Realisierung von Kooperationsanwendungen auf der Basis erweiterter Diensttypbeschreibungen

K. Müller–Jones, M. Merz, W. Lamersdorf
Universität Hamburg, Fachbereich Informatik, Vogt–Kölln–Str. 30, D–22527 Hamburg
kmueller@dbis1.informatik.uni-hamburg.de

Zusammenfassung

Angesichts der Anzahl und Vielfalt vorhandener Dienste in offenen verteilten Systemen gewinnt deren möglichst einfache Nutzung innerhalb verteilter Anwendungen – hier *Kooperationsanwendungen* genannt – zunehmend an Bedeutung. Um darüberhinaus komplexe verteilter Anwendungen auch effizient unter Verwendung derartiger Dienste realisieren zu können, sind neuartige Abstraktionen notwendig, die eine möglichst nahtlose Integration mit den in verteilten Systemen vorhandenen Konzepten bereitstellen. Dieses betrifft im speziellen auch die Nutzung bereits vorhandener, in sog. Schnittstellenbeschreibungssprachen (IDL) spezifizierter Dienstbeschreibungen, die die Grundlage für die Dienstnutzung darstellen.
Dieser Beitrag zeigt anhand der vorgestellten TRADE–Architektur, wie eine derartige Integration in einer offenen verteilten Systemumgebung erfolgen kann. Hierbei wird insbesondere auf die neuartige Vorgangsbeschreibungssprache PAMELA eingegangen, die die Aspekte der Dienstnutzung mit einer für Kooperationsanwendungen typischen vorgangsorientierten Verarbeitungssicht verbindet und eine weitgehende Einbettung bereits vorhandener Dienstbeschreibungen erlaubt.

1 Kooperationsanwendungen in einem offenen Dienstemarkt

Die Nutzung bereits in offenen verteilten Systemen vorhandener Dienste gewinnt bei der Entwicklung neuer verteilter Anwendungen immer mehr an Bedeutung. Aufgrund der zu erwartenden und zum Teil heute schon vorhandenen Anzahl und Vielfalt von Diensten in einem *offenen Dienstemarkt* [MML94] rückt somit die Vision eines weltweit verfügbaren *Systembaukastens* von Diensten in greifbare Nähe. Dieser bildet die notwendige Voraussetzung für die Entwicklung und Realisierung komplexer verteilter Anwendungen unter *weitgehender Wiederverwendung und Kombination* der bereits im verteilten System angebotenen Dienste. 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 der Kontroll– bzw. Kooperationsaspekte verteilter Anwendungen wer-

den 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 *Aktivitäten*, die eine oder mehrere *Aktionen* unter Nutzung externer Dienste ausführen können.
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, der verteilten Transaktionsverarbeitung und der computergestützen Fertigung (CIM).
Der vorliegende Beitrag gibt im folgenden einen Überblick über Probleme der systemtechnischen Unterstützung von Kooperationsanwendungen und weist auf schon vorhandene Lösungsansätze hin. Es wird gezeigt, wie diese größtenteils unabhängig 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 eine entsprechende Prototypimplementierung eingegangen, die zur Zeit an der Universität Hamburg durchgeführt wird.

2 Unterstützung von Kooperationsanwendungen

Im folgenden werden die Probleme, mit denen der Programmierer bei der Realisierung von Kooperationsanwendungen in offenen verteilten Systemen konfrontiert ist, kurz untersucht. Dabei wird — soweit möglich — auf bereits existierende Lösungsansätze hingewiesen, und es werden Beziehungen zur im nachfolgenden Abschnitt beschriebenen TRADE–Architektur aufgezeigt.

Dienstvermittlung und –verwaltung: Notwendige Voraussetzung für die kontrollierte Vermittlung und Verwaltung von Diensten in offenen Systemen ist das Vorliegen eines formalisierten Dienst(typ)begriffes, der eine Klassifikation von Diensten aufgrund wohldefinierter Kategorisierungs– und Beschreibungsmittel erlaubt. Ein Beispiel dafür ist der *Diensttypbegriff*, wie er im Rahmen der internationalen Standardisierung 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 semantisch 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 der Grundlage derartiger Diensttypbeschreibungen können nun generische Systemdienste, sog. *Trader* [ISO94], eine Dienstvermittlung und –verwaltung realisieren. Die Dienstvermittlung ist ein wichtiger Bestandteil der systemtechnischen Unterstützung von Kooperationsanwendungen innerhalb der TRADE–Architektur.

Beschreibung von Anwendungsvorgängen: Die Koordination der Nutzung externer Dienste eines offenen verteilten Systems innerhalb eines komplexen Anwendungsvorganges stellt einen zentralen Aspekt von Kooperationsanwendungen dar. Zur Beschreibung von Aktivitätsabfolgen müssen geeignete formale Hilfsmittel gewählt werden, die sowohl eine Darstellung der Ablaufstrukturen als auch die externe Dienstnutzung adäquat auszudrücken vermögen. Hierbei sollten auch verschiedene Aufrufsemantiken, z. B. die transaktionale Ausführung mehrerer 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. textuell oder grafisch) sollte auf möglichst hoher Abstraktionsebene angesiedelt sein, um eine Trennung der Koordinationsaspekte innerhalb von Anwendungsvorgängen von der eigentlichen Ausführung der Aktivitäten zu ermöglichen. Dabei sollten verteilungspezifische Aspekte, die sich durch den Aufruf externer Dienste ergeben, weitgehend vor dem Programmierer verborgen bleiben. Dies erfordert unter anderem entsprechende Ausdrucksmöglichkeiten bei der Dienstauswahl, um eine möglichst große Diensttransparenz zu erreichen.
Konkrete Ansätze zur Beschreibung von Anwendungsvorgängen finden sich vor allem in dem Anwendungsbereich des Workflow–Managements und der Transaktionsverarbeitung. Beispiele hierfür sind die Aktivitätenbeschreibungssprache *ActSpec* [Rei93] sowie die *Distributed Operation Language* (DOL) [HAB+92]. Zur Repräsentation der Anwendungsvorgänge werden oftmals, z. B. im *ConTract*–Modell [WR91] oder im *FlowMark*–System [LR94], Petri–Netz verwandte Modelle verwendet, die auf höheren Petri–Netzen, wie z. B. den *Gefärbten Petri–Netzen* [Jen92] basieren.
Eine Einschränkung der meisten vorhanden Ansätze ist jedoch, 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 der 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 ebenfalls auf einem höheren Petri–Netz–Modell basiert.

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 benötigten Dienstes, 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 Vorgangsbeschrei-

bung 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 [DGMH+93], die innerhalb der verteilten Transaktionsverarbeitung eine zentrale Rolle bei der Abwicklung verteilter Transaktionen spielen.

Dienstzugriff: Die Frage des Zugriffs auf externe Dienste während der Vorgangsbearbeitung hängt eng mit der Art der Verarbeitung der Ablaufkontrollkomponente zusammen. 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 im Voraus bekannt sind, sind keine generischen Aufrufmechanismen notwendig, die das Generieren von Diensterbringeraufrufen zur Laufzeit erfordern. Dieses ist jedoch beim 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.
Eine generelle Frage beim Dienstzugriff ist, ob RPC–orientierte Dienste dennoch mit einem DII–orientierten Kommunikationsmechanismus verbindbar sind. Ein mögliche Antwort darauf geben die in Abschnitt 3.2 im Rahmen der TRADE–Architektur eingeführten *Dienstagenten*.

3 Die TRADE–Architektur zur Unterstützung von Kooperationsanwendungen

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 Projekte TRADE und COSM an der Universität Hamburg, welche als übergeordnetes Ziel die Schaffung einer Infrastruktur zur Nutzung der in einem offenen verteilten Dienstemarkt vorhandenen Dienste haben [GGL+95, MJM94, MML94].
Die Architektur läßt sich grob in eine *Definitions–* und in eine *Laufzeitumgebung* unterteilen (Abbildung 1). 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 Unterstützungmecha-

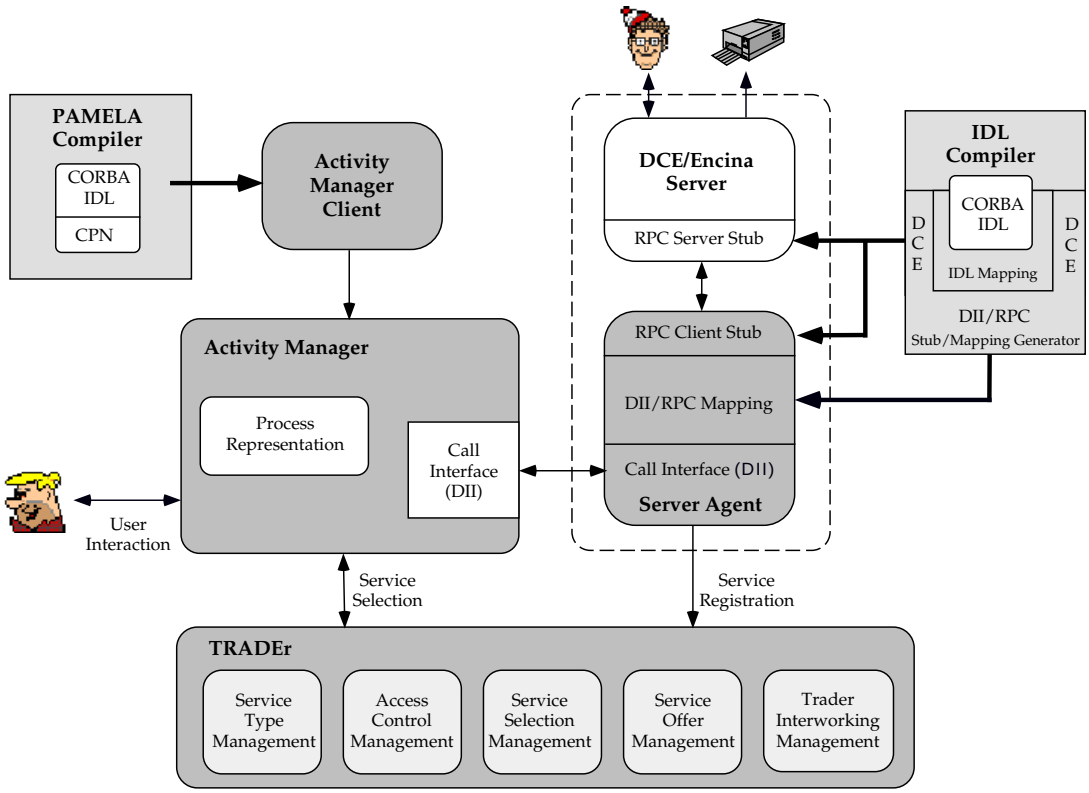


Abbildung 1: TRADE–Gesamtarchitektur

nismen, die zum Ablauf von Kooperationsanwendungen notwendig sind. 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.

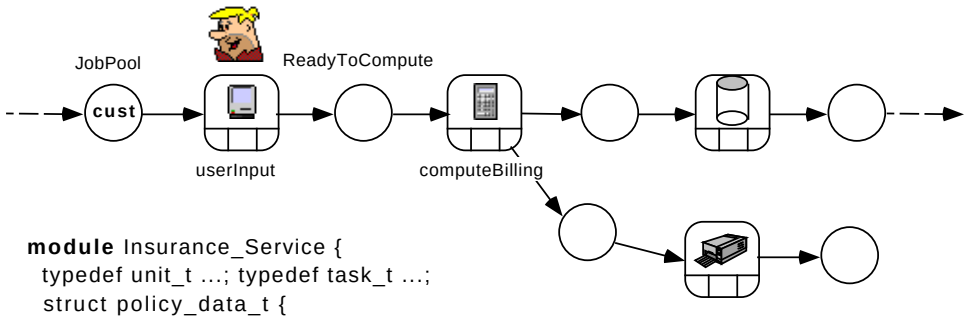
3.1 PAMELA–Vorgangsbeschreibungssprache

Vorgangsbeschreibungen innerhalb der TRADE–Architektur werden mittels der Beschreibungssprache *PAMELA* (*P*etri net based *A*ctivity *M*anagement *E*xecution *La*nguage) spezifiziert. PAMELA ist als Sprache auf *hoher anwendungsspezifischer Abstraktionsebene* mit integriertem Dienstkonzept speziell für die Beschreibung von Anwendungsvorgängen innerhalb von Kooperationsanwendungen nutzbar. Formale Grundlage von PAMELA bilden gefärbte Petri–Netze (coloured petri–nets, CPN), wie sie zum Beispiel in [Jen92] definiert sind. PAMELA ermöglicht damit unter anderem:

- die sprachliche Umsetzung von CPNs mit Erweiterungen hinsichtlich des Dienstkonzeptes. Dabei ist das Dienstkonzept für Berechnungs–Server und Benutzer, die innerhalb einer Aktivität aktiv angestoßen werden, identisch;
- die Einbettung der in offenen verteilten Systemen etablierten IDL–Definitionen von Diensterbringern, im speziellen der CORBA–IDL zur Definition von Diensttypen (d. h. Schnittstellentypen, Operationstypen, Attributtypen und Datentypen),
- die Spezifikation von Aktivitätsemantiken, wie z. B. synchrone, asynchrone und transaktionale Ausführung von Aktionen innerhalb von Aktivitäten,

- die Darstellung parallel oder sequentiell auszuführender Aktionen innerhalb einer Aktivität und
- die Beschreibung externer, synchroner und asynchroner Benutzerinteraktionen, die von außen an einen laufenden Anwendungsvorgang herangetragen werden.

Ein Ziel des Sprachentwurfs von PAMELA ist die Anlehnung der Sprache an die bereits in der CORBA–IDL gewählten syntaktischen Konstrukte, die der Programmiersprache C++ ähneln. Dieses betrifft sowohl Datentypdefinitionen als auch die Wahl der Kontrollkonstrukte, insbesondere aber die Art des Aufrufs externer Dienste in Form von Operationen. Das hierbei verwendete Aktionskonzept in PAMELA ist angelehnt an das ebenfalls auf der Grundlage gefärbter Petri–Netze entwickelte *LOOPN* [Lak92].




```
module Insurance_Service {
 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);
  }; //CORBA interface
}; //CORBA module

process PartOfInsurance {
 import: service Insurance_Service with 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 with {unit = Insurance && task = Billing};
    } ...
 } } //process
```

Abbildung 2: Ausschnitt einer PAMELA–Vorgangsbeschreibung

Abbildung 2 zeigt am Beispiel eines vereinfachten Ausschnitts einer Versicherungsanwendung die Ausdrucksmöglichkeiten von PAMELA hinsichtlich der Beschreibung von Anwendungsvorgängen. 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.

Der Ablauf wird innerhalb eines `process`–Blocks dargestellt. Dieser klammert die Definition der Aktivitäten (`trans`), der Aktivitätsübergänge (`place`) und der Kontrollobjekte (`token`), welche von Aktivität zu Aktivität über die Aktivitätsübergänge weitergegeben werden. Die in der Abbildung beschriebene 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 Daten ergänzte Kontrollobjekt an den *ReadyToCompute*–Aktivititätsübergang weitergereicht (`put`–Anweisung), welcher mit der *ComputeBill*–Aktivität verbunden ist.
Der Operationsaufruf *Add_Insurance_Data* wird qualifiziert durch den dazugehörigen Schnittstellentyp (*Employee*) und die Dienstattribute (*unit = Insurance && task = Billing*), die den gewünschten Dienst genauer spezifizieren. Der Bezug des Schnittstellentyps zu dem dazugehörigen Diensttyp (*Insurance_Service*) wird mittels der `service`–Deklaration innerhalb der `import`–Anweisung des `process`–Blocks angegeben. Hierbei wird die Beziehung der in der CORBA–IDL definierten Schnittstelle (`interface` *Employee*) mit der diese Schnittstelle umfassenden CORBA–Moduldefinition (`module` *Insurance_Service*) hergestellt. Hierbei findet ein semantisches Überladen des CORBA–Modulkonzeptes statt, in dem der Modulname als Diensttypname innerhalb von PAMELA und auch für die Dienstvermittlung durch den TRADEr verwendet wird.
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.
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* bildet neben dem TRADEr den Kern der TRADE–Laufzeitumgebung. Er 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*, wobei im folgenden speziell auf die ersten drei Schnittstellen eingegangen wird, um die dienstspezifischen Aspekte der Vorgangsabarbeitung näher zu erläutern.
Die Vorgangsgenerierungsschnittstelle wird durch den *Vorgangsmanager–Klienten* 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* (process representation) erzeugt wird. Ebenso erfolgt das Starten der Bearbeitung des kreierten Anwendungsvorganges.
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, MJM94] verwiesen.
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. Er selbst kann weitgehend aus der Diensttypbeschreibung generiert werden. Abbildung 1 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/Encina–Diensterbringer der TRADE–Prototypumgebung durch.

3.3 Aspekte der 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 bildet die verteilte *DCE*–Entwicklungs– und Laufzeitumgebung [Fou93]. Zusätzlich wird das auf DCE aufsetzende *Encina*–System [IBM93] benutzt, welches die Ausführung verteilter Transaktionen unterstützt.
Sämtliche Interaktionen der TRADE–Systemkomponenten (Vorgangsmanager, Vorgangsmanager–Klient, Diensterbringer, Dienstagenten und TRADEr) untereinander erfolgen unter Verwendung der durch DCE und Encina angebotenen RPC–Kommunikationsmechanismen. Der TRADEr als Dienstvermittlungskomponente ist hierbei weitgehend in die DCE–Systemumgebung integriert [MJML95].
Die Kommunikation des Vorgangsmanagers mit den jeweiligen Dienstagenten wird mittels einer dynamischen Aufrufschnittstelle (dynamic invocation interface, DII) durchgeführt, die ein Kreieren von Diensterbringeraufrufen zur Laufzeit ermöglicht. Je nach Semantik der Aktionsausführung wird dann entsprechend der herkömmliche RPC oder der transaktionale RPC (TRPC) als unterliegender Kommunikationsmechanismus verwendet.

4 Zusammenfassung und Ausblick

Kooperationsanwendungen stellen zum Teil neuartige programmier– und systemtechnische Anforderungen, die bisher meist nur beschränkt für spezifische Anwendungsbereiche und isoliert voneinander entwickelt worden sind. Die vorgestellte, auf Petri–Netzen basierende Vorgangsbeschreibungssprache PAMELA realisiert einen Ansatz zur Integration einer für Kooperationsanwendungen typischen vorgangsorientierten Sichtweise mit der in offenen verteilten Systemen verbreiteten Dienstsicht. PAMELA ist auf der anwendungsspezifischen Abstraktionsebene der Kooperationsanwendungen definiert und verbindet kontrollspezifische Aspekte der Vorgangsbearbeitung mit einer dienstorientierten Aktionsausführung. Dabei können bei der Definition von Vorgangsbeschreibungen die schon oftmals vorliegenden (CORBA–)IDL–Dienstbeschreibungen durch semantische Ergänzungen weitgehend integriert werden. Die Komplexität der Dienstauswahl und des Dienstzugriffs mit ihren spezifischen Fehlerfällen wird dabei weitgehend vor dem Programmierer von Kooperationsanwendungen verborgen. Eine geeignete systemtechnische Unterstützung hierfür wird durch die Dienstvermittlungskomponente, den TRADEr, und die Ablaufkontrollkomponente, den Vorgangsmanager, bereitgestellt.
Zur Zeit werden weitergehende Arbeiten durchgeführt, welche die Ausdrucksmächtigkeit des jetzigen Typmanagements innerhalb des TRADE–Architektur hinsichtlich der Beschreibung von Diensttypen auf Basis der semantisch erweiterten CORBA–IDL und der Repräsentation von Typinformationen innerhalb der Laufzeitumgebung erhöhen. Weiterhin sollen erweiterte Typvergleichsmechanismen bereitgestellt werden, die die Dienstauswahl durch den TRADEr noch flexibler gestalten. Hierbei werden insbesondere die Randbedingungen berücksichtigt, die sich aus den erweiterten Möglichkeiten der Dienstvermittlung durch das sog. Trader–Interworking [VBB95] und den daraus folgenden Interoperabilitäts– bzw. Heterogenitätsproblemen ergeben.

Literatur

[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.

[GGL+95] K. Geihs, H. Gründer, W. Lamersdorf, M. Merz, K. Müller und A. Puder. Systemunterstützung für offene verteilte Dienstemärkte. In K. Franke, U. Hübner und W. Kalfa, Hrsg., *Kommunikation in Verteilten Systemen: Neue Länder – Neue Netze – Neue Dienste*, Informatik–Aktuell, S. 445–459. Springer–Verlag Berlin Heidelberg New York, Februar 1995.

[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.

[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. Springer–Verlag, August 1994.

[MJML95] K. Müller-Jones, M. Merz und W. Lamersdorf. The TRADEr: Integrating trading into DCE. In *Proceedings of the Third International Conference on Open Distributed Processing (ICODP '95)*, Brisbane, Australia, 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.

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

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

[VBB95] A. Vogel, M. Bearman und A. Beitz. Enabling interworking of traders. In *Proceedings of the Third International Conference on Open Distributed Processing (ICODP '95)*, Brisbane, Australia, Februar 1995.

[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.