationen über die von der Komponente angebotenen Dienste.
Im Cache jeder Komponente sind Adress- und Zustandsinformationen über die seit Start der Komponente adressierten Dienste zu finden.
In der NameService-Datenbank werden alle Dienstinformationen der Komponenten einer Inselgruppe persistent gespeichert. (Eine Inselgruppe ist ein Verbund von mehreren Rechnerknoten, die auf ein gemeinsames Dateisystem zugreifen.)

Um Dienste adressieren, aktualisieren, einfügen und löschen zu können, verfügt die Komponente über Verwaltungsroutinen.

Die Datenstrukturen und die Verwaltungsroutinen der Einsprungtabelle, des Caches und des NameService werden in Abschnitt 3.1 , Abschnitt 3.2 und in Abschnitt 3.3 erläutert.

Für die zuverlässige rechnerlokale und -übergreifende Kommunikation wurde der Mailbox- und Briefträgermechanismus eingeführt.
Kommuniziert eine Komponente mit einer anderen, muß sie die zu versendende Nachricht in die Mailbox der aufzurufenden Komponente geschrieben werden. Welche Funktionen den Komponenten hierfür zur Verfügung stehen, wird in Abschnitt 3.4 erläutert.
Erfolgt die Kommunikation rechnerübergreifend, muß ein Briefträger beauftragt werden die Nachricht in die Mailbox zu schreiben. Die Funktionalitäten, die es dafür gibt, werden in Abschnitt 3.5 beschrieben.

3.1 Prozeßlokale Einsprungtabelle

Informationen über die prozeßlokal angebotenen Dienste werden in einer prozeßlokalen Einsprungtabelle gehalten. Die Dienstdaten in der Einsprungtabelle werden statisch vor Laufzeit der Komponente konfiguriert. Zur Laufzeit der Komponente werden keine Dienstdaten hinzugefügt oder gelöscht. Die Komponenten haben die Möglichkeit zur Laufzeit Adressenrecherchen durchzuführen und die Verfügbarkeit des Dienstes zu aktualisieren.

Jeder angebotene Dienst entspricht einem Tabelleneintrag mit folgenden Informationen:

- dem Dienstnamen
  (*service_name*)
- der ResourceManager-Klasse
  (*resource_manager_class*)
- dem Komponentenname
  (*component_name*)
- der Aufrufadresse
  (*service_address*)
- die Verfügbarkeit des Dienstes
  (binäres *Validflag*)

Folgende Funktionalitäten stehen der Komponente durch die darunterstehenden Funktionen zur Verwaltung der Einsprungtabelle zur Verfügung:

- Kennzeichnung der Verfügbarkeit eines Dienstes.
  (cs_insert_into_function_table() markiert den Dienst als verfügbar;
  cs_delete_from_function_table() markiert den Dienst als nicht verfügbar)
- Ermittlung einer Liste von passenden Dienstadressen.
  (cs_lookup_function_table())
- Ermittlung der Dienstadresse eines bestimmten Services.
  (cs_get_function_address())

3.2 Der lokale Cache

Das CommunicationSystem recherchiert im lokalen Cache nach einer Dienstadresse, wenn der Dienst nicht prozeßlokal angeboten wird. Der lokale Cache enthält Dienstinformationen, die seit Komponentenstart über den NameService extrahiert wurden. Der Cache dient als schneller Zugriffspeicher.

Der Cache besteht aus einem Cache-Henkel und einem Cache-Bereich. Der Cache-Henkel speichert statistische Daten über den Cache.
Der Cache-Bereich teilt sich in n-1 Cache-Elemente. Ein Cache-Element entspricht einem Diensteintrag. Die detaillierte Cache-Struktur wird in [Schü93] beschrieben.

Jedes Cache-Element enthält folgende Daten mit den nachfolgenden Variablennamen:

- den Dienstnamen
  (*service_name*)
- die ResourceManager_Klasse
  (*rm_class_name*)
- den Komponentennamen
  (*component*)
- die Prozeßidentifikation
  (*process_id*)
- den Rechnername, auf dem die Komponente läuft
  (*hostname*)
- den Komponenten- bzw. Prozeßzustand
  (*component_state*)
- die Mailboxadresse
  (*mailbox_address*)

Zur Verwaltung des Caches stehen der Komponente folgende Funktionalitäten zur Verfügung:

- Installieren und Initialisieren des Caches.
  (cs_local_cache_install())
- Einfügen einer Liste von Diensten.
  (cs_local_cache_replace())
- Einfügen eines Cache-Eintrages.
  (cs_local_cache_overwrite())
- Ermittlung einer Liste von passenden Dienstadressen.
  (cs_local_cache_lookup())
- Ermittlung der Dienstadresse.
  (cs_local_cache_get_address())
- Löschen eines Cache-Eintrages.
  (cs_local_cache_delete())

- Aktualisieren eines bestimmten Cache-Elements.
  (cs_local_cache_update())

- Ersetzung der Cache-Elemente.
  (*cs_local_cache_replace_fifo()* ersetzt die Cache-Einträge nach dem FIFO-Prinzip;
  cs_local_cache_replace_clock1() ersetzt die Cache-Einträge nach dem „Second Chance“-Algorithmus.)

3.3 Der NameService

In der NameService-Datenbank sind alle Informationen über die in einer Inselgruppe verfügbaren Dienste gespeichert. Für jeden Dienst ist eine Relation Service_Name mit folgenden Attributen angelegt:

- Dienstname
  (*service_name*)

- ResourceManager-Klasse
  (*RM_Class_Name*)

- Komponentenname
  (*Log_Comp_Name*)

- Prozeßidentifikation
  (*Process_id*)

- Rechnername, auf dem die Komponente läuft
  (*Hostname*)

- Komponenten- bzw. Prozeßzustand
  (*State_of_Component*)

- Verweis auf Quellcodes und Definitionen des Dienstes
  (*Config_Reposit_Entry*)

- Mailboxadresse
  (*Mailbox_Id*)

Folgende Funktionalitäten mit den danach genannten Funktionsaufrufen stehen den Komponenten zur Verwaltung des NameServices zur Verfügung:

- Verbindung zum NameService auf- und abbauen.
  (cs_name_service_connect(); cs_name_service_disconnect())

- Ermittlung aller Adressen zu einem Dienst.
  (cs_name_service_lookup())

- Ermittlung der Dienstadresse.
  (cs_name_service_get_address())

- Einfügen einer Liste von Diensten.
  (cs_name_service_insert_services())

- Löschen einer Liste von Diensten.
  (*cs_name_service_delete_services()*)

- Aktualisieren einer Liste von Services.
  (cs_name_service_update_services())

Zugriff auf Dienstdaten andere NameServices erhält das Kommunikationssystem über den Dialog mit einem NameServer, der andere NameServer innerhalb anderer Inselgruppen kennt. Der NameServer der anderen Inselgruppe wird beauftragt eine Liste von gesuchten Diensten zu ermitteln. Diese wird in den lokalen NameService eingefügt.

3.4 Die Mailbox

Damit die Komponenten unabhängig von ihrer eigenen Tätigkeit (Dienstaktivierung, Dienstausführung, usw.) Aufträge von anderen Komponenten empfangen können, wurde in APRICOTS die indirekte Kommunikation mittels einem Mailbox-Mechanismus gewählt.

Der in [Laub92] verwendete Mailbox-Mechanismus, basierend auf dem Shared-Memory-Bereich, wurde in [Schü93] mit dem Message-Queue-Mechanismus unter POSIX Stufe 4 reimplementiert.

Die Vorteile gegenüber dem in [Laub92] vorgestellten Mechanismus sind:

- Die Bereitstellung von Message-Queue Primitiven für die Prozeßsynchronisation von lesenden und schreibenden Prozessen.

- Die permanente Kreation der Message-Queues, d.h. sie können auf stabilem Speicher kreiert werden und stehen somit auch nach einem Prozeßabsturz und Rechnerausfall den Komponenten zur Verfügung.

- Der POSIX-Standard Stufe 4

Jeder Komponente stehen die unten genannten Funktionalitäten mit den dazugehörigen Funktionsaufrufen für die Mailbox-Kommunikation zur Verfügung:

- Die Mailbox installieren.
  (cs_install_mailbox())

- Verbindung zur Mailbox auf- und abbauen.
  (cs_connect_mailbox(); cs_disconnect_mailbox())

- Die Nachricht in die Mailbox schreiben.
  (cs_write_message_into_mailbox())

- Die Nachricht aus der Mailbox lesen und sie dabei zu entfernen.
  (cs_read_message_out_of_mailbox())

- Die Nachricht lesen, ohne daß sie aus der Mailbox gelöscht wird.
  (cs_read_message_from_mailbox())

- Die Nachricht aus der Mailbox zu entfernen.
  (cs_delete_message_from_mailbox())

- Den Status der Nachricht in der Mailbox bestimmen.
  (cs_inquire_message_status())

3.5 Der Briefträger

Komponenten in APRICOTS, die rechnerübergreifend kommunizieren, müssen einen Übermittler, den Briefträger, benutzen. Auf jedem Rechner läuft ein Briefträgerserver, der von den Komponenten über einen bestimmten Port angesprochen wird.

Die Komponenten beauftragen mit der Funktion remote_message(OPTION,.....) den Briefträger. Mit der OPTION wird die Funktionalität des Briefträgers gewählt. Die drei Funktionalitäten und die dazugehörigen Optionen sind:

- Die Nachricht in die Empfänger-Mailbox schreiben.
  (SEND)

- Die Nachricht aus der Empfänger-Mailbox entfernen.
  (CANCEL)

- Den Zustand (bearbeitet/nicht bearbeitet) der Nachricht in der Empfänger-Mailbox überprüfen.
  (CHECK)

4 Fehlererkennung

Das APRICOT-System soll in höchstem Maße zuverlässig und fehlertolerant arbeiten. Dafür ist es notwendig, die im System vorkommenden Fehler zu erkennen und entsprechend auf sie zu reagieren.

In diesem Kapitel werden mögliche Fehler in APRICOTS eingeführt und ein Konzept zur Fehlererkennung vorgestellt. In Kapitel 5 „Fehlerbehandlung“ werden die notwendigen Maßnahmen auf die Fehler erläutert.

Der Idee dieser Fehlererkennung liegt zu Grunde, daß funktionsunfähige Dienste, Komponenten, Rechner und Netze nicht festgestellt werden müssen, wenn sie nicht verwendet werden. Verwendet werden die Komponenten, wenn sie für andere Komponenten Dienste ausführen, d.h. wenn mit ihnen kommuniziert wird.

Das Konzept beruht auf den drei Kommunikationsarten in APRICOTS, prozeßlokal, rechnerlokal, rechnerübergreifend. Da jeder Komponente diese Kommunikationsarten zur Verfügung stehen, ist jede Komponente in der Lage, Fehler zu erkennen und auf sie zu reagieren.

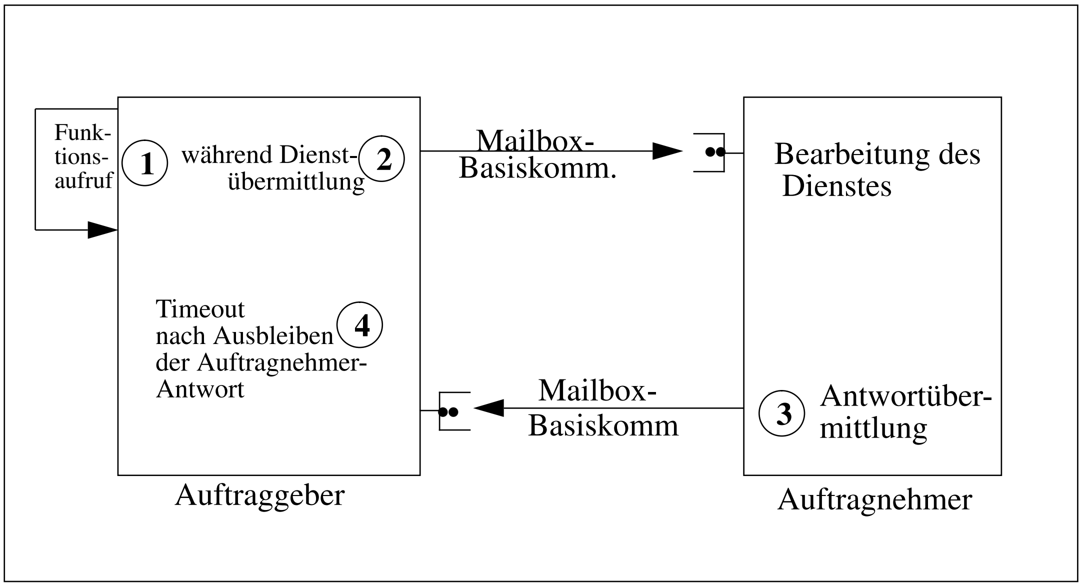

Abb. 8 Prozeß- und rechnerlokale Fehlererkennung

In Abbildung 8 und in Abbildung 9 sind die Zeitpunkte der Fehlererkennung während den drei Kommunikationsarten als Punkte dargestellt. Im Abschnitt 4.1 werden zunächst die vorkommenden Fehlerarten und die Erkennungskriterien zu den unterschiedlichen Zeitpunkten beschrieben, um dann in Abschnitt 4.2 zu erläutern, wie die Fehler erkannt werden.

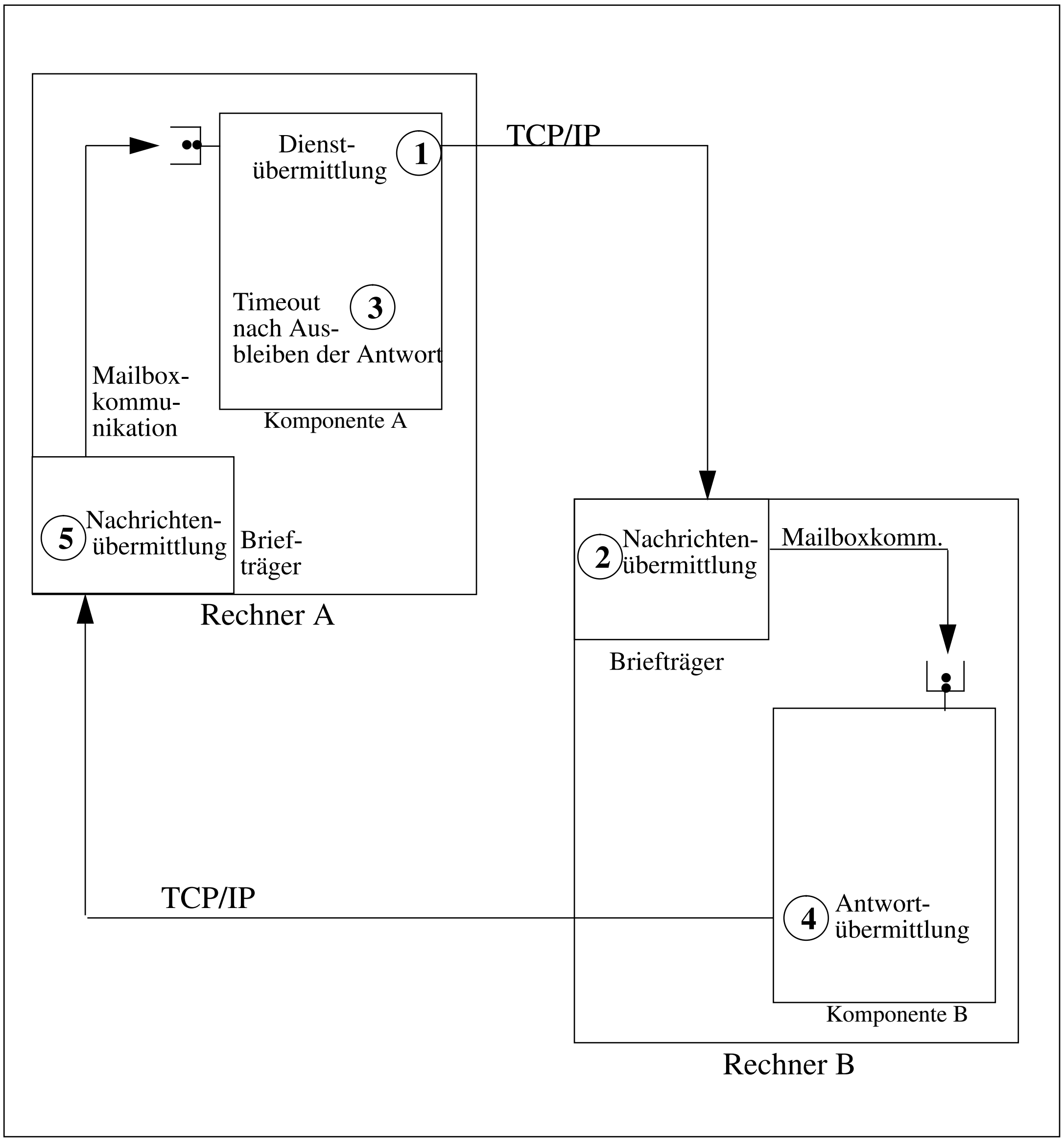

Abb. 9 Rechnerübergreifende Fehlererkennung

4.1 Klassifikation der Fehler

Die Fehler, die in dieser Ausarbeitung vorgestellt werden, betreffen zum einen Komponentenfehler (in APRICOTS entspricht eine Komponente einem Prozeß) und zum anderen Kommunikationsfehler. Beide Fehlertypen werden von den Kommunikationspartnern während der Kommunikation festgestellt.

4.1.1 Fehlerarten

Im Verlauf einer ConTract-Verarbeitungwerden in APRICOTS folgende Fehlertypen unterschieden:

- Komponentenfehler:
  - Prozeßfehler

    Die Komponente existiert nicht, d.h. der Komponente steht kein Speicherplatz auf dem Rechner zur Verfügung.
    Dienste, die sie im NameService anbietet, können von ihr nicht bearbeitet werden.

  - Dienstfehler

    Der Dienst einer Komponente stellt die Fehlerquelle dar. Der prozeßlokale Dienstaufruf entspricht einem Funktionsaufruf. Wird ein Dienst mit einer falschen Adresse oder mit falscher Anzahl von Argumenten aufgerufen, stürzt der Prozeß ab. Tritt ein Fehler (z.B. Endlosschleife) in der Dienstfunktion auf, ist die Komponente funktionsunfähig.

- Kommunikationsfehler:
  - Verbindungsfehler mit der Mailbox und mit dem Rechner

    Bei der *rechnerlokalen* Kommunikation tritt der Verbindungsfehler während dem Verbindungsaufbau zur Mailbox auf.
    Die Kommunikation mit der Mailbox ist eine indirekte Interprozeßkommunikation und ist in der Sitzungsschicht (Schicht 5 des OSI-7 Schichten-Modells) realisiert.

    Bei der *rechnerübergreifenden* Kommunikation tritt der Verbindungsfehler mit dem Rechner, d.h. mit dem Briefträger auf. Gründe für einen Verbindungsfehler sind Leitungsbruch und Fehler des Briefträgers.
    Die Kommunikation mit dem Briefträger wird in der Transportschicht (Schicht 4 des OSI-7 Schichten-Modells) realisiert.

  - Mailboxfehler

    Ein Mailboxfehler tritt auf, wenn die Nachricht nicht in Mailbox geschrieben werden kann.

Dies kann der Fall sein, wenn

- die Mailboxadresse ungültig ist.
- die Mailbox nicht existiert.
- die Mailbox voll ist.

- Rechnerfehler

  Ein Rechnerfehler entsteht, wenn der Rechner nicht betriebsbereit ist. Dies kann z.B. der Fall sein, wenn er wegen Reparatur vom Netz genommen wurde oder er ausgeschalten ist.

4.1.2 Erkennungskriterien

Fehler kommen in der Komponente, im Rechner und im Netz vor. Die Komponenten und der Briefträger erkennen abhängig von der *Kommunikationsart* Komponenten- und Kommunikationsfehler.

Der Auftraggeber stellt während der *prozeßlokalen* Dienstinitiierung (Funktionsaufruf), dargestellt in Abb. 8 auf Seite 29 Punkt 1, einen *Dienstfehler* fest.

Bei der *rechnerlokalen* Kommunikation erkennt der Auftraggeber und der Auftragnehmer während der Nachrichten- und Antwortübermittlung, in Abbildung 8, Punkt 2 und 3, Prozeßfehler, Verbindungsfehler mit der Mailbox und Mailboxfehler.

Bei der *rechnerübergreifenden* Kommunikation erkennt der Auftraggeber und der Auftragnehmer während der Nachricht- bzw. Antwortübermittlung, in Abb. 9 auf Seite 30, Punkt 1 und 4, Verbindungsfehler zum aufgerufenen Rechner und Rechnerfehler.

Bleibt die Antwort des Auftragnehmers aus, erhält der Auftraggeber nach einer bestimmten Zeit einen *Timeout*, Punkt 3 in Abbildung 8 und Punkt 4 in Abbildung 9. Die Timeout-Zeit muß so bemessen sein, daß es dem Auftragnehmer möglich ist, den Dienst zu empfangen, zu bearbeiten und die Antwort zurück zu schicken.
Der Auftraggeber erkennt nach dem Timeout, einen *Prozeßfehler* der Auftragnehmer-Komponente, einen *Verbindungsfehler* zur „eigenen“ Mailbox und zum „eigenen“ Rechner und eine fehlerhafte „eigene“ *Mailbox*.

Der *Briefträger*, Punkt 2 und 5 in Abb. 9, ist der Übermittler der Nachricht. Er kommuniziert mit den Komponenten indirekt über die Mailbox. Dadurch erkennt er die gleichen Fehler, die der Auftraggeber bzw. der Auftragnehmer während der rechnerlokalen Nachricht- bzw. Antwortübermittlung festgestellt.

Tabelle 1 zeigt den Zusammenhang zwischen den einzelnen Fehlern, die von den Komponenten und dem Briefträger zu den unterschiedlichen Zeitpunkten während den drei Kommunikationsarten erkannt werden.

Wie die Fehler von den Komponenten und dem Briefträger erkannt werden, wird im nächsten Abschnitt „Fehlererkennung“ erläutert.

erkennt	Prozeßlokal	Rechnerlokal	Rechnerübergreifend
Auftraggeber **während der Nachrichtenübermittlung**	- fehlerhafter Dienst - fehlerhafter Dienstaufruf (Pkt.1,Abb.8)	- Prozeßfehler des Auftragnehmers - keine Verbindung zur Mailbox - ungültige Mailboxadresse / Mailbox existiert nicht - volle Mailbox (Pkt. 2, Abb. 8)	- keine Verbindung zum Auftragnehmer-Rechner - Auftragnehmer-Rechnerfehler - Netzfehler (Pkt 1 in Abb.9)
Auftraggeber **nach einem Timeout**		- Prozeßfehler des Auftragnehmers - keine Verbindung mit „eigener“ Mailbox - „eigene“ Mailboxadresse ist ungültig/ „eigene“ Mailbox existiert nicht - „eigene“ Mailbox ist voll (Pkt. 4, Abb.8)	- keine Verbindung zum Auftragnehmer-Rechner - keine Verbindung mit „eigenem“ Rechner - keine Verbindung mit „eigener“ Mailbox - „eigene“ Mailboxadresse ist ungültig/ „eigene“ Mailbox existiert nicht - „eigene“ Mailbox ist voll (Pkt. 3, Abb.9)

TABELLE 1. Fehlererkennung

erkennt	Prozeßlokal	Rechnerlokal	Rechnerübergreifend
***Auftragnehmer* während der Übermittlung der Antwort**		- Prozeßfehler des Auftraggebers - keine Verbindung zur Mailbox - ungültige Mailbox-adresse/ Mailbox existiert nicht - volle Mailbox (Pkt. 3, Abb.8)	- keine Verbindung mit Auftraggeber-Rechner - Auftraggeber-Rechnerfehler - Netzfehler (Pkt. 4, Abb.9)
Briefträger			- Prozeßfehler - keine Verbindung zur Mailbox - ungültige Mailbox-adresse/ Mailbox existiert nicht - volle Mailbox (Pkt. 2 und Pkt. 5, Abb. 9)

TABELLE 1. Fehlererkennung

4.2 Fehlererkennung

Dieses Kapitel zeigt, **wie** die in Tabelle 1 gezeigten Fehler vom Auftraggeber, Auftragnehmer und vom Briefträger während der prozeßlokalen, rechnerlokalen und rechnerübergreifenden Kommunikation erkannt werden.

4.2.1 Prozeßlokale Fehlererkennung

Um Fehler prozeßlokal zu erkennen, wird ein Überwachungsthread eingeführt. Er soll die Dienste, die in Threads ablaufen, überwachen.

Threads kommunizieren über globale Variablen des Prozesses miteinander. Der lesende und schreibende Zugriff auf diese Variablen muß exklusiv sein. Um dies zu erreichen, müssen sie mit Hilfe von Mutexen (mutual exclusion) synchronisiert werden.

Jeder Dienst muß, um sein „Leben“ zu bekunden, zu Beginn der Dienstausführung und nach einem Timeout, eine im Prozeß globale „*I‘m alive*“-Variable setzen. Während der Ausführung des Programms erhält der Thread, wenn die vorher bestimmte Zeitspanne überschritten ist (*Timeout*), ein Signal vom Überwachungsthread. Auf dieses Signal hin, muß die im Prozeß globale „*I‘m alive*“-Variable vom Thread gesetzt werden.

Der Überwachungsthread überprüft, für jeden Thread, den Zustand der Variable und setzt sie wieder in den Originalzustand.
Bei einem fehlerhaften Dienstaufruf kann die *„I‘m alive“-Variable* zu Beginn des Threads nicht gesetzt werden. Bei einer fehlerhaften Dienstfunktion kann nach einem Timeout die *„I‘m alive“-Variable* nicht gesetzt werden. Der Überwachungsthread erkennt dadurch den fehlerhaften Dienstaufruf bzw. die fehlerhafte Dienstfunktion.

4.2.2 Rechnerlokale Fehlererkennung

Die Komponenten kommunizieren indirekt über Mailboxen. Fehler werden während der Mailbox-Kommunikation, in der Sitzungsschicht (Schicht 5) des OSI-7 Schichten-Modells, von den Komponenten festgestellt. Die von dieser Schicht bereitgestellten Kommunikationsfunktionen zeigen *Prozeß*-, *Verbindungs*- und *Mailboxfehler* durch entsprechende *Fehlermeldungen* an.

Fehler der aufgerufenen Komponente, erkennt der *Auftraggeber* bzw. der *Auftragnehmer während* der Nachrichten- bzw. Antwortübermittlung (Punkt 2 und 4 in Abbildung 8) durch nachfolgende Fehlermeldungen.

Ein *Verbindungsfehler* zur Mailbox wird durch die Fehlermeldung CS_RC_NO_QUOTA beim Verbindungsaufbau mit der Mailbox angezeigt.

Ein *Prozeßfehler* wird durch die Fehlermeldung CS_RC_COMP_DOWN erkannt. Eine andere Komponente hat bereits den Prozeßfehler erkannt und markierte das „*Flag*“ an der Mailbox als „DOWN“. Diese Fehlermeldung impliziert, daß die Komponente bereits im NameService der Inselgruppe als „DOWN“ aktualisiert wurde.

Durch die Fehlermeldung CS_RC_MAILBOX_ADDRESS_INVALID erkennt der Auftraggeber, daß *keine Mailbox* mit der angegebenen Adresse *existiert.*
Eine Verbindung mit der genannten Mailbox kann nicht hergestellt werden und die Nachricht kann nicht in sie hineingeschrieben werden.

Die Fehlermeldung CS_RC_MAILBOX_OVERFLOW bedeutet, daß die *Mailbox voll* ist und nichts in sie hinein geschrieben werden kann.

Die aufrufende Komponente stellt nach dieser Fehlermeldung, den Prozeßstatus der aufgerufen Komponente fest.

Existiert der Prozeß, handelt es sich um einen *Mailbox*fehler. Existiert der Prozeß nicht, handelt es sich um einen *Prozeßfehler*.

Der Prozeßzustand der aufgerufenen Komponente läßt sich aus der Prozeßtabelle des Rechners ermitteln. Das Betriebssystem legt für die auf dem Rechner laufenden Prozesse eine Prozeßtabelle an. In ihr stehen, für jeden Prozeß, alle für das Betriebssystem notwendigen Daten, die es beim und nach dem Auslagern des Prozesses benötigt.
Die in der Prozeßtabelle eingetragenen Informationen werden hier ausgenutzt, um den Prozeßstatus zu ermitteln.

Ein Prozeß *existiert*, wenn er in der Prozeßtabelle steht und der Zustand des Prozesses auf Funktionsfähigkeit hindeutet.

Ein Prozeß *existiert nicht*, wenn er nicht in der Prozeßtabelle steht oder der Zustand des Prozesses auf Funktionsunfähigkeit schließen läßt.

Der Auftraggeber erkennt nach einem *Timeout* (Punkt 2 Abbildung 8) entweder einen *Prozeßfehler* der Auftragnehmer-Komponente, indem er den Prozeßstatus aus der Prozeßtabelle des Betriebssystems ermittelt, oder *Prozeß*-, *Verbindungs*- und *Mailboxfehler* durch Simulation der Mailbox-Kommunikation mit der „eigenen“ Mailbox. Die Fehler werden durch die oben beschriebenen Fehlermeldungen bestimmt.

4.2.3 Rechnerübergreifende Fehlererkennung

Auftraggeber bzw. Auftragnehmer kommunizieren rechnerübergreifend mit dem Briefträger, um den Dienstauftrag bzw. die Dienstantwort in die Mailbox des Auftragnehmers bzw. des Auftraggebers schreiben zu lassen.

Die Verbindung mit dem Briefträger steht solange bis die Nachricht erfolgreich in die Mailbox geschrieben wurde oder die Auftraggeber bzw. die Auftragnehmer eine Fehlermeldung erhalten.

Die Briefträger-Kommunikation ist eine verbindungsorientierte Kommunikation, die in der Transportschicht (Schicht 4) des OSI-7 Schichten-Modells mit dem TCP-Protokoll des Internet-Protokolls realisiert wird. Fehler, die bei der Kommunikation des Rechners auftreten, werden von Fehlermeldungen der Kommunikationsfunktionen dieser Schicht angezeigt.

Der *Auftraggeber* bzw. der *Auftragnehmer* erkennt beim Verbindungsaufbau zum Briefträger (Punkt 1und 4 in Abbildung 9) durch das TCP-Protokoll einen *Verbin-*

dungsfehler, den die Fehlermeldung CS_RC_NO_CONN anzeigt.
Diese Fehlermeldung gibt keine Auskunft über den Zustand des Rechners. Die Verbindung kann auch wegen Rechnerausfall nicht zustande gekommen sein.

Der *Rechnerausfall* wird in der Netzwerkschicht (Schicht 3) des OSI-7 Schichten - Modells festgestellt. Ein Kontroll-Paket wird mit der Kommunikationsfunktionalität des IP-Protokolls des Internet-Protokolls an den anderen Rechner geschickt. Reagiert der andere Rechner auf dieses Paket, ist der Rechner funktionsfähig, wenn nicht ist der Rechner funktionsunfähig oder das Netz fehlerhaft. Um festzustellen, ob das Netz funktionsfähig ist, wird ein Kontroll-Paket an den „eigenen“ Rechner gesendet. Reagiert der „eigene“ Rechner nicht, ist das *Netz* die Fehlerursache.

Der *Briefträger* erkennt *beim* Übermitteln der Nachricht zur Mailbox (Punkt 2 und 5 in Abbildung 9) auf die gleiche Weise wie der Auftraggeber und der Auftragnehmer bei der rechnerlokalen Kommunikation (Punkt 2 und 3 in Abbildung 8) *Prozeß-*, *Verbindungs*- und *Mailbox*fehler.

Nach einem *Timeout* (Punkt 2 in Abbildung 9) beauftragt der Auftraggeber den Briefträger, den Prozeßzustand der Auftragnehmer-Komponente festzustellen. Der Briefträger informiert sich über den Prozeßzustand aus der Prozeßtabelle des Betriebsystems und gibt das Resultat an den Auftraggeber zurück.
Im Fall „Auftragnehmer lebt“, geht der Auftraggeber von einen fehlerhaften Kommunikation mit der „eigenen“ Mailbox aus. Aus diesem Grund simuliert er die rechnerübergreifende Kommunikation mit „seiner“ Mailbox. Der Auftraggeber erkennt so *Netzfehler*, *Verbindung*s- und *Mailboxfehler* der „eigenen“ Mailbox und einen *Prozeßfehler* der „eigenen“ Komponente.

5 Fehlerbehandlung

Nach der Fehlererkennung muß die erkennende Komponente entsprechend auf den Fehler reagieren und ihn anderen Komponenten bekanntgeben. Dies geschieht, indem der ConTractManager, die Mailbox-Verwaltung und der lokale und bei rechnerübergreifender Kommunikation der entfernte NameService aktualisiert werden.

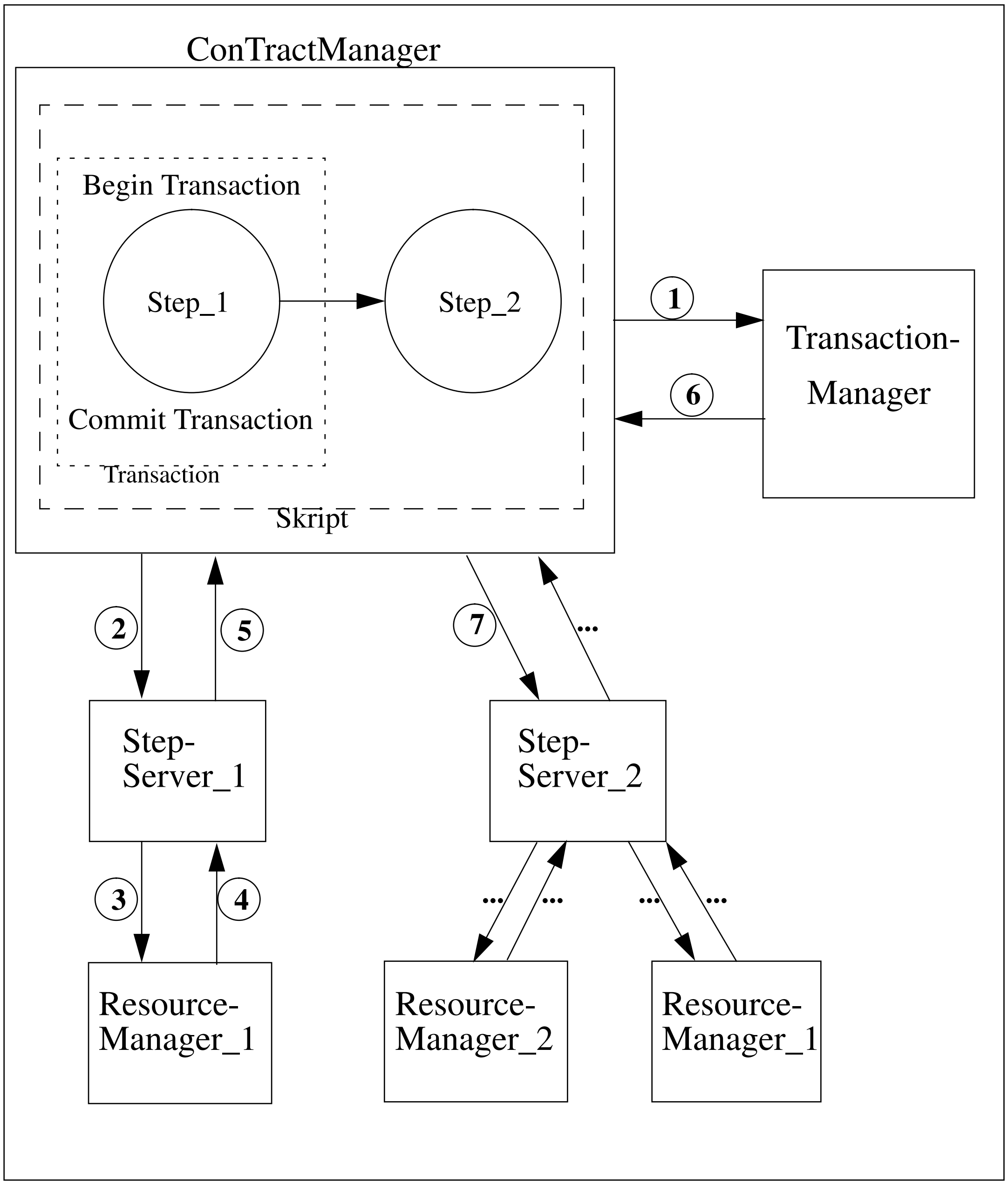

Abb. 10 Aufrufkette bei der Bearbeitung eines ConTracts

Abbildung 10 zeigt schematisch die Aufrufkette des Skriptes eines ConTracts. Der ConTract besteht aus zwei Steps, die sequentiell bearbeitet werden. Step_1 läuft unter Transaktionsschutz ab, Step_2 nicht.

Vor Initiierung des StepServer_1 (Punkt 2) teilt der ConTractManager dem TransactionManager den Beginn der Transaktion (Punkt 1) mit. Der StepServer_1 initiiert den ResourceManager_1 (Punkt 3), der den Dienst bearbeitet und die Antwort an den StepServer_1 zurücksendet (Punkt 4). Der StepServer_1 wiederum bearbeitet die Antwort und meldet die Beendigung des Steps_1 an den ConTractManager (Punkt 5). Der ConTractManager beendet die Transaktion, indem er dem TransactionManager eine „Commit"-Nachricht sendet (Punkt 6). Als nächstes wird der StepServer_2 initiiert (Punkt 7).
Da Step_2 nicht unter Transaktionsschutz steht, muß der TransactionManager nicht benachrichtigt werden. Der StepServer_2 initiiert die zwei ResourceManager, die den Dienst bearbeiten, empfängt die Antworten, bearbeitet sie und teilt die Beendigung des Step_2 dem ConTractManager mit.

Fehler in der Aufrufkette müssen sofort dem ConTractManager mitgeteilt werden und **nicht** dem TransactionManager. Der *TransactionManager* führt Buch über die in der Transaktion beteiligten Komponenten, schreibt den Log, verschickt möglicherweise „Abort"-Nachrichten an die Komponenten und führt die Transaktion nach dem „2-Phase-Commit"-Protokoll aus. Er hat keine Informationen über den im Skript beschriebenen Flußverlauf des ConTracts. Die hat nur der ConTractManager. Der ResourceManager beginnt, beendet und bricht die Transaktion ab, indem er eine „begin", „commit" und „abort"-Nachricht an den TransactionManager sendet.

Der ConTractManager entscheidet, welche Maßnahmen innerhalb des ConTract-Skripts durchzuführen sind. Tritt z.B. ein Fehler in einer unter Transaktionsschutz stehenden Aktion auf (Step_1, Abb. 10), muß der ConTractManager die Transaktion abbrechen und dem TransactionManager eine „Abort"-Nachricht schicken. Dieser bricht die, in der Transaktion, beteiligten Komponenten ab. Wie weit und welche Aktionen der ConTractManager nach dem Fehler zurücksetzten muß und welche StepServer neu initiiert werden müssen, ergibt sich aus dem Skript.

Stehen Teile im Skript nicht unter Transaktionsschutz (Step_2, Abb. 10) ist es auch denkbar, daß eine andere Komponente beauftragt wird, die fehlerhafte Komponente zu ersetzen. Beispiel: In Abbildung 10 stellt der ResourceManager_2 bei der Antwortübermittlung den Ausfall von StepServer_2 fest. Der ConTractManager könnte dem ResourceManager_2 einen anderen StepServer nennen, der die gleiche Funktionalität anbietet. Der ResourceManager könnte dann die Antwort an diesen StepServer übermitteln. Der StepServer bearbeitet die Antwort und schickt das Resultat an den ConTractManager. In diesem Fall müßte der ConTractManager keine Rücksetzaktionen einleiten.

Aus den zwei oben genannten Gründen müssen die *StepServer* und die *ResourceManager* als erstes den *ConTractManager* über den Fehler informieren. Damit ist es dem ConTractManager möglich, sofort die erforderlichen Maßnahmen einzuleiten.

In Kapitel 4 „Fehlererkennung“ wurde erläutert, wie die Kommunikationspartner während der Kommunikation Fehler erkennen. Es wird von Auftrageber, Auftragnehmer und Briefträger gesprochen. In der Aufrufhierarchie (Abbildung 10) der ConTract-Verarbeitung nimmt der ConTractManager immer die Rolle des Auftraggebers, der StepServer beide Rollen und der ResourceManager die Rolle des Auftragnehmers ein.

Daraus ergibt sich, daß der ConTractManager Kommunikationsfehler zum StepServer, StepServer-Prozeßfehler und „eigene“ Kommunikationsfehler erkennt.
Der StepServer erkennt in der Rolle des Auftraggebers Kommunikationsfehler mit dem ConTractManager und ConTractManager-Prozeßfehler und in der Rolle des Auftragnehmers Kommunikationsfehler mit dem ResourceManager, ResourceManager-Prozeßfehler und „eigene“ Kommunikationsfehler.
Der ResourceManager stellt Kommunikationsfehler mit dem StepServer und einen StepServer-Prozeßausfall fest.

Im folgenden wird die Fehlerbehandlung durch die „Threadüberwachung“, den ConTractManager, den StepServer, den ResourceManager und den Briefträger erläutert.

5.1 Fehlerbehandlung durch die „Threadüberwachung“

Die „*Threadüberwachung*“ in der Komponente stellt einen fehlerhaften Dienst bzw. einen fehlerhaften Dienstaufruf fest. Ist der Dienst im Flußverlauf des ConTracts involviert, wird der ConTractManager vom Überwachungsthread über den fehlerhaften Dienst informiert.

Folgende Maßnahmen müssen durchgeführt werden:

1. Der Dienst wird von der Threadüberwachung abgebrochen.
2. Der *ConTractManager* wird von der „Threadüberwachung“ über den fehlerhaften Dienst informiert.
3. Der Dienst wird im *NameService* der Inselgruppe aktualisiert.
4. Der Dienst wird in der prozeßlokalen *Einsprungtabelle* als nicht verfügbar markiert.

5.2 Fehlerbehandlung durch den ConTractManager und den StepServer

Während der Dienstinitiierung erkennt der *ConTractManager* (Punkt 2, Abb. 10) Prozeß- und Kommunikationsfehler des *StepServers* und der wiederum (Punkt 3, Abb. 10) in der Rolle des Auftraggebers Prozeß- und Kommunikationsfehler des *ResourceManager*.

Auf die einzelnen rechnerlokalen Kommunikationsfehler reagiert der ConTractManager bzw. der StepServer wie folgt:

Nach einem *Verbindungsfehler* mit der Mailbox aktualisiert der ConTractManager bzw. der StepServer den NameService **nicht**, da der Verbindungsfehler kein langfristiger Fehler ist, sondern nur im Moment des Verbindungsaufbaus oder etwas länger auftritt.
Trotzdem löscht der ConTractManager die Dienste des StepServers.
Der StepServer löscht die ResourceManager-Dienste aus dem lokalen *Cache*.

Tritt ein *Mailboxadressenfehler/Nichtexistenz* der Mailbox auf, müssen alle Dienste der StepServer- bzw. der ResourceManager-Komponente im *NameService* und im lokalen *Cache* als „DOWN“ markiert werden.

Stellt der ConTractManager bzw. der StepServer eine „*volle Mailbox*“ fest, ermittelte er den Prozeßzustand der Auftragnehmer-Komponente.
Ist der StepServer (Auftragnehmer) bzw. der ResourceManager *funktionsfähig*, wird niemand benachrichtigt und nichts aktualisiert.
Wird ein funktions*un*fähiger StepServer bzw. ResourceManager festgestellt, werden die Dienste des StepServers bzw. des ResourceManagers im *NameService* als funktionsunfähig markiert und die *Mailbox*-Verwaltung beauftragt das „Flag“ der Mailbox auf funktionsunfähig zu setzen.

Nach der rechnerübergreifenden Fehlererkennung muß zwischen Zugehörigkeit und Nichtzugehörigkeit zur Inselgruppe unterschieden werden. Der ConTractManager bzw. der StepServer aktualisiert bei einem Verbindungs- Rechner- und Netzfehler den NameService, wenn der Rechner zur Inselgruppe gehört. Wenn nicht, wird der NameServer aktualisiert.

Bei einem *Verbindungsfehler* mit dem anderen Rechner aktualisiert der ConTractManager und der StepServer den NameService bzw. den NameServer **nicht**.

Stellt der ConTractManager bzw. der StepServer einen *Rechnerfehler* fest, wird im NameService bzw. im NameServer der Rechner als „DOWN“ markiert und alle Diensteinträge des Rechners aus dem lokalen Cache gelöscht.

Bei einem *Netz*fehler kann keine Verbindung zum NameService hergestellt werden. Keine Dienste können rechnerübergreifend initiiert werden. Das „Rescheduling" ist *unnötig*. Der ConTractManager muß informiert werden.

Nach einem *Prozeßfehler* werden die Dienste der StepServer- bzw. der Resource-Manager-Komponente aus dem lokalen *Cache* gelöscht.

In *keinem* der oben genannten Fehlerfällen wird der ConTractManager informiert, da der Dienst beim StepServer bzw. beim ResourceManager nicht initiiert wurde und somit der Dienst nicht im Flußverlauf des ConTracts integriert ist.

Nach dem fehlerhaften Dienstauftruf adressiert der ConTractManager und der ResourceManager einen neuen Dienst. Entsprechend der Scheduling-Strategie wird entweder rechnerlokal oder rechnerübergreifend ein anderer möglicher Dienst gesucht und initiiert.

Die Reaktionen des ConTractManagers und des StepServers *während* der Nachrichtenübermittlung sind in tabellarischer Form in Tabelle 2 zusammengefaßt.

	Reaktionen des ConTractManagers und des StepServers auf Fehler während der Nachrichtenübermittlung
Verbindungsfehler mit Mailbox (CS_RC_NO_QUOTA)	1. Löschen aus lokalem Cache. 2. Rescheduling.
Mailboxadressenfehler/ Nichtexistenz der Mailbox (CS_RC_MAILBOX_-ADDRESS_INVALID)	1. Aktualisieren des NameService. 2. Löschen aus lokalem Cache. 3. Rescheduling.
Volle Mailbox (CS_RC_MAILBOX_-OVERFLOW)	1. Ermittlung des Prozeßzustand des StepServer (Auftragnehmer) bzw. des ResourceManager - funktionsfähig: 1. Rescheduling - funktionsunfähig: 1. Aktualisieren des NameService. 2. „Flag" an Mailbox „DOWN" setzen. 3. Löschen aus lokalem Cache. 2. Rescheduling.

TABELLE 2: Fehlerbehandlung während der Nachrichtenübermittlung

	Reaktionen des ConTractManagers und des StepServers auf Fehler während der Nachrichtenübermittlung
Prozeßfehler des StepServers (Auftragnehmer) bzw. des ResourceManager (CS_RC_COMP_DOWN)	1. Löschen aus lokalem Cache. 2. Rescheduling.
Verbindungsfehler mit Rechner (CS_RC_NO_CONN)	1. Rescheduling.
Rechnerfehler	1. Aus anderer Inselgruppe: - Löschen aus NameService - Aktualisieren des NameServers. Aus Inselgruppe: - Aktualisieren des NameService. 2. Löschen aus lokalem Cache. 3. Rescheduling.
Netzfehler	1. ConTractManager informieren.

TABELLE 2: Fehlerbehandlung während der Nachrichtenübermittlung

Nun wird die Fehlerbehandlung der festgestellten Fehler nach einem *Timeout* auf das Ausbleiben der Antwort des StepServers (Punkt 5, Abb. 10) und des ResourceManagers (Punkt 4, Abb. 10) aufgezeigt. Die vom ConTractManager und StepServer erkannten Fehler sind: *Prozeßfehler* und „eigene" *Kommunikationsfehler.*

Der initiierte Dienst ist nach allen vier Fehlern im Flußverlauf des ConTracts eingebettet, deshalb wird der *ConTractManager* immer über die Fehler informiert.

Stellt der ConTractManager bzw. der StepServer einen *Prozeßfehler* des StepServers (Auftragnehmer) bzw. des ResourceManager fest, müssen nach Benachrichtigung des ConTractManagers, die Dienste der fehlerhaften Komponente im NameService „DOWN" markiert werden und im lokalen Cache gelöscht werden.

Erkennt der ConTractManager bzw. der StepServer einen *Verbindungsfehler* mit „seiner" Mailbox wird, im Gegensatz zum Verbindungsfehler während der Nachrichtenübermittlung, im NameService die Dienste der „eigene" Komponente als fehlerhaft aktualisiert. Der Grund dafür ist, daß die fehlerhafte Verbindung die korrekte Antwortübermittlung seit langer Zeit (Timeout) behindert.

Die Dienste der ConTractManager- bzw. der StepServer-Komponente müssen im Fall einer *falschen Mailboxadresse* bzw. der *Nichtexistenz* der Mailbox im NameService als defekt markiert werden.

Ist die „eigene“ *Mailbox voll*, sollten die Aufträge darin so schnell wie möglich bearbeitet werden. Tritt beim Lesen aus der Mailbox ein Fehler auf, wird entsprechend der Fehlermeldung „falsche Mailboxadresse“ verfahren.

Stellt der ConTractManager bzw. der StepServer an der „eigenen“ Mailbox einen *Prozeßfehler* fest, wird das „Flag“ an der Mailbox auf *funktionsfähig* gesetzt. Der StepServer informiert den ConTractManager über seine Funktionsfähigkeit. Eventuell sind die vom ConTractManager eingeleiteten Aktionen auf die Meldung des ResourceManagers hinfällig. Wenn nicht, wird die Information des StepServers ignoriert. Natürlich müssen im NameService die Dienste der ConTractManager- bzw. der StepServer-Komponente auf „UP“ gesetzt werden.

Wird ein *Kommunikationsfehler* (Verbindungs-, Rechner- und Netzfehler) mit dem „eigenen“ Rechner festgestellt, muß der ConTractManager informiert werden. Sonst muß nichts getan werden.

In Tabelle 3 sind die einzelnen Reaktionen nach Erhalt eines Timeouts auf die Fehler nochmals zusammengefaßt.

	Reaktionen auf einen Timeout nach Ausbleiben der Antwort
Prozeßfehler des StepServers (Auftragnehmer) bzw. des ResourceManagers	1. Informieren des ConTractManagers. 2. Aktualisieren des NameService. 3. „Flag“ an Mailbox „DOWN“ setzen. 4. Aktualisieren des lokalen Caches.
Verbindungsfehler mit „eigenen“ Mailbox (CS_RC_NO_QUOTA)	1. Informieren des ConTractManagers. 2. Aktualisieren des NameServices.
Adressenfehler bzw. Nichtexistenz der „eigenen“ Mailbox (CS_RC_MAILBOX_ADDRESS_-INVALID)	1. Informieren des ConTractManagers. 2. Aktualisieren des NameService.

TABELLE 3: Fehlerbehandlung nach einem Timeout

	Reaktionen auf einen Timeout nach Ausbleiben der Antwort
Volle Mailbox (CS_RC_MAILBOX_OVERFLOW)	1. Nachrichten aus der Mailbox lesen. Wenn Lese-Fehler auftritt: - ConTractManager informieren. - NameService aktualisieren.
„Flag“ an „eigener“ Mailbox ist auf „DOWN“ gesetzt. **(Prozeßfehler des Auftraggebers).** (CS_RC_COMP_DOWN)	1. „Flag“ an Mailbox „UP“ setzen. 2. Informieren des ConTractManagers. 3. Aktualisieren des NameService.
Verbindungsfehler mit „eigenem“ Rechner; **Rechner- und Netzfehler**	1. Informieren des ConTractManagers.

TABELLE 3: Fehlerbehandlung nach einem Timeout

5.3 Fehlerbehandlung durch den StepServer und den ResourceManager

Während der Antwortübermittlung erkennt der *StepServer* in der Rolle des Auftragnehmers Prozeß- und Kommunikationsfehler des *ConTractManagers* (Punkt 5, Abb. 10) und der *ResourceManager* Prozeß- und Kommunikationsfehler des *StepServers* (Auftragnehmer).

Nun wird die Behandlung der Fehler erläutert, die der ResourceManager und der StepServer als Auftragnehmer feststellen.

Der ConTractManager muß nach jedem Fehlerfall informiert und der NameService aktualisiert werden. Die Dienste des ConTractManagers bzw. des StepServers werden nicht aus dem lokalen Cache gelöscht, da die ConTractManager- bzw. StepServer-Adresse als Argument des *resource_manager_call()* mitgeschickt wurde. Die Auftragnehmer-Komponente ermittelte die Adresse nicht selbst. Die Adresse steht nicht im lokalen Cache.

Stellt der StepServer bzw. der ResourceManager einen *Verbindungsfehler* (der Mailbox oder des Rechners) fest, versucht er solange eine Verbindung aufzubauen bis die Verbindung steht oder der ConTractManager bzw. der TransactionManager den Dienst abbricht.

Im Fall der *vollen* Mailbox und einer *intakten* ConTract- bzw. StepServer-Komponente versucht der StepServer bzw. der ResourceManager solange die Nachricht in die Mailbox zu schreiben bis es gelingt oder der ConTractManager bzw. der TransactionManager den Dienst abbricht.

Im Fall der *vollen* Mailbox und eines *defekten* ConTractManagers kann der ConTractManager nicht informiert werden. Der StepServer versucht immer wieder die Antwort an den ConTractManager zu schicken.
Je nach Transaktionszustand (z.B. commit) oder der Aufrufart (synchron) sind die aufgerufenen Dienste des StepServers und des ResourceManagers blockiert. In diesem Fall sollte es möglich sein einen anderen ConTractManager zu informieren oder direkt mit dem TransactionManager zu kommunizieren.
Auf jeden Fall müssen die Dienste der ConTractManager-Komponente im NameService aktualisiert und das „Flag“ an der Mailbox auf „DOWN“ gesetzt werden.

Im Fall der *vollen* Mailbox und eines *defekten* StepServers wird der ConTractManager informiert, im NameService die Dienste der StepServer-Komponente aktualisiert und das „Flag“ gesetzt.

Nach einem *Rechnerfehler* wird sofort der ConTractManager informiert und der NameService und der NameServer aktualisiert.

Bei einem *Netzfehler* muß der ConTractManager informiert werden, sonst kann nichts aktualisiert werden.

Stellt der ResourceManager einen StepServer-*Prozeßfehler* fest, muß der ConTractManager informiert und die Dienste der StepServer-Komponente im NameService als defekt markiert werden.

Stellt der StepServer einen ConTractManager-*Prozeßfehler* fest, sollte eventuell ein anderer ConTractManager oder der TransactionManager über den fehlerhaften ConTractManager informiert werden. Die Dienste der ConTractManager-Komponente werden im NameService als „DOWN“ markiert.

In tabellarischer Form sind die einzelnen Fehlerbehandlungsschritte des Auftragnehmers während der Antwortübermittlung nochmals in Tabelle 4 dargestellt.

	Reaktionen des StepServers und des ResourceManager auf Fehler bei der Antwortübermittlung
Keine Verbindung zur Mailbox (CS_RC_NO_QUOTA)	1. Informieren des ConTractManagers.
Falsche Mailboxadresse bzw. Nichtexistenz (CS_RC_MAILBOX_-ADDRESS_INVALID)	1. Informieren des ConTractManagers. 2. Aktualisieren des NameService.
Volle Mailbox (CS_RC_MAILBOX_-OVERFLOW)	1. Feststellen des Prozeßzustands des ConTractManagers bzw. des StepServers (Auftraggeber): - funktionsfähig 1. Nachricht in Mailbox schreiben. - funktionsunfähig 1. ConTractManager wird informiert. 2. Aktualisieren des NameService. 3. „Flag“ an der Mailbox auf „DOWN“ setzen.
Prozeßfehler des ConTractManagers (CS_RC_COMP_DOWN)	1. Informieren eines anderen ConTractManagers.
Verbindungsfehler zum Rechner (CS_RC_NO_CONN)	1. Verbindung erneut herstellen.
Rechnerfehler	1. Informieren des ConTractManagers. 2. Aus anderer Inselgruppe: - Löschen aus NameService - Aktualisieren des NameServers. Aus Inselgruppe: - Aktualisieren des NameService.
Netzfehler	1. ConTractManager informieren.

TABELLE 4: Fehlerbehandlung während der Antwortübermittlung

5.4 Fehlerbehandlung durch den Briefträgers

Der Briefträger muß nach einem von ihm erkannten *Prozeßfehler* das „Flag" an der Mailbox auf „DOWN" setzen und den NameService *seiner* Inselgruppe aktualisieren, wenn der Briefträger sich in einer anderen Inselgruppe befindet als die aufrufende Komponente. Auch bei einer ungültigen Mailboxadresse bzw. einer nicht existierenden Mailbox muß der Briefträger den NameService aktualisieren, wenn die aufrufende Komponente sich in einer anderen Inselgruppe befindet.

Die sonst vom Briefträger festgestellten Fehlermeldungen werden an die aufrufende Komponente weitergeleitet.

	Reaktionen des Briefträgers bei der Nachrichtenübermittlung
Verbindungsfehler mit Mailbox (CS_RC_NO_QUOTA)	1. Fehlermeldung wird an Komponente weitergeleitet.
Mailboxadressenfehler/Nichtexistenz der Mailbox (CS_RC_MAILBOX_ ADDRESS_INVALID)	Aus anderer Inselgruppe: 1. Aktualisieren des NameService der Inselgruppe. 2. Eintrag ins Repository. Aus gleicher Inselgruppe: 1.Weiterleiten der Fehlermeldung an Komponente.
Volle Mailbox (CS_RC_MAILBOX_ OVERFLOW)	Feststellen des Komponenten-Zustands: - funktionsfähig: 1. CS_RC_MAILBOX_OVERFLOW weiterleiten. - funktionsunfähig: 1. „Flag" an Mailbox „DOWN" setzen. 2. Aus anderer Inselgruppe: - Aktualisieren des NameService. - Eintrag ins Repository. 3. CS_RC_COMP_DOWN weiterleiten.
Prozeßfehler (CS_RC_COMP_-DOWN)	1. Fehlermeldung wird an Komponente weiterleiten.

TABELLE 5: Fehlerbehandlung des Briefträgers bei der Nachrichtenübermittlung

6 Realisierung

In diesem Kapitel wird eine Möglichkeit aufgezeigt die in Kapitel 4 beschriebene Fehlererkennung und die in Kapitel 5 beschriebene Fehlerbehandlung zu realisieren.

Die in der Einleitung beschriebene Architektur des APRICOT-Systems wurde durch das zum Teil realisierte *ControlSystem*-Modul erweitert, siehe Abb. 11. Das Modul wird in das *CommunicationSystem*-Modul eingebunden.

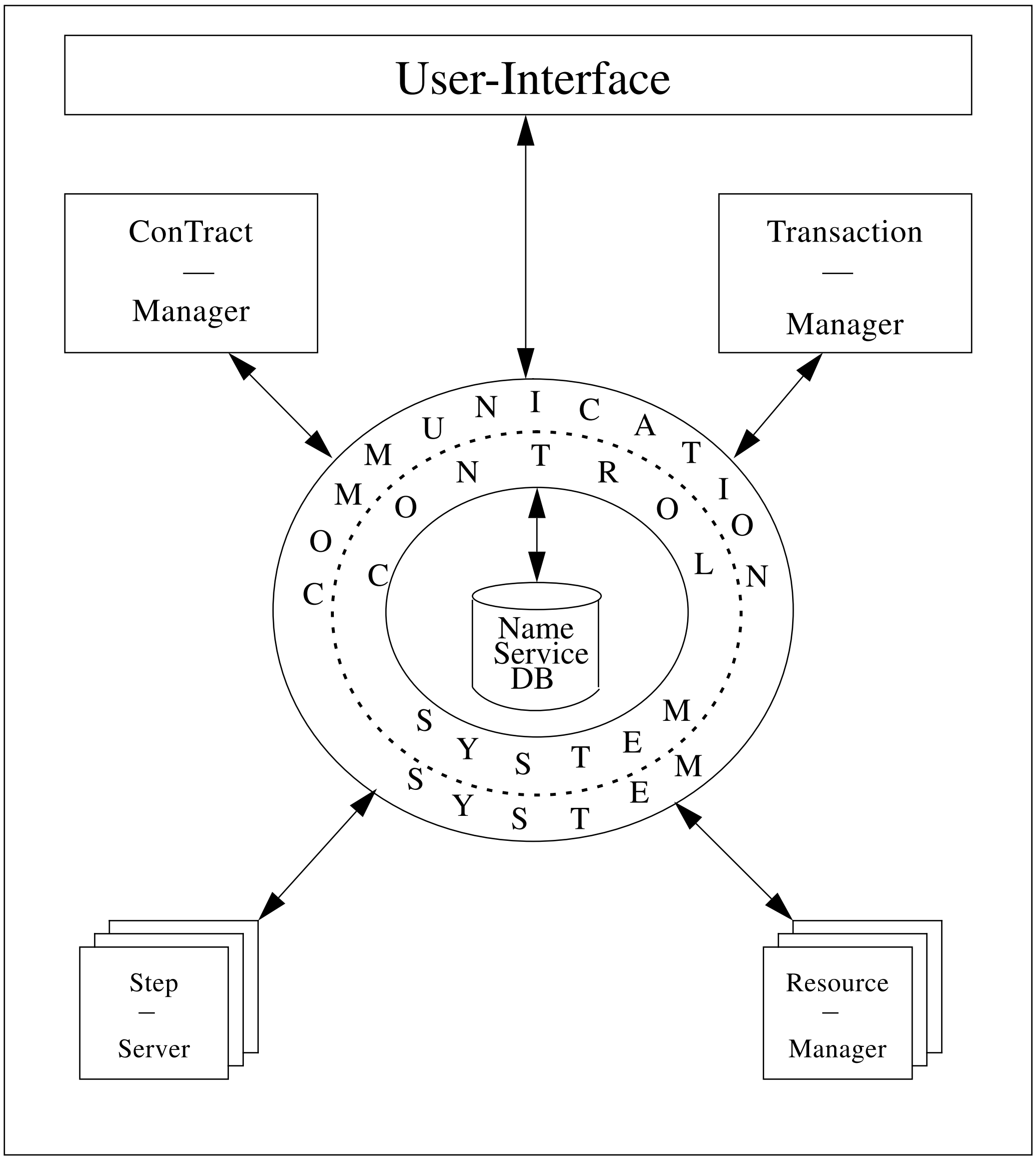

Abb. 11 Erweiterte APRICOTS-Architektur

Jeder Komponente stehen die Funktionalitäten des *ControlSystems* über das *KommunikationsSystem* zur Verfügung.

Das *ControlSystem*-Modul besteht aus der Funktionalität der Fehlererkennung und der Fehlerbehandlung, die in diesem Kapitel beschrieben sind.

6.1 Fehlererkennung

Die in Abschnitt 4.2 beschriebenen Fehlermeldungen werden durch die in APRICOTS zur Verfügung stehenden Kommunikationsfunktionen zurückgegeben.

Die Funktion *cs_connect_mailbox()* stellt eine Verbindung auf der Sitzungsschicht des OSI-7 Schichten-Modells zur Mailbox her und liefert die Fehlermeldung CS_RC_NO_QUOTA zurück. Die Nachricht wird mit der Funktion cs*_write_message_into_mailbox()* in Mailbox geschrieben. Die Funktion liefert die Fehlermeldungen CS_RC_MAILBOX_OVERFLOW und CS_RC_MAILBOX_ADDRESS_INVALID zurück. Diese Schreib-Funktion muß um ein „Flag“ erweitert werden.

Die Funktion muß vor dem Schreiben in die Mailbox das „*Flag*“ abfragen. Wenn das „*Flag*“ auf „UP“ gesetzt ist, wird die Nachricht in die Mailbox geschrieben. Wenn es auf „DOWN“ gesetzt ist, darf die Nachricht nicht in die Mailbox geschrieben werden.
Die Mailbox darf nicht völlig geschlossen werden. Es muß möglich sein, Nachrichten aus der Mailbox zu lesen und zu entfernen.

Aus Zeitmangel war es nicht möglich, die Mailbox um das „*Flag*“ zu erweitern. Darum wurde auch die Funktion cs*_write_message_into_mailbox()* nicht erweitert und die Fehlermeldung CS_RC_COMP_DOWN nicht realisiert.

Die Funktion *remote_message()* stellt eine Verbindung auf der Transportschicht des OSI-7 Schichten-Modells her. Diese Funktion liefert die Fehlermeldung CS_RC_NO_CONN zurück. Nach dieser Fehlermeldung wird die genaue Fehlerquelle mit der in Abschnitt 6.1.2 „Rechnerüberprüfung“ vorgestellten Funktion *check_host()* ermittelt.

Der Timeout sollte so realisiert werden, daß sobald die Nachricht verschickt wurde, eine Alarmuhr gestellt wird, die nach Ablauf der Zeit (Timeout) dem Thread, der den Dienst initiierte, ein bestimmtes Signal mit der in SunOS5.3 verfügbaren Funktion *thr_kill()* schickt. Der Thread muß auf dieses Signal hin, die von der Kommunikationsart abhängige Fehlererkennung und -behandlung ausführen.

6.1.1 Komponentenprüfung

Die aufrufende Komponente soll nach der Fehlermeldung CS_RC_MAILBOX-_OVERFLOW und nach zu langem Warten auf die Auftragnehmerantwort, den Komponentenzustand der aufgerufenen Komponente prüfen. (Siehe „Fehlererkennung“ auf Seite 34.)

Die hier vorgestellten Funktion *check_comp()* und *check_comp_solaris()* stellen den Prozeßzustand der Komponente im Betriebssystem SunOS4.1.3 und in SunOS5.3 (Solaris) fest und liefern die Returncodes CS_RC_COMP_UP und CS_RC_COMP_-DOWN zurück.

Zunächst wird auf die Prozeßverwalung in den UNIX-Derivaten SunOS4.1.3 und SunOS5.3 eingegangen und dann die notwendigen Routinen für die Feststellung des Prozesseszustandes vorgestellt.

Jeder aktive Prozeß allokiert Speicher. In UNIX-Systemen wird für jeden aktiven Prozeß ein Prozeßtabelleneintrag angelegt. Sie beinhaltet alle notwendigen Daten, Adressraum, user_id (u_id), parent_process_id (ppid), process_id (pid) und statistische Daten über den Zustand des Prozesses, die vom System beim und nach dem Auslagern (swap out) des Prozesses benötigt werden.

Prozeßzustandsinformationen sind:

- Wait.

  Der Prozeß wartet auf ein Ereignis (event).

- Run.

  Der Prozeß arbeitet.

- IDL.

  Prozeß wird erzeugt.

- Zombie.

  Der Prozeß ist beendet, aber sein Resultat konnte dem Vaterprozeß nicht zugestellt werden.

- Stop.

  Der Prozeß ist terminiert.

Im UNIX Betriebssystem SunOS4.1.3 wird auf die Prozeßtabelle über *Kernel Virtuell Memory* (kvm) Routinen zugegriffen.

Die Funktion *kvm_getproc()*, spezifiziert mit der Prozeßnummer (pid), liefert einen Pointer auf den Prozeßtabelleneintrag dieses Prozesses zurück. Über die Variable *p_stat* im Tabelleneintrag kann der Prozeßzustand bestimmt werden. Liefert *kvm_getproc()* einen NULL-Pointer zurück, existiert kein Prozeßeintrag für die spezifizierte Prozeßnummer. Daraus läßt sich folgern, daß der Prozeß nicht existiert.

Die Aktualität der Daten mit der *kvm_getproc()*-Routine ist nicht gesichert. Jeder Aufruf der Routine liefert einen Pointer auf den statischen Memorybereich, der eine Kopie des Prozeßtabelleneintrages enthält. Der Zustand des Prozesses kann sich im Moment des *kvm_getproc()*-Aufrufs geändert haben. Für die vorgestellte Prozeßüberwachung sind diese Änderungen nicht wesentlich. Es interessieren nicht die Unterschiede zwischen den einzelnen Zuständen Wait, Run, IDL, Zombie und Stop. Es ist eine Unterscheidung zwischen aktiven und nicht aktiven Prozeß notwendig.
Ein Prozeß ist *aktiv*, wenn er in der Prozeßtabelle eingetragen ist und sich in einem der Prozeßzustände Wait, Run oder IDL befindet.
Ein Prozeß ist *nicht aktiv*, wenn er nicht in der Prozeßtabelle eingetragen ist oder wenn er eingetragen ist und sich in einem der Prozeßzustände Zombie oder Stop befindet.

Im UNIX-Betriebssystem Solaris von SUN greift die Funktion *check_comp_solaris()* über ein Prozeß-Filesystem auf die Prozeßtabelleneinträge zu. Die Namen der Files im /proc Verzeichnis entsprechen der Prozeßnummer (pid) im System. Mit dem Systemaufruf *ioctl()* kann durch unterschiedliche Codes im *ioctl*()-Aufruf auf die Prozeßfiles zugegriffen werden. Der in der Funktion *check_comp_solaris()* verwendete PIOCPSINFO-Code entspricht dem ps-Aufruf in UNIX-Systemen und liefert einen Pointer auf die *prpsinfo*- Struktur. Die Variable *pr_stat* in dieser Struktur enthält den numerischen Prozeßzustand.

Die Zustandsdefinitionen der Prozesse im Betriebssystem SunOS4.1.3 sind nicht die gleichen wie in SunOS5.3. Es muß auf die Prozeßstatusdefinitionen im Headerfile geachtet werden. Die Codes und Datenstrukturen sind im Headerfile <sys/procfs.h> definiert.

6.1.2 Rechnerüberprüfung

Bei der rechnerübergreifenden Kommunikation wird eine Verbindung in der Transportschicht mit dem Socket des Briefträgers aufgebaut. Wird keine Verbindung aufgebaut, untersucht die aufrufende Komponente den Rechnerzustand. Hierfür

soll eine Nachricht in der darunterliegenden Netzwerkschicht an den Rechner geschickt werden.
Reagiert der Rechner auf diese Nachricht ist das Netz und der Rechner funktionsfähig.
Reagiert der Rechner *nicht* wird an den „eigenen“ Rechner die Nachricht geschickt.

Das UNIX-Betriebssystem stellt diese Funktionalität zur Verfügung, um in der Netzwerkschicht ein Kontroll-Paket an den anderen Rechner zu senden. Der Shell-Befehl *ping host* sendet ein ICMP[1]-Paket an den Rechner. Wenn 'ping' erfolgreich ist und der Rechner antwortet, ist der Exit-Status '0'. Antwortet der Rechner nicht, ist der Exit-Status '1'.

Die Funktion *check_host()* führt den Shell-Befehl *ping* mit der *system()*-Routine aus. Lebt der Rechner, gibt *check_host()* den Returncode CS_RC_HOST_UP zurück. Lebt der Rechner nicht, wird der Returncode CS_RC_HOST_DOWN zurückgegeben.

6.2 Fehlerbehandlung

Zu den vier Zeitpunkten (siehe Abbildung 8 und Abbildung 9) der Fehlererkennung müssen die jeweiligen Kommunikationspartner auf die Fehlermeldungen reagieren, indem sie die in Kapitel 5 beschriebenen Schritte der Fehlerbehandlung durchführen.

Die einzelnen Schritte der Fehlerbehandlung werden durch die folgenden Funktionen ausgeführt.

Dem ConTractManager wird eine Fehler-Nachricht entsprechend seiner Adresse prozeßlokal, rechnerlokal oder rechnerübergreifend geschickt. Die Fehler-Nachricht enthält die Adresse der fehlerhaften Komponente (vom Typ CS_ADDRESS_PARAM), den Fehlercode, die Transaktions-Id und die Adresse der erkennenden Komponente.

Die Dienste einer fehlerhaften Komponente werden im lokale NameService mit der Funktion *cs_name_service_update_services()* mit der Option „DOWN“ als fehlerhaft markiert. Befindet sich die fehlerhafte Komponente in einer anderen Inselgruppe, werden die Diensteinträge aus dem lokalen NameService mit der Funktion *cs_name_service_delete_services()* gelöscht.

Bei einem Rechnerfehler wird im NameServer der Rechner mit der Funktion *cs_ns_db_update()* als „DOWN“ markiert.

1. InternetControlMessageProtocol

Dienste einer fehlerhaften Komponente müssen einzeln aus dem lokalen Cache mit der Funktion *cs_local_cache_delete()* gelöscht werden.

Die Funktionen werden zu Anweisungsblöcken laut den Fehlerbehandlungsschritten der Fehler zusammengesetzt. Nach dem fehlerhaften Verbindungsaufbau (*cs_connect_mailbox()* und *cs_connect()*) und nach dem fehlerhaften Schreiben in die Mailbox (cs*_write_message_into_mailbox()*), werden die Anweisungsblöcke der zurückgegebenen Fehlermeldung (*switch*-Anweisung) je nach Fehlermeldung ausgeführt.

Erhält die Auftraggeber (ConTractManager, StepServer) einen Timeout, wird zuerst mit der Funktion *check_comp()* der Prozeßzustand des Auftragnehmers (StepServer, ResourceManager) geprüft. Nach dem Returncode CS_RC_COMP_UP wird eine Verbindung durch die Funktion *cs_connect_mailbox()* mit der „eigenen" Mailboxadresse hergestellt und dann eine leere Nachricht mit der Funktion *cs_write_message_into_mailbox()* in die Mailbox geschrieben. Die möglichen Fehlermeldungen werden in einer *switch*-Anweisung abgefangen und durch einen Anweisungsblock, der den Fehlerbehandlungsschritten entspricht, ausgeführt.

Der Briefträger wird um eine Option CHECK_COMP erweitert. Der Briefträger ermittelt mit der Funktion *check_comp()* den Prozeßzustand und gibt den Returnwert an die aufrufende Komponente weiter. Nach dem Returnwert „CS_RC_COMP_-DOWN" markiert der Briefträger mit der noch nicht realisierten Funktion *cs_mailbox_update()* das „Flag" an der Mailbox „DOWN". Außerdem muß der Briefträger nach der Kommunikation mit der Mailbox entsprechend den Fehlermeldungen eine Fehlerbehandlung durchführen.

7 Zusammenfassung und Ausblick

Mit dem in dieser Studienarbeit vorgestellten Konzept zur Fehlererkennung und Fehlerbehandlung des ConTract-verarbeitenden Systems, APRICOTS, werden Prozeß-, Rechner und Netzfehler erkannt und den Instanzen, die den Ablauf des ConTracts steuern und verwalten, bekanntgegeben.

Das Fehlererkennungskonzept basiert auf den in APRICOTS verwendeten Kommunikationsarten.
Auftretende Fehler werden während der Kommunikation und nicht von *einem* Prozeß erkannt, der die im Rechner bzw. in der Inselgruppe vorhandenen Komponenten ständig überwacht.
Daraus folgt, daß es jeder Komponente möglich ist, Fehler zu erkennen und auf sie zu reagieren. Es folgt aber auch, daß auftretende Prozeß-, Rechner- und Netzfehler nicht sofort erkannt werden, sondern erst, wenn mit dem Prozeß oder dem Rechner kommuniziert wird. Bei Rechner- oder Netzfehlern reicht dieser Erkennungszeitpunkt aus.

Der Prozeßfehler einer ConTractManager-Komponente wird nach diesem Konzept zu spät erkannt. Der ConTractManager steuert den Ablauf des ConTracts und ist für die Steuerung (begin, commit, abort) der Transaktion verantwortlich. Durch Ausfall der ConTractManager-Komponente können Ressourcen lange blockiert werden. Das heißt die ConTractManager-Komponente muß ausfallsicher gemacht werden.

In [GrRe93] wird das Konzept der Prozeßpaare vorgestellt, um einen Prozeß ausfallsicher zu machen. Einem Hauptprozeß wird ein Partnerprozeß zugeordnet. Der Hauptprozeß meldet sich von Zeit zu Zeit bei seinem Partnerprozeß und teilt ihm die aktuellen Daten mit. Meldet sich der Hauptprozeß nicht, übernimmt der Partnerprozeß die Aufgaben des ausgefallenen Prozesses.

Die ConTractManager-Komponente sollte nach dem Prozeßpaar-Konzept realisiert werden, um sie aus den oben genannten Gründen ausfallsicher zu machen.

Um die Fehlerdaten zu veröffentlichen, reicht es nicht aus Fehlermeldungen auf ein Terminal zu drucken und sie einem Benutzer zur Verfügung zu stellen. Es sollten mehrere Benutzer Zugriff auf die Fehler haben.
Dazu ist es nötig eine Fehler-Relation *error* einzuführen, die dauerhaft auf der Datenbank gespeichert wird. In die Relation sollte auf jeden Fall der Komponentenname und der detaillierte Fehlertyp eingetragen werden. Über den Komponentenname kann dann entsprechend dem Fehlertyp auf die notwendigen Daten, wie z.B Rechnername, Mailboxidentifikation, zugegriffen werden.

Nach der Fehlererkennung sollte ein error-Tupel in die Datenbank eingefügt werden. Ein administratives Überwachungs-Tool sollte die Daten zum einen für einen

Administrator und zum anderen für den Benutzer des ConTract-User-Interface graphisch darstellen. Der Administrator leitet die notwendigen Reparaturmaßnahmen ein, der Benutzer des User-Interface kann während der dynamischen Konfiguration des verarbeiteten ConTracts Maßnahmen, wie z.B. Migration eines Dienste, einleiten.

8 Literatur

[Coff89]: Stephen Coffin
UNIX-Befehlsbibliothek
McGraw-Hill Book Company, 1989

[DaGe91]: Ch. Dannegger, P. Geugelin-Dannegger
Parallele Prozesse unter UNIX: Simulation und Anwendung bekannter Synchronisationsmethoden in C unter UNIX
Hanser, 1991

[GrRe93]: Jim Gray, Andreas Reuter
Transaktion Processing: Concepts and Techniques
Morgan Kaufmann, 1993

[Häup93]: Michael Häuptle
Erweiterung eines Kommunikationsmechanismus für den rechnerübergreifenden Betrieb.
Studienarbeit Nr. 1231, IPVR, Universität Stuttgart, 1993

[Laub92]: Dieter Laube
Transationsorientierte Kommunikationsmechanismen für das verteilte Transaktionssystem APRICOTS.
Diplomarbeit Nr. 865, IPVR, Universität Stuttgart, März 1992

[Mill93]: Mark A. Miller
Troubleshooting TCP/IP: Fehleranalyse und -behandlung.
Verlag Heinz Heinse, 1993

[Reut89]: Andreas Reuter
ConTracts: A means for Extending Control beyond Transaction Boundaries.
Proc. 2rd Int. Workshop on High Performance Transaction Systems, Asilomar, Sept.1989

[Schü93]: Alexander Schützle
Portierung/Reimplementierung eines transaktionsorientierten Kommunikationssystems.
Diplomarbeit Nr.1026, IPVR, Universität Stuttgart, 1993

[Trap95]: Jens Trapp
Am dünnen Faden
Zeitschrift: *iX*, S.136 -145, April 1995

[WäRe91]: Helmut Wächter, Andreas Reuter
The ConTract Model.
Computer Science Report No. 8/91,
IPVR Universität Stuttgart, Sept. 1991

[ZiKr93]: Ch. Zimmermann, Albrecht Kraas
Mach: Konzepte und Programmierung.
Springer Verlag, 1993

Erklärung

Hiermit versichere ich, diese Arbeit selbstständig verfaßt und nur die angegebenen Quellen benutzt zu haben.

Anett Frank