Kooperationsanwendungen: Integrierte Vorgangskontrolle und Dienstvermittlung in offenen verteilten Systemen

K. Müller–Jones, M. Merz, W. Lamersdorf

Universität Hamburg
Fachbereich Informatik
Vogt–Kölln–Str. 30, D–22527 Hamburg, Germany

[kmueller|merz|lamersd]@dbis1.informatik.uni-hamburg.de

Zusammenfassung

Angesichts der Anzahl und Vielfalt vorhandener Standard– und Individualdienste in offenen verteilten Systemen gewinnt die Verwendung dieser Dienste innerhalb verteilter Anwendungen zunehmend an Bedeutung. Um die Dienste jedoch auch effizient in die Entwicklung komplexer verteilter Anwendungen einbeziehen zu können, sind geeignete Abstraktionen notwendig, die eine möglichst einfache Nutzung der Dienste ermöglichen. Hierdurch ergeben sich auch neue Anforderungen an die darunterliegende Systemtechnik, welche die notwendige Infrastruktur für eine solche Dienstnutzung bzw. Anwendungskooperation bereitstellen muß. Eine Analyse vorhandener Systeme zeigt jedoch, daß bisher keine integrierten Lösungsansätze existieren, welche die Aspekte der Dienstnutzung mit einer für kooperierende Anwendungen typischen vorgangsorientierten Verarbeitungssicht verbinden.

Auf dem Hintergrund dieser Einschränkungen stellt der vorliegende Beitrag die an der Universität Hamburg entwickelte TRADE–Architektur zur Unterstützung von Kooperationsanwendungen in offenen verteilten Systemen vor. Hierbei wird insbesondere auch auf die neu entwickelte Vorgangsbeschreibungssprache PAMELA eingegangen, welche eine Definition von Anwendungsvorgängen auf der hohen anwendungsspezifischen Abstraktionsebene der Kooperationsanwendungen bietet, in dem sie kontrollspezifische Aspekte der Vorgangsbearbeitung mit einer dienstorientierten Aktionsausführung verbindet.

1 Kooperationsanwendungen in einem offenen Dienstemarkt

Angesichts der weltweiten Verfügbarkeit und rasanten technischen Entwicklung heutiger Telekommunikations– und Netzwerktechnologien gewinnt die effektive, globale Nutzung von offenen verteilten Systemen zunehmend an Bedeutung. Bedingt durch diese Globalisierung eröffnet sich den Benutzern und den Anwendungsentwicklern erstmals ein nahezu unbegrenzter *offener Markt an Diensten* [MML94], die in großer Anzahl und Vielfalt zur Verfügung stehen und in effizienter Art und Weise nutzbar sind. Die Vision eines weltweit verfügbaren *Systembaukastens* von Diensten rückt somit in greifbare Nähe. Dieser bildet die notwendige Voraussetzung für die Entwicklung und Realisierung neuer komplexer verteilter Anwendungen unter *weitgehender Wiederverwendung und Kombination* der bereits im verteilten System angebotenen Standard– und Individualdienste. Diese neue Möglichkeit der weitgehenden Nutzung vorhandener externer Dienste bei der Entwicklung verteilter Anwendungen beeinflußt maßgeblich die Art und Weise der verteilten Programmierung. Ebenso ergeben

sich neue Anforderungen an die darunterliegende Systemtechnik, welche die notwendige Infrastruktur für eine solche Nutzung externer Dienste in einem offenen verteilten System bereitstellen muß. Hierbei hat insbesondere die *Koordination bzw. Steuerung der Kooperation* der genutzten externen Dienste innerhalb einer komplexen verteilten Anwendung eine zentrale Bedeutung.

Zur besonderen Betonung des Kontroll– bzw. Kooperationsaspektes verteilter Anwendungen werden diese im folgenden deshalb als **Kooperationsanwendungen** bezeichnet. Maßgeblich für Kooperationsanwendungen ist, daß sie komplexe *Anwendungsvorgänge* ausführen, die ein oder mehrere externe, a–priori bekannte und wohldefinierte Dienste im Verlaufe ihrer Bearbeitung in Anspruch nehmen. Anwendungsvorgänge bestehen hierbei aus Folgen von (Einzel–)*Aktivitäten*, die eine oder mehrere *Aktionen* unter Nutzung externer Dienste ausführen können.

Hierbei beschränkt sich die Dienstsicht nicht nur auf Diensterbringer im herkömmlichen Sinne (z. B. Berechnungs–, Datenbank– oder Namens–Server), sondern umfaßt auch den Dienst eines menschlichen Benutzers mit seinen Fähigkeiten und Kompetenzen als Diensterbringer. Kooperationsanwendungen bilden hierbei einen zusammenfassenden Begriff für eine Vielzahl verschiedener, sich jedoch zum Teil auch überschneidender Anwendungsbereiche in offenen verteilten Systemen, z. B. dem „Workflow Management“, dem Konfigurationsmanagement, der klassischen verteilten Transaktionsverarbeitung (Buchungen usw.) und der computergestützen Fertigung (CIM).

Innerhalb dieses Anwendungsumfeldes von Kooperationsanwendungen in offenen verteilten Systemen ist der Anwendungsprogrammierer mit zahlreichen, zum Teil neuartigen Problemen konfrontiert. Grundlegende Probleme sind hierbei — auf dem Hintergrund der ständig vorhandenen systeminhärenten Komplexität verteilter Systeme — insbesondere

- die Vermittlung und Verwaltung einer großen Anzahl und Vielfalt räumlich verteilter Dienste,
- die Spezifikation von Anwendungsvorgängen,
- die Ablaufkontrolle bei der Abwicklung der Anwendungsvorgänge sowie
- die Kommunikation beim Zugriff auf externe Dienste innerhalb von Aktivitäten.

Zur Zeit existieren zur Lösung dieser Problemstellungen nur rudimentäre, größtenteils auf spezielle Anwendungsbereiche konzentrierte Ansätze, die jedoch jeweils nur einen Teilbereich davon abdecken. Insbesondere existieren keine Ansätze, die eine *integrierte Unterstützung* von Kooperationsanwendungen bieten. Als ein Beispiel für eine Teillösung kann die wohlbekannte RPC–Programmierung von Client/Server–Anwendungen dienen, die den Zugriff auf entfernte Prozeduren in weitgehend gleicher Art und Weise wie auf lokale Prozeduren erlaubt. Trotz dieser recht mächtigen Zugriffstransparenz ist der Programmierer jedoch nicht vollständig von den Verteilungsproblemen befreit. So wird meist nicht die *Diensttransparenz* unterstützt, die eine Dienstnutzung unabhängig von einem konkreten Diensterbringer erlaubt, d. h. der Programmierer muß noch immer *explizit* den gewünschten Diensterbringer adressieren.

Diese Zugriffs– und Diensttransparenz ist jedoch nur ein Teil des Gesamtproblems der Unterstützung von Kooperationsanwendungen, insbesondere der Forderung nach geeigneten *Abstraktionen*, die eine einfache und effiziente Nutzung der im offenen verteilten System vorhandenen Dienste innerhalb von Kooperationsanwendungen ermöglichen.

Der vorliegende Beitrag gibt im folgenden einen Überblick über Probleme der Unterstützung von Kooperationsanwendungen und weist auf schon vorhandene Lösungsansätze hin. Aufgrund dieser Analyse wird aufgezeigt, wie diese größtenteils isoliert voneinander entwickelten Ansätze im Rahmen einer übergreifenden Architektur, der Systemarchitektur TRADE (Service *Tra*ding and Coor*d*ination *E*nvironment) [MJML95, MJM94, MML94] integriert werden können. Hierbei wird auch näher auf Aspekte einer Prototypimplementierung eingegangen, die zur Zeit an der Universität Hamburg durchgeführt wird. Abschließend erfolgt eine Bewertung der bisher im TRADE–Projekt vorliegenden Forschungsergebnisse, und es wird ein Ausblick auf zukünftig noch zu bearbeitende Fragen der Unterstützung von Kooperationsanwendungen in offenen verteilten Systemen gegeben.

2 Mechanismen zur Unterstützung von Kooperationsanwendungen

Wie schon im vorherigen Abschnitt dargestellt, ist der Programmierer von Kooperationsanwendungen mit zahlreichen Problemen bei der Unterstützung von Kooperationsanwendungen in offenen verteilten Systemen konfrontiert. Im folgenden werden diese Probleme genauer untersucht und — soweit vorhanden — wird auf bereits existierende Lösungsansätze hingewiesen. Gleichzeitig werden Beziehungen zur im nachfolgenden Abschnitt beschriebenen TRADE–Architektur aufgezeigt.

2.1 Dienstvermittlung und –verwaltung

Eine notwendige Voraussetzung für die kontrollierte Vermittlung und Verwaltung von Diensten in offenen Systemen ist das Vorliegen eines formalisierten Dienstbegriffes, der eine Klassifikation von Diensten aufgrund wohldefinierter Kategorisierungs– und Beschreibungsmittel erlaubt. Eine derartige Formalisierung des Dienstbegriffes erlaubt z. B. der *Diensttypbegriff*, wie er im Rahmen internationaler Standardisierung innerhalb der International Standardization Organization (ISO) im Bereich *Open Distributed Processing* (ODP) [ODP93] bzw. *ODP Trading Function* [ISO94] definiert ist. Dieser beeinhaltet die grundlegende Beschreibung der Art des Dienstes, die Menge der angebotenen Operationen mit Argument– und Resultattypen an seiner Schnittstelle sowie die Diensteigenschaften, die den Dienst über syntaktische Aspekte der Schnittstelle hinausgehend anhand von Diensteigenschaften näher charakterisieren. Teilaspekte einer konkreten Beschreibung eines Diensttyps, der sogenannten *Diensttypbeschreibung*, werden in heutigen verteilten Systemplattformen im eingeschränkten Maße durch Schnittstellenbeschreibungssprachen (interface definition languages – IDL) realisiert. Beispiele hierfür sind die DCE–IDL [Fou93] oder die CORBA–IDL [OMG91], auf die im Rahmen der im Abschnitt 3 vorgestellten TRADE–Prototypimplementation noch einmal eingegangen wird.

Auf der Grundlage derartiger Diensttypbeschreibungen können nun generische Systemdienste, die sogenannten *Broker* oder *Trader* [Kel93], eine Dienstvermittlung und –verwaltung realisieren. Diese stellen Mechanismen bereit, die den Erbringern eines speziellen Dienstes (sog. *Exporteuren*) die Registration ihrer Dienstangebote unter einem wohldefinierten Diensttyp erlauben. Dienstnehmer (sog. *Importeure*) können sich nun mit Hilfe einer *Dienstanforderung* spezielle Diensterbringer durch den Trader vermitteln lassen. Diese Dienstanforderung ist ein wichtiger integraler Bestandteil der systemtechnischen Un-

terstützung von Kooperationsanwendungen und wird im Zusammenhang mit der Vorgangsbeschreibung und Ablaufkontrolle innerhalb der TRADE–Architektur in den Abschnitten 3.1 und 3.2 noch genauer konkretisiert.

2.2 Beschreibung von Anwendungsvorgängen

Die koordinierte Nutzung externer Dienste eines offenen verteilten Systems innerhalb eines komplexen Anwendungsvorganges, d. h. definierte Folgen von Aktivitäten, stellt das Hauptmerkmal von Kooperationsanwendungen dar. Zur Beschreibung bzw. Modellierung derartiger Abfolgen von Aktivitäten müssen geeignete formale Hilfsmittel gewählt werden, die sowohl eine Darstellung der Ablaufstrukturen als auch die externe Dienstnutzung als Kernbestandteil von Aktivitäten adäquat ausdrücken. Hierbei sollten auch verschiedene Aufrufsemantiken, z. B. die transaktionale Ausführung mehrere Aktionen innerhalb einer Aktivitität, berücksichtigt werden. Ebenso müssen die Aspekte der Benutzerinteraktion, wie z. B. benutzerseitige Eingaben oder das Anstoßen von Benutzeraktionen, sowie der Fehlerbehandlung innerhalb von Aktivitäten behandelt werden. Die hierfür zu entwickelnde Beschreibungstechnik, z. B. in textueller oder grafischer Form, sollte eine möglichst hohe Abstraktionsebene anbieten, die eine Trennung der Koordinationsaspekte innerhalb von Anwendungsvorgängen von der eigentliche Ausführung der Aktivitäten erlaubt. Dabei lassen sich verteilungspezifische Aspekte, die sich durch das Ausführen der Aktionen innerhalb der Aktivitäten bei den externen Diensten ergeben, soweit wie möglich vor dem Programmierer verbergen. Dies erfordert unter anderem entsprechende Ausdrucksmöglichkeiten für die mit der Dienstnutzung verbundene Dienstauswahl, um eine weitgehende Diensttransparenz zu erreichen.

Es gibt verschiedene Ansätze zur Beschreibung von Anwendungsvorgängen, die vor allem aus dem Anwendungsbereich des Workflow–Managements und der Transaktionsverarbeitung stammen. Beispiele hierfür sind die Aktivitätenbeschreibungssprache ActSpec [Rei93] sowie die Distributed Operation Language (DOL) [HAB+92] und die im ConTract–Modell [WR91] verwandte Definitionssprache, die beide speziell für die verteilte Transaktionsverarbeitung entwickelt wurden.

Zur internen Repräsentation der Anwendungsvorgänge werden oftmals, z. B. auch im ConTract–Modell oder im FlowMark–System [LR94], Petri–Netz ähnliche Modelle verwendet. Petri–Netze [Pet81, Mur89] eignen sich speziell zur Darstellung des strukturellen Aufbaus sowie des dynamischen Verhaltens von Anwendungsvorgängen. Insbesondere höhere Petri–Netze, wie z. B. Prädikat/Transitionsnetze [Gen87] und Gefärbte Netze [Jen92], bieten hierfür geeignete Modellierungsmittel. Zusätzlich zu den Beschreibungsmöglichkeiten können aufgrund der mathematischen Basis von Petri–Netzen auch Analysen über statische und dynamische Netzeigenschaften, wie z. B. die Erreichbarkeit und Lebendigkeit, durchgeführt werden.

Eine Einschränkung der meisten Ansätze ist, daß sie zumeist keine explizite Unterstützung der für Kooperationsanwendungen notwendigen Dienstsicht bieten. Insbesondere die Dienstauswahl bzw. Diensttransparenz findet keine oder nur eine sehr rudimentäre Berücksichtigung (z. B. innerhalb von DOL durch a–priori Zuweisung von Diensterbringern zu einem Anwendungsvorgang). Diese Defizite bilden unter anderem die Grundlage für die in Abschnitt 3 vorgestellte Vorgangsbeschreibungssprache *PAMELA* der TRADE–Architektur, die ihrerseits auf einem höheren Petri–Netz–Modell basiert.

2.3 Ablaufkontrolle

Aufgabe der *Ablaufkontrolle* von Kooperationsanwendungen ist die Steuerung des gesamten Anwendungsvorganges während seiner Abarbeitung. Diese beeinhaltet im speziellen auch die Überwachung von Aufrufen externer Dienste innerhalb der Aktivitäten des Anwendungsvorganges. Der Aufruf externer Dienste erfordert neben den Kommunikationsaspekten vor allem auch die Anforderung der Dienste bei den verantwortlichen Dienstvermittlungskomponenten. Hierbei sind unter anderem systemspezifische Fehlerfälle, wie z. B. das Nichtvorhandensein eines Dienstes gemäß der Anforderungsdefinition, zu berücksichtigen, für die eine geeignete Fehlerbehandlung zu gewährleisten ist. Des weiteren muß die Ablaufkontrolle Mechanismen zur Behandlung von Benutzerinteraktionen innerhalb der Anwendungsvorgangsabwicklung bereitstellen.

Grundlage der Ablaufkontrolle sind die in der Vorgangsbeschreibungssprache definierten Kontrollkonstrukte, deren Ausdrucksmächtigkeit den Funktionsumfang der Ablaufkontrollkomponente bestimmt. Generell läßt sich bei der Abarbeitung zwischen einem *Generator*– und *Interpreteransatz* unterscheiden. Während im ersten Fall die Vorgangsbeschreibung als Grundlage für die Generierung der entsprechenden, vorgangsspezifischen Ablaufkontrollkomponente dient, wird beim Interpreteransatz diese Vorgangsbeschreibung einer generischen Ablaufkontrollkomponente übergeben, die interpretativ eine Abarbeitung des definierten Vorganges durchführt. Kritierien bei der Wahl einer geeigneten Ablaufkontrolle sind hierbei unter anderem die Effizienz und Flexibilität der Abarbeitung sowie die Generizität der Ablaufkontrollkomponente. Der innerhalb der TRADE–Architektur vorgestellte sog. *Vorgangsmanager* verfolgt einen hybriden Ansatz, der sowohl aus einer Generierungs– als auch aus einer Interpreterkomponente besteht.

Bekannte Beispiele für bisher existierende Ablaufkontrollkomponenten sind vor allem die TP–Monitore [Ber90, DGMH+93], die innerhalb der verteilten Transaktionsverarbeitung eine zentrale Rolle bei der Abwicklung verteilter Transaktionen spielen.

2.4 Dienstzugriff

Die Frage des Zugriffs externer Dienste während der Vorgangsbearbeitung hängt eng mit der Art der Verarbeitung der Ablaufkontrollkomponente ab. Bei einem Generatoransatz eignet sich der bekannte RPC–Ansatz für die Kommunikation der Ablaufkontrollkomponente mit dem externen Dienst bzw. Diensterbringer. Die mit dem RPC–Ansatz inhärent verbundene, sog. Stub–Generierung (aus Sicht der Ablaufkontrolle speziell die der Dienstnehmer–Stubs) läßt sich nahtlos mit der Generierung der Ablaufkontrollkomponente kombinieren. Da alle Dienstnehmer während eines Anwendungsvorganges bekannt sind, sind keine generischen Aufrufmechanismen notwendig, die das Generieren von Diensterbringeraufrufen zur Laufzeit erfordern.

Dieses ist jedoch im Interpreteransatz erforderlich, da aufgrund der Generizität der Ablaufkontrollkomponente die für die Abarbeitung von beliebigen Anwendungsvorgängen benötigten Dienste nicht a–priori vorhersehbar sind und dementsprechend bei der initialen Entwicklung der Ablaufkontrollkomponente nicht berücksichtigt werden können. Aus diesem Grund müssen die Diensterbringeraufrufe zur Laufzeit aufgrund der Vorgangsbeschreibung erzeugt werden. Ein Beispiel für eine derartige Kommunikationsabstraktion ist das sog. Dynamic Invocation Interface (DII), wie es in der CORBA–Architektur [OMG91] definiert ist.

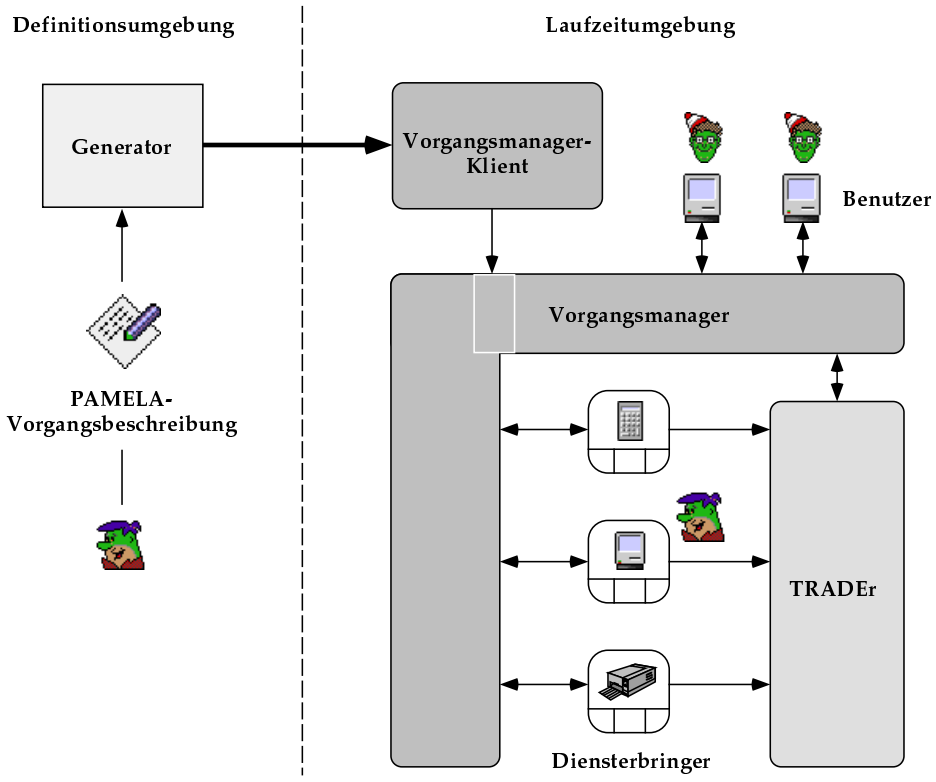


Abbildung 1: TRADE–Gesamtarchitektur

Eine generelle Frage beim Dienstzugriff ist, ob RPC–orientierte Dienste dennoch mit einem DII–orientierten Kommunikationsmechanismus verbindbar sind. Ein mögliche Antwort darauf bieten die in Abschnitt 3.2.3 im Rahmen der TRADE–Architektur eingeführten *Dienstagenten*.

3 Die TRADE–Architektur zur Unterstützung von Kooperationsanwendungen

Das Ziel der in diesem Abschnitt vorgestellten TRADE–Architektur ist die programmier– und systemtechnische Unterstützung von Kooperationsanwendungen in offenen verteilten Systemen. Die TRADE–Architektur ist Teil der an der Universität Hamburg laufenden COSM und TRADE Projekte [GGL+95, MJM94, MML94], welche als übergeordnetes Ziel die Schaffung einer Infrastruktur zur Nutzung der in einem offenen verteilten Dienstemarkt vorhandenen Dienste haben. Innerhalb der TRADE–Architektur sind hierbei insbesondere Kooperationsanwendungen und ihre Anforderungen an eine geeignete Systeminfrastruktur von Interesse. Abbildung 1 gibt einen ersten Überblick über die TRADE–Gesamtarchitektur.

Die Architektur läßt sich grob in eine *Definitions–* und in eine *Laufzeitumgebung* unterteilen. Während die Definitionsumgebung programmiertechnische Unterstützung für die Beschreibung von Anwendungsvorgängen mittels der PAMELA–Vorgangsbeschreibungssprache und einem dazugehörigen Programmgenerator bietet, behandelt die Laufzeitumgebung die

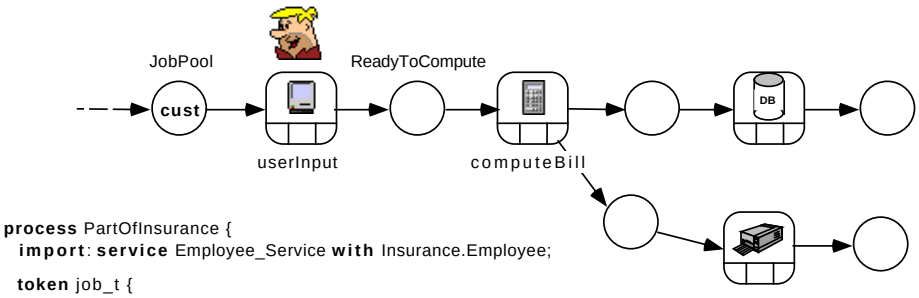
spezifischen systemtechnischen Unterstützungmechanismen, die zum Ablauf von Kooperationsanwendungen notwendig sind. Im einzelnen sind dieses der *Vorgangsmanager* zur Ablaufkontrolle, der *TRADEr* [MJML95, MJM94, MML95, MML94] zur Dienstvermittlung und –verwaltung und die in dieser Abbildung nicht explizit dargestellten Kommunikationsmechanismen zur Interaktion von Benutzern mit dem Vorgangsmanager und des Vorgangsmanagers mit den beim TRADEr registrierten, externen Diensten. Eine konkrete Kooperationsanwendung setzt sich aus dem Vorgangsmanager und dem Vorgangsmanager–Klienten zusammen. Der Vorgangsmanager–Klient wird aus der dazugehörigen PAMELA–Vorgangsbeschreibung als Laufzeitkomponente generiert. Die Komponenten und ihre Beziehungen untereinander werden nun im folgenden genauer beschrieben.

3.1 PAMELA–Vorgangsbeschreibungssprache

Vorgangsbeschreibungen innerhalb der TRADE–Architektur werden mittels der Vorgangsbeschreibungssprache *PAMELA* (*P*etri net based *A*ctivity *M*anagement *E*xecution *La*nguage) spezifiziert. PAMELA ist eine Sprache auf *hoher anwendungsspezifischer Abstraktionsebene* mit einem integriertem Dienstkonzept, welche speziell für die Beschreibung von Anwendungsvorgängen innerhalb von Kooperationsanwendungen nutzbar ist. Formale Grundlage von PAMELA bilden zeitgesteuerte gefärbte Petri–Netze (timed coloured petri–nets, TCPN), wie sie zum Beispiel in [Jen95] definiert sind. PAMELA bietet unter anderem die folgenden Konzepte:

- Sprachliche Umsetzung von TCPNs mit Erweiterungen hinsichtlich des Dienstkonzeptes. Dabei ist das Dienstkonzept für Berechnungs–Server und Benutzer, die innerhalb einer Aktivität aktiv angestoßen werden, identisch. Beide besitzen einen Diensttyp, wobei der Benutzer systemtechnisch durch das für den Anwendungsvorgang zugeschnittene Interaktionswerkzeug repräsentiert wird (verdeutlicht durch das Monitorsymbol in Abbildung 1),
- Einbettung der in offenen verteilten Systemen etablierten IDL–Definitionen von Diensterbringern, im speziellen der CORBA–IDL zur Definiton von Diensttypen (d. h. Schnittstellentypen, Operationstypen, Attributtypen und Datentypen),
- Möglichkeit der Spezifikation von Aktivitätsemantiken, wie z. B. synchrone, asynchrone und transaktionale Ausführung von Aktionen innerhalb von Aktivitäten,
- Darstellung parallel oder sequentiell auszuführender Aktionen innerhalb einer Aktivität,
- Beschreibung externer, synchroner und asynchroner Benutzerinteraktionen, die von außen an einen laufenden Anwendungsvorgang herangetragen werden (in der Abbildung 3 durch die mit dem Vorgangsmanager verknüpften Benutzer dargestellt) und
- Vererbungskonzept zur Wiederverwendung von vorhandenen PAMELA–Vorgangsbeschreibungen.

Ein Ziel des Sprachentwurfs von PAMELA ist hierbei die Anlehnung der Sprache an die bereits in der CORBA–IDL gewählten syntaktischen Konstrukte, die ähnlich der Programmiersprache C++ sind. Dieses betrifft sowohl Datentypdefinitionen als auch die Wahl der Kontrollkonstrukte, insbesondere aber auch die Art des Aufrufs externer Dienste in Form




```
process PartOfInsurance {
  import: service Employee_Service with Insurance.Employee;

  token job_t {
    long  job_nr;
    Insurance:policy_data_t policy;
  };
  places: job_t  JobPool, ReadyToCompute; ...

  [sync] trans userInput{
     get cust from JobPool;
     put cust to ReadyToCompute;
     action {
         Add_Insurance_Data ( cust.policy );
        using Employee_Service with { unit = Insurance && task = Billing};
     } ...
  } } //process
```

```
module Insurance { // CORBA IDL definitions
  typedef unit_t ...; typedef task_t ...;
  struct policy_data_t {
            customer_id customer;
            name_t       name;  ...
           } ;
interface Employee {
     readonly attribute unit_t unit;
     readonly attribute task_t task;
      Add_Insurance_Data ( [inout] policy_data_t customer);
   } //interface
} //module
```

Abbildung 2: Ausschnitt einer PAMELA–Vorgangsbeschreibung

von Operationen. Das hierbei verwendete Aktionskonzept in PAMELA ist angelehnt an das ebenfalls auf der Grundlage gefärbter Petri–Netze entwickelte LOOPN–System [Lak92].

Abbildung 2 gibt ein Beispiel für die Ausdrucksmöglichkeiten von PAMELA hinsichtlich der Beschreibung von Anwendungsvorgängen. Das Beispiel stellt einen stark vereinfachten Ausschnitt einer Versicherungsanwendung dar, bei der die Bearbeitung eines Versicherungsantrages zu modellieren und der Ablauf seiner Bearbeitung zu steuern ist. Im Verlaufe dieses Vorgangs werden mehrere externe Dienste in Anspruch genommen, wie z. B. der eines Sachbearbeiters zur Eingabe von Daten oder die Nutzung eines Druckers zur Ausfertigung einer Bestätigung für den Versicherungskunden. Die Darstellung des gesamten Ablaufs erfolgt innerhalb eines `process`–Blocks. Dieser umklammert die Definiton der Aktivitäten (`trans`), der Aktivitätsübergänge (`place`) und der Kontrollobjekte (`token`), welche von Aktivität zu Aktivität über die Aktivitätsübergänge weitergegeben werden. Die in der Abbildung dargestellte Aktivität *userInput* entnimmt hierbei ein Kontrollobjekt von dem Aktivitätsübergang *JobPool* (`get`–Anweisung) und führt die Aktion (`action`) *Add_Insurance_Data* mit den entsprechenden Argumentwerten aus. Nach erfolgreicher Ausführung der Aktion wird das um die entsprechenden Datenfelder ergänzte Kontrollobjekt an den *ReadyToCompute*–Aktivititätsübergang weitergereicht (`put`–Anweisung), welcher mit der *ComputeBill*–Aktivität verbunden ist.

Der Dienstaufruf *Add_Insurance_Data* wird qualifiziert durch den Diensttyp (*Employee_Service*) und die Dienstattribute (*unit = Insurance && task = Billing*), die den gewünschten Dienst (genauer: Diensterbringer) spezifizieren. Der Bezug des Diensttyps zu der Schnittstelle, welche die benutzte Operation anbietet, und den Dienstattributen wird mittels der `service`–Deklaration innerhalb der `import`–Anweisung des `process`–Blocks angegeben. Hierbei wird die Beziehung zur CORBA–IDL Definition der entsprechenden Schnittstelle (*Employee*) hergestellt. Diese explizite Verknüpfung des Diensttyps mit der dazugehörigen CORBA–Schnittstelle ist notwendig, da CORBA selbst kein explizites Diensttypkonzept besitzt, wie es innerhalb von PAMELA notwendig ist. Bei der Referenzierung müssen die

CORBA–IDL spezifischen Sichtbarkeitsregeln (Moduldefinitionen) berücksichtigt werden. Je nach Definition der Aktivität können prinzipiell auch mehrere Aktionen sequentiell bzw. parallel (durch hier nicht dargestellte `par`– und `seq`–Anweisungen) bei einem oder jeweils für jede Aktion separat spezifizierten Diensterbringer ausgeführt werden. Dieses wird durch entsprechende Schachtelung der `using`– und `action`–Anweisungen erreicht. Sollen innerhalb eines Anwendungsvorganges Aktionen, die innerhalb verschiedener Aktivitäten auftreten, bei demselben Diensterbringer ausgeführt werden, so muß dieses explizit durch die entsprechende Wahl der qualifizierenden Dienstattribute ausgedrückt werden. Hiermit wird ein Halten von Diensterbringerreferenzen über mehrere Aktivitäten hinweg erreicht.

Weitere, hier nicht dargestellte Ausdrucksmöglichkeiten von PAMELA sind die sog. *Task–Transitionen*, die Spezifikation einer *transaktionalen* Ausführungssemantik der Aktionen einer Aktivität und die Fehlerbehandlung.

Im Gegensatz zu den durch Dienstaufrufe initiierten externen Eingaben (z. B. durch die obige *userInput*–Aktivität) erlauben Task–Transitionen externe Eingaben in den Anwendungsvorgang unabhängig von seinem konkreten Abarbeitungszustand. In Bezug auf das obige Beispiel könnte eine Task–Transition *dispatchJobs* existieren, die asynchron neue Aufträge durch eine entsprechende Benutzereingabe in den durch *JobPool* spezifizierten Aktivitätsübergang einfügt.

Die Spezifikation einer Aktivität als transaktional (`[transactional] trans`) garantiert die Ausführung der spezifizierten Aktionen gemäß der bekannten ACID–Transaktionseigenschaften [GR93]. So lassen sich mehrere Aktionen zu einer Transaktion zusammenfassen, wobei je nach Ausgang der Transaktion (`commit`– oder `abort`–Anweisung) entsprechende Übergänge in anwendungsspezifische Aktivitätsübergänge definiert werden müssen.

Ähnlich der beiden `commit`– oder `abort`–Ausgänge einer Transaktion lassen sich für jede Aktivität mittels einer `error`–Anweisung aktionsspezifische Fehler behandeln. Dieses betrifft jedoch nur die mit der Dienstnutzung verbundenen Fehlerfälle (z. B. Nichtvorhandensein eines geeigneten Diensterbringers), Fehlerfälle aufgrund anwendungsspezifischer Resultatauswertungen eines Dienstaufrufes müssen bei der Definition des Anwendungsvorganges explizit modelliert werden.

3.2 Vorgangskontrolle: Der Vorgangsmanager

Der *Vorgangsmanager* (process manager) bildet den Kern der in Abbildung 3 detaillierter dargestellten TRADE–Gesamtarchitektur. Feiner dargestellte Pfeile in der Abbildung verdeutlichen die Interaktion von Systemkomponenten in der Laufzeitumgebung, Fett dargestellte Pfeile stellen die Verknüpfung der Definitionsumgebung mit der Laufzeitumgebung dar.

Der Vorgangsmanager bietet einen generischen *Ablaufkontrolldienst*, welcher die Bearbeitung einer konkreten Instanz eines in PAMELA spezifizierten Anwendungsvorganges durchführt. Intern verfügt der Vorgangsmanager über vier Schnittstellen:

- eine *Vorgangsgenerierungsschnittstelle*,
- eine *Dienstauswahlschnittstelle*,
- eine *Dienstaufrufschnittstelle* sowie
- eine *Benutzerinteraktionsschnittstelle*.

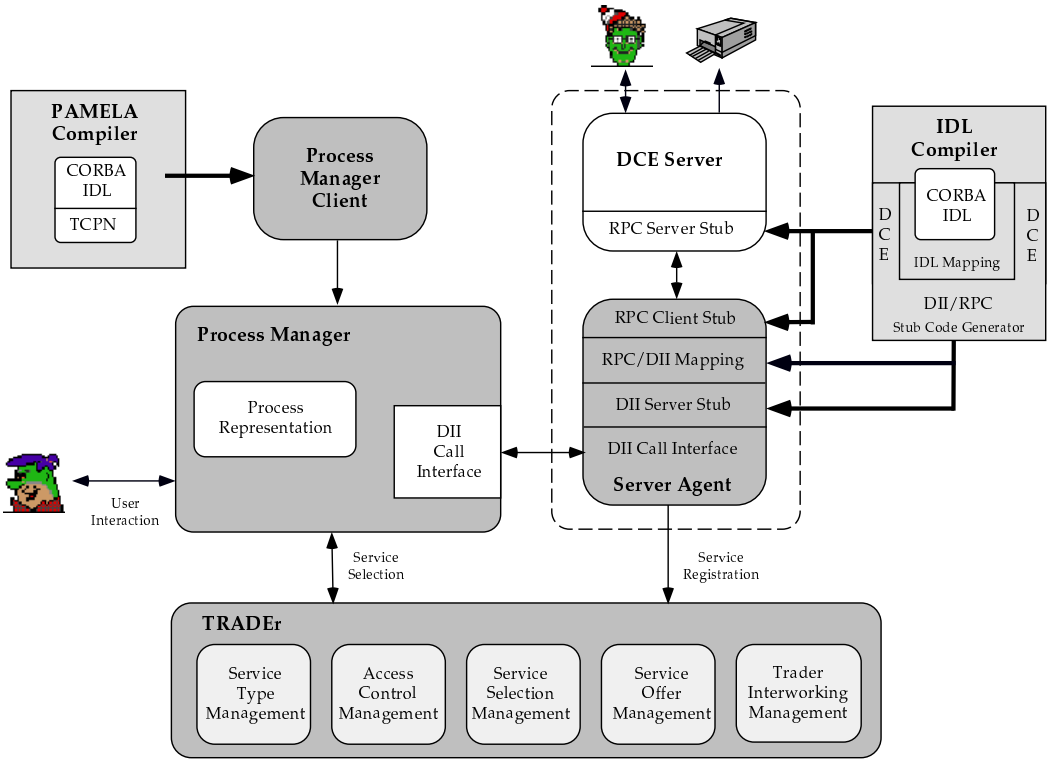


Abbildung 3: TRADE–Gesamtarchitektur

Im folgenden wird speziell auf die ersten drei Schnittstellen eingegangen, um die dienstspezifischen Aspekte der Vorgangsabarbeitung näher zu erläutern.

3.2.1 Vorgangsmanager–Klient

Die Vorgangsgenerierungsschnittstelle wird durch den *Vorgangsmanager–Klienten* (process manager client) zur Kreierung einer konkreten Instanz eines ablauffähigen Anwendungsvorganges genutzt. Der Vorgangsmanager–Klient selbst wird hierbei mittels des PAMELA–Sprachübersetzers erzeugt und ist seinerseits eine ablauffähige Komponente der TRADE–Laufzeitumgebung. Bei seinem Ablauf führt dieser Aufrufe an die Vorgangsgenerierungsschnittstelle des Vorgangsmanagers durch, wobei innerhalb des Vorgangsmanagers eine interne *Vorgangsrepräsentation* erzeugt wird. Ebenso erfolgt das Starten der Bearbeitung des kreierten Anwendungsvorganges.

3.2.2 Dienstauswahl

Im Verlaufe der Abarbeitung des Anwendungsvorganges nutzt der Vorgangsmanager die Dienstauswahlschnittstelle, um entsprechende Diensterbringer für die in den Aktvitäten definierten Dienstaufrufe bei der Dienstvermittlungskomponente, dem TRADEr, anzufordern. Die Dienstauswahlschnittstelle bietet hierfür die TRADEr–spezifische *Import–Anweisung*, die unter anderen die Angabe der Diensttypbeschreibung und einer beliebig komplexen Dienstattributanfrage erfordert. Für eine detaillierte Beschreibung der Aufgaben und Funktion des TRADErs sei hier auf [MJML95] verwiesen.

3.2.3 Dienstzugriff

Der eigentliche Dienstzugriff erfolgt nun mittels der dynamischen Dienstaufrufschnittstelle, dem sog. *Dynamic Invocation Interface* (DII), welches eine dynamische Generierung von Operationsaufrufen zur Laufzeit ermöglicht. Hierbei wird gemäß des in der Vorgangsbeschreibung spezifizierten Dienstaufrufes die entsprechende Operation mit ihren Argumenten und Rückgabeparametern erzeugt und der Aufruf bei dem Diensterbringer ausgeführt.

Dieser Operationsaufruf wird einem sog. *Dienstagenten* übermittelt, der jedem RPC–orientierten Diensterbringer eindeutig zugeordnet ist. Der Dienstagent übernimmt hierbei die Abbildung des DII–Operationsaufrufes in den entsprechenden RPC–Aufruf des RPC-orientierten Diensterbringers. Der Dienstagent selbst kann weitgehend aus der Diensttypbeschreibung generiert werden. Abbildung 3 stellt einige Generierungsdetails dar. Der IDL–Sprachübersetzer führt bei der Generierung auch gleichzeitig eine Abbildung der innerhalb der TRADE–Umgebung verwendeten CORBA–IDL Diensttypbeschreibungen auf die DCE–IDL Diensttypbeschreibungen der konkreten DCE–Diensterbringer der TRADE–Prototypumgebung durch.

4 TRADE–Prototypimplementierung

Ein Ziel der Prototypimplementierung ist die weitgehende Nutzung vorhandener, verteilter generischer Systemdienste, die eine effiziente Implementierung der TRADE–Systemkomponenten erlauben. Grundlage der Implementierung auf Basis von IBM RS/6000 Arbeitsplatzrechnern bildet die verteilte *DCE*–Entwicklungs– und Laufzeitumgebung [Fou93], welche unter anderem als Kernbestandteil einen RPC–Dienst anbietet. Zusätzlich wird das auf DCE aufsetzende *ENCINA*–System [IBM93] benutzt, welches die Ausführung verteilter Transaktionen unterstützt. Je nach Semantik der Aktionsausführung wird entsprechend der Standard–RPC oder der transaktionale RPC (TRPC) im Vorgangsmanager verwendet. Die Interaktion der TRADE–Systemkomponenten (Vorgangsmanager, Vorgangsmanager–Klient und TRADEr) untereinander erfolgt ebenfalls unter Verwendung des DCE–RPCs.

Die Realisierung des PAMELA–Sprachübersetzers bzw. von Teilen der Dienstagentengenerierung erfolgt mittels der Standardwerkzeuge LEX und YACC, die innerhalb der Unix–Umgebung verfügbar sind.

Die Prototypentwicklung befindet sich zur Zeit in ihrer zweiten Entwicklungsphase, in der die Konzeption und Realisierung des Vorgangsmanagers bzw. die Generierung des Vorgangsmanager–Klienten und der Dienstagenten im Vordergrund steht. Teilkomponenten, wie z. B. die Sprachübersetzer und das DII, stehen bereits zur Verfügung. In der ersten Entwicklungsphase wurde bereits das der TRADE–Architektur unterliegende Dienstkonzept entwickelt und eine entsprechende Dienstvermittlungskomponente, der TRADEr, entworfen und implementiert [MJML95, MJM94, MML95, MML94].

5 Zusammenfassung und Ausblick

Kooperationsanwendungen stellen zum Teil neuartige programmier– und systemtechnische Anforderungen, die bisher in vorhandenen Systemen meist nur sehr eingeschränkt für spezifische Anwendungsbereiche isoliert voneinander entwickelt wurden (siehe Abschnitt 2). Diese

Anforderungen und auch die Einschränkungen heutiger Systeme bilden die Grundlage für die im vorherigen Abschnitt vorgestellte TRADE–Architektur. Die in dieser Architektur neu entwickelte Vorgangsbeschreibungssprache PAMELA verdeutlicht hierbei den neuartigen Ansatz der Integration einer für Kooperationsanwendungen typischen vorgangsorientierten Sichtweise mit der in offenen verteilten Systemen verbreiteten Dienstsicht. PAMELA bietet eine Sprachebene, die sich auf der hohen anwendungsspezifischen Abstraktionsebene der Kooperationsanwendungen befindet, in dem sie kontrollspezifische Aspekte der Vorgangsbearbeitung mit einer dienstorientierten Aktionsausführung verbindet. Die Komplexität der Dienstauswahl und des Dienstzugriffs mit ihren spezifischen Fehlerfällen wird dabei weitgehend vor dem Programmierer von Kooperationsanwendungen verborgen.

Für die Beschreibung der Kontrollaspekte von Kooperationsanwendungen bieten hierbei die zeitgesteuerten gefärbten Petri–Netze eine geeignete Ausdrucksmächtigkeit, deren mathematische Fundierung gleichzeitig auch als Grundlage für Analysen der Eigenschaften definierter Anwendungsvorgänge dienen kann.

Auf systemtechnischer Seite der Unterstützung von Kooperationsanwendungen sind insbesondere die Dienstvermittlungskomponente, der TRADEr, und die Ablaufkontrollkomponente, der Vorgangsmanager, hervorzuheben. Diese stellen die notwendigen Laufzeitmechanismen bereit, die die Bearbeitung der in PAMELA spezifizierten Anwendungsvorgänge durchführen.

Neben systemtechnischen Fragen, wie z. B. der transaktionalen Ausführung von Aktionen, sind zukünftig vor allem auch Erweiterungen der sprachlichen Ausdrucksmöglichkeiten bei der Dienstanforderung im Rahmen der PAMELA–Vorgangsbeschreibung zu untersuchen. Dies betrifft insbesondere die Dienstqualifizierung, um den für die Ausführung einer Aktion *am besten geeigneten* Diensterbringer zu erhalten. Hierbei sind die Randbedingungen zu berücksichtigen, die sich aus den Kooperationsfähigkeiten verschiedener Dienstvermittlungskomponenten (das sog. Trader–Interworking [ISO94, Vog94]) ergeben. Durch diese Kooperation lassen sich die im offenen verteilten System befindlichen Dienste wesentlich strukturierter verwalten und vermitteln. Gleichzeitig erfordern diese erweiterten Suchmöglichkeiten jedoch auch eine Erweiterung der bisherigen Ausdrucksmöglichkeiten von Dienstanforderungen, um z. B. den Suchbereich einer Anfrage lokal einzuschränken.

Literatur

[Ber90] P.A. Bernstein. Transaction processing monitors. *Communications of the ACM*, Band 33, Heft 11, S. 75–88, November 1990.

[DGMH+93] U. Dayal, H. Garcia-Molina, M. Hsu, B. Kao und M. Shan. Third Generation TP–Monitors: A Database Challenge. *SIGMOD RECORD*, Band 22, Heft 2, S. 393–397, Juni 1993.

[Fou93] Open Software Foundation. *OSF DCE Application Development Guide*. Prentice–Hall, Inc., Englewood Cliffs, New Jersey, 1993.

[Gen87] H.J. Genrich. Predicate/transition nets. In W. Brauer, W. Reisig und G. Rozenberg, Hrsg., *Petri Nets: Central Models and Their Properties, Advances in Petri Nets 1986 Part I*, Band 254 of *Lecture Notes in Computer Science*, S. 207–247. Springer–Verlag, 1987.

[GGL+95] K. Geihs, H. Gründer, W. Lamersdorf, M. Merz, K. Müller und A. Puder. Systemunterstützung für offene verteilte Dienstemärkte. In *Kommunikation in Verteilten Systemen (KIVS '95)*, Februar 1995. to appear, in german.

[GR93] J. Gray und A. Reuter. *Transaction Processing: Concepts and Techniques.* Morgan–Kaufmann, San Mateo, California, 1993.

[HAB+92] Y. Halabi, M. Ansari, R. Batra, W. Jin, G. Karabatis, P. Krychniak, M. Rusinkiewicz und L. Suardi. Narada: An environment for specification and execution of multi–system applications. In *Proceedings of the 2nd International Conference on Systems Integration ICSI '92*, S. 680–690. IEEE, Juni 1992.

[IBM93] IBM. *Encina for AIX/6000 Base Reference.* IBM, 1993. Order Number SC23–2464.

[ISO94] ISO/IEC JTC 1/SC 21. Recommendation X.9tr/Draft ODP Trading Function ISO/IEC 13235: 1994/Draft ODP Trading Function, Juli 1994. ISO/IEC JTC 1/SC 21 N9122.

[Jen92] K. Jensen. *Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use*, Band 1. Springer–Verlag, 1992.

[Jen95] K. Jensen. *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use. Volume 2.* EATCS Monographs on Theoretical Computer Science. Springer–Verlag, 1995.

[Kel93] L. Keller. Vom Name–Server zum Trader – Ein Überblick über Trading in verteilten Systemen. *Praxis der Informationsverarbeitung und Kommunikation (PIK)*, Band 16, Heft 3, S. 122–133, 1993.

[Lak92] C. Lakos. The LOOPN User Manual. Technical report TR 92–1, Department of Computer Science, University of Tasmania, 1992.

[LR94] F. Leymann und D. Roller. Business process management with flowmark. In *Proceedings COMPCON Spring 94*, San Francisco, CA., Februar 1994. IEEE.

[MJM94] K. Müller, K. Jones und M. Merz. Vermittlung und Verwaltung von Diensten in offenen verteilten Systemen. In B. Wolfinger, Hrsg., *Innovationen bei Rechen– und Kommunikationssystemen — Eine Herausforderung für die Informatik*, Informatik Aktuell, S. 219–226, 24. GI–Jahrestagung within the scope of the 13th World Computer Congress IFIP Congress '94, Hamburg, Germany, August 1994. Springer–Verlag. in german.

[MJML95] K. Müller-Jones, M. Merz und W. Lamersdorf. The TRADEr: Integrating Trading Into DCE. accepted for publication in: Third International Conference on Open Distributed Processing (ICODP '95), Februar 1995.

[MML94] M. Merz, K. Müller und W. Lamersdorf. Service trading and mediation in distributed computing environments. In *Proceedings of the 14th International Conference on Distributed Computing Systems (ICDCS '94)*, S. 450–457. IEEE Computer Society Press, 1994.

[MML95] K. Müller, M. Merz und W. Lamersdorf. Der TRADE–Trader: Ein Basisdienst offener verteilter Systeme. In C. Popien und B. Meyer, Hrsg., *Neue Konzepte für die Offene Verteilte Verarbeitung*, S. 35–44. Verlag der Augustinus Buchhandlung, Aachen, Germany, September 1995. in german.

[Mur89] T. Murata. Petri nets: Properties, analysis and applications. In *Proceedings of the IEEE*, Band 77, S. 541–580, 1989.

[ODP93] ISO/IEC JTC1/SC21/WG7: Basic Reference Model of Open Distributed Processing — Part 1: Overview and Guide to Use. International Standardization Organization, August 1993.

[OMG91] The Common Object Request Broker: Architecture and Specification. Digital Equipment Corporation, Hewlett–Packard Company, HyperDesk Corporation, NCR Corporation, Object Design Incorporated, SunSoft Incorporated, 1991.

[Pet81] J.L. Peterson. *Petri Net Theory and the Modeling of Systems.* Prentice–Hall, 1981.

[Rei93] B. Reinwald. *Workflow–Management in verteilten Systemen.* B.G. Teubner Verlagsgesellschaft, 1993.

[Vog94] A. Vogel. Towards the Implementation of Interworking of Traders. verfügbar via WWW: http://www.dstc.edu.au/public/iwt/welcome.html, 1994.

[WR91] H. Wächter und A. Reuter. The ConTract Model. In A.K. Elmagarmid, Hrsg., *Database Transaction Models for Advanced Applications*, S. 219–263. Morgan Kaufmann Publishers, 1991.