Systemunterstützung für offene verteilte Dienstemärkte

K. Geihs, H. Gründer, A. Puder

Fachbereich Informatik, J. W. Goethe-Universität
Postfach 11 19 32, D-60054 Frankfurt
{geihs,gruender,puder}@informatik.uni-frankfurt.de

W. Lamersdorf, M. Merz, K. Müller

Fachbereich Informatik, Universität Hamburg
Vogt–Kölln-Str. 30, D-22527 Hamburg
{lamersd,merz,kmueller}@dbis1.informatik.uni-hamburg.de

Abstract. Mit der globalen Vernetzung von Rechnersystemen entsteht ein wachsendes Potential für eine Vielzahl spezialisierter Online–Dienste, welche sowohl Endbenutzern als auch Software–Anwendungen zur Verfügung stehen. Derartige offene verteilte Dienstemärkte sind insbesondere durch ihre Vielfalt des Dienstangebotes und die Dynamik der dort stattfindenden Nachfrager-Anbieter-Beziehungen gekennzeichnet. Dieser Artikel analysiert die Anforderungen einer solchen Umgebung und schlägt ein Architekturmodell vor, welches eine adäquate Grundlage für die Spezifikation, die Vermittlung und die Benutzung solcher Dienste bietet. Insbesondere Systemdienste zur Vermittlung und zum Zugriff auf Dienstangebote bilden einen Schwerpunkt der Betrachtung. Dabei werden auch konkrete Lösungsansätze vorgestellt, die zur Zeit in den am vorliegenden Beitrag beteiligten Arbeitsgruppen verfolgt und prototypisch implementiert werden.

1 Einleitung

Sowohl die Dezentralisierung der Informationsverarbeitung als auch der Zusammenschluß zuvor isolierter Netzwerkdomänen führt zu einer nach wie vor wachsenden Kommunikationsinfrastruktur. Die steigende Leistungsfähigkeit von Rechner– und Kommunikationstechnologie ermöglicht es, immer mehr rechnergestützte Dienste über Kommunikationsnetze direkt zugreifbar zu machen. Es entstehen damit offene verteilte Dienstlandschaften mit einer Vielzahl von Dienstarten, Dienstanbietern und Dienstnachfragern. Neben dieser Vielfalt ist dabei z. B. die Dynamik des Angebots charakteristisch für solche Dienstemärkte: Wir gehen z. B. davon aus, daß das Dienstangebot einer ständigen Fluktuation unterliegt, daß Dienstanbieter temporär nicht verfügbar sein können und daß

bei Dienstanfragen in der Regel erst zur Laufzeit eine Bindung zwischen dem Klienten und einem geeigneten Dienstanbieter erfolgt.

Zur Realisierung solcher Dienstemärkte reicht die heute verfügbare Technologie verteilter Systeme jedoch nicht aus. Diese ist gekennzeichnet durch die Annahme, daß die Menge der verfügbaren *Diensttypen* im wesentlichen bereits zur Übersetzungszeit von Software–Komponenten den Entwicklern bekannt und somit eine Bindung zwischen diesen Komponenten relativ statisch ist. Aufgrund von Vielfalt und Wechsel des Dienstangebots entstehen jedoch neue Anforderungen an eine angemessene Systeminfrastruktur. Zur Verdeutlichung soll das heutige Internet mit seinem fast unerschöpflichen Informations- und Dienstangebot dienen, welches einen Eindruck auf von uns betrachtete zukünftige Dienstlandschaften vermittelt. Als konkretes Beispiel sei insbesondere der WWW-Dienst (World Wide Web) im Internet genannt, welcher den Zugriff auf Informationen und Dienste im Netz bei hoher Zugriffstransparenz erlaubt. WWW stellt dabei die generischen Hilfsmittel zur Interaktion zwischen Dienstanbietern und Dienstnutzern zur Verfügung, schreibt aber nicht spezifische Diensttypen — also Spezifika der Dienstschnittstelle und –semantik — vor. Das Auffinden eines gewünschten Diensttyps für eine gegebene Diensttypspezifikation wird dabei garnicht bzw. nur sehr rudimentär — mit anderen Werkzeugen — unterstützt.

Unser Ziel ist es nun, eine Systemplattform zur Unterstützung eines *offenen Dienstemarktes* zu entwickeln, welche die Benutzer (Endbenutzer und Anwendungsprogrammierer) insbesondere beim Auffinden der Dienste, beim Zugang zu den Diensten aber auch beim Anbieten von Diensten adäquat unterstützt. In diesem Beitrag diskutieren wir einige der dabei identifizierten Probleme und zeigen konkrete Lösungsansätze auf.

2 Offene Dienstemärkte: Eigenschaften, Anforderungen, Problemstellungen

2.1 Modell

Die architektonische Basis unserer Betrachtungen offener Dienstemärkte bildet ein allgemeines Objektmodell. Es erlaubt die Strukturierung der Problemwelt auf eine natürliche Art und Weise und definiert präzise Schnittstellen zwischen einzelnen Strukturkomponenten, den Objekten. Die Zustände der Objekte sind nur über wohldefinierte Schnittstellen manipulierbar und daher vor fehlerhaftem Zugriff besser geschützt.

Unser Objektmodell folgt der Definition nach [Boo91]: „Ein Objekt hat einen Zustand, ein Verhalten und eine Identität; die Struktur und das Verhalten ähnlicher Objekte sind in ihrer gemeinsamen Klasse definiert. Die Bezeichnungen *Instanz* und *Objekt* sind wechselseitig verwendbar."

Von einem abstrakten Standpunkt aus gesehen kann eine gemäß dem Objektmodell strukturierte Anwendung — ob sie verteilt ist oder nicht — als gerichteter Graph wahrgenommen werden. Die Knoten des Graphen repräsentieren Objekte und die Kanten symbolisieren Referenzen. Wenn ein Objekt eine Referenz auf ein

anderes Objekt besitzt, kann es Methoden bei diesem Objekt aufrufen (zur Problematik von Objektreferenzen in heterogenen Systemen siehe auch [GHH93]). Der *Objektgraph* spiegelt somit die Beziehungen zwischen den Objekten wider und zeigt auf, welche Objektidentität welchem Objekt bekannt ist. Werden die Methoden, die bei einem Objekt aufrufbar sind, als Dienst angesehen, so gibt die Richtung der Kanten an, wer die Rolle des Dienstnutzers bzw. Diensterbringers einnimmt. Die Struktur des Graphen ist dynamischen Änderungen unterworfen, es kommen neue Objekte und Referenzen hinzu oder bestehende werden entfernt. Demnach zeigt der Objektgraph einen momentanen Schnappschuß einer objektbasierten Berechnung.

2.2 Autonomie und Dynamik

Zu den wesentlichen Eigenschaften offener verteilter Dienstemärkte gehören die weitgehende Autonomie einzelner Rechnerknoten und Software–Komponenten sowie die Dynamik bezüglich der augenblicklich verfügbaren Objekte, d. h. neue Diensterbringer kommen hinzu, während andere ausscheiden oder zeitlich begrenzt unerreichbar sind. Eine zeitliche und räumliche Entkopplung, die die Beziehungen zwischen Diensterbringern und Dienstnutzern unabhängiger und flexibler macht, ist daher wünschenswert. Im Gegensatz dazu muß eine explizite Identität der Objekte gefordert werden, um eine eindeutige Objektreferenzierung über mehrere Objektzugriffe hinaus gewährleisten zu können. Die Einführung einer Referenz zur Aufrechterhaltung eines gemeinsamen Verbindungskontextes zwischen Klient und Dienstanbieter läuft jedoch der zuvor beschriebenen Entkopplung zuwider. Wir sprechen daher von einer Dualität bezüglich der Objektidentität.

2.3 Vermittlung und Verwaltung von Diensten

Der Objektgraph kann als Schnappschuß einer objektbasierten Berechnung interpretiert werden, der über die Zeit hinweg Transformationen unterworfen ist. Dabei interessiert insbesondere die Vergabe von Referenzen zwischen beliebigen Objekten. Üblicherweise kann dies in einer objektbasierten Berechnung nur durch Weitergabe von Referenzen als Parameter von Methodenaufrufen geschehen, bzw. durch implizites Hinzufügen von Referenzen während der Instanziierung eines neuen Objekts. Bei der Interpretation des Objektgraphenmodells im Kontext eines offenen, verteilten Dienstemarktes muß jedoch von einer losen Kopplung des Objektgraphen ausgegangen werden. Dadurch wird der Tatsache Rechnung getragen, daß ein Objekt wegen der räumlichen Entkopplung kein globales Wissen über die Existenz anderer Objekte hat. Ohne weitere Mechanismen einer zugrundeliegenden Infrastruktur ist es demnach nicht möglich, zwischen zwei Objekten verschiedener Partitionen eine Referenz zu vergeben.

Die Lösung des zuvor beschriebenen Problems ist typischerweise Aufgabe eines *Dienstvermittlers* bzw. Brokers oder *Traders* (siehe hierzu Abschnitt 3.3). Eine wichtige Voraussetzung für einen erfolgreichen Suchvorgang von Diensten über einen Dienstvermittler ist ein globaler Konsens über die Typbeschreibung

eines jeden Dienstes. Ein Dienstanbieter registriert seinen Dienst hierbei durch seine Typbeschreibung. Ein Dienstnutzer muß bei einer Anfrage an den Dienstvermittler ebenfalls eine Typbeschreibung als Parameter mitgeben. Nur so kann der Dienstvermittler eine potentielle Typkonformität feststellen. Das formale Konzept für eine derartige Beschreibung von Diensten ist der sog. *Diensttyp* [ISO94b]. Der Diensttyp setzt sich zusammen aus einem *Schnittstellentyp*, welcher die operationale Schnittstelle eines Dienstes anhand von Operationstypen spezifiziert, und den *Dienstattributtypen*, die die Eigenschaften eines Dienstes näher spezifizieren. Eine Typbeschreibung anhand des Diensttyps kommt somit einem Standard gleich, der ein global einheitliches Verständnis über die Art des erbrachten Dienstes abstrakt beschreibt. Auch bei der Typbeschreibung stoßen wir ebenso wie bei der Objektidentität auf eine Dualität der Betrachtungen: Einerseits müssen alle Typbeschreibungen *a priori* festgelegt werden, andererseits widerspricht die Notwendigkeit der Standardisierung der Typbeschreibung der Idee eines offenen Dienstemarktes mit wechselndem Angebot und einer räumlichen und zeitlichen Entkopplung. Idealerweise sollte es Dienstanbietern und –erbringern jedoch möglich sein, auch ohne vorherige Absprachen eine Typbeschreibung zu formulieren, die der Dienstvermittler trotzdem als konform zueinander erkennt.

Aus diesem Grunde führen wir die Begriffe *klassifizierter* und *unklassifizierter Dienst* ein (s. auch [MJM94]): Ein klassifizierter Dienst erfüllt gemäß der Definition des Diensttyps eine spezifische Erwartung bzgl. der angebotenen Signatur der operationalen Schnittstelle und des an dieser Schnittstelle zu erwartenden Verhaltens. Die Dienstsemantik ist hierbei a priori durch den Diensttyp festgelegt und bindend für alle Diensterbringer, welche als Instanz dieses Diensttyps zugreifbar sein wollen. Stellt der Diensterbringer Konformität zum Diensttyp sicher, beschränkt sich die Anfrage eines Dienstnachfragers auf die Benennung des erforderlichen Diensttyps ohne eine direkte Referenz zum Diensterbringer explizit besitzen zu müssen. Alle beteiligten Instanzen — Diensterbringer, –nutzer und –vermittler — können hierbei jedoch nur sinnvoll interagieren, wenn die Konvention des Diensttyps zum Zeitpunkt der Implementierung der Klient– und Server–Anwendung beachtet wird (siehe *Anwendungsdienste* im folgenden Abschnitt). Im Falle der unklassifizierten Dienste hingegen besitzt jeder Diensterbringer zwar einen (impliziten) Diensttyp, jedoch besteht keine Konvention (Standardisierung) bzgl. Schnittstelle und Semantik zwischen Nachfrager, Vermittler und Anbieter. Diese Situation kann eintreten, wenn ein neuartiger Dienst angeboten wird, bevor sein Diensttyp standardisiert ist. Darüberhinaus kann die durch Standardisierung erreichte Substituierbarkeit von Diensterbringern sogar kontraproduktiv sein, wenn Erbringer im kompetitiven Umfeld eines kommerziellen Dienstemarktes auf die Möglichkeit der Individualisierung von Schnittstelle und Semantik angewiesen sind. Derartige unklassifizierte Dienste erfordern spezifische Werkzeuge, welche Endbenutzern einen interaktiven Zugriff erlauben (siehe *Benutzerdienste* im folgenden Abschnitt).

3 Ein Architekturmodell für die Systemunterstützung eines offenen Dienstemarktes

Um den verschiedenartigen Anforderungen nach Autonomie der Diensterbringer, der Dynamik des Dienstemarktes, einer flexiblen Dienstvermittlung und –verwaltung und einem flexiblen Dienstzugriff gerecht zu werden, gehen wir für die weitere Betrachtung von einer groben Unterteilung der Komponenten eines offenen Dienstemarktes in die in Abbildung 1 dargestellten drei Ebenen aus. Diese sind die Ebene der *Benutzerdienste*, die Ebene der *Anwendungsdienste* und die Ebene der *Basisdienste*. Die dritte Dimension bezeichnet die *Infrastruktur*, die sich in sämtliche Ebenen als mögliche Implementierungsgrundlage wiederfindet. Das grundlegende Ebenenkonzept des Architekturmodells ist hierbei das Konzept der Dienstnutzung ähnlich wie in ISO/OSI [EF86] oder im ODP–Referenzmodell [ISO94a]. Die Aufgaben und Funktionen dieser Ebenen bzw. Dimensionen und ihre gegenseitigen Wechselwirkungen werden im folgenden näher erläutert. An dieser Stelle sei darauf hingewiesen, daß das vorgestellte Architekturmodell primär einen generellen Rahmen für die Betrachtung offener Dienstemärkte und deren systemtechnischer Unterstützungsmechanismen bildet. Es eignet sich insbesondere für die Darstellung der Beziehungen der konkreten Forschungs– und Entwicklungsarbeiten der Arbeitsgruppen der Autoren, erhebt hierbei jedoch noch nicht den Anspruch auf eine kohärente Integration der bisher entwickelten Systemkomponenten bzw. Unterstützungstechnologien.

3.1 Benutzerdienste

Benutzerdienste unterstützen den Endanwender transparent oder generisch bei der Nutzung entfernter Anwendungsdienste. Im Falle einer transparenten Nutzung treten *anwendungsspezifische Benutzerdienste* — in der Rolle eines dedizierten Klienten — spezifischen Anwendungsdiensten gegenüber und lassen durch diese Aufträge wie z. B. den Dokumentenausdruck oder eine Sprachübersetzung entfernt ausführen. Die Kopplung zwischen dem Benutzer– und Anwendungsdienst ist hierbei a priori vorgegeben. Im Gegensatz dazu ist ein *generischer Benutzerdienst* unspezifisch gegenüber den genutzten Anwendungsdiensten. Er realisiert den Zugang des Endanwenders zu beliebigen Anwendungsdiensten, wobei zur Laufzeit anhand von Dienstbeschreibungen die Typsicherheit und Protokollkonformität bei der Dienstnutzung gewährleistet wird.

Ein Beispiel für einen generischen Benutzerdienst ist der im COSM/TRADE–Projekt entwickelte *Generische Klient* [ML93, MML94a]. Er dient dazu, auf beliebige, im Dienstemarkt vorhandene Anwendungsdienste interaktiv zuzugreifen. Wie schon in Abschnitt 2.3 beschrieben, ist die grundlegende Annahme hierbei, daß sich in einem offenen Dienstemarkt unter Umständen Anwendungsdienste befinden, deren „Diensttypen“ zum Zeitpunkt der Dienstnutzung noch nicht a priori bekannt sind. Trotzdem sollen diese Anwendungsdienste den Benutzern zugreifbar gemacht werden. Da jedoch keine speziellen Kenntnisse des Benutzers bzgl. der Semantik eines derartigen Anwendungsdienstes vorhanden

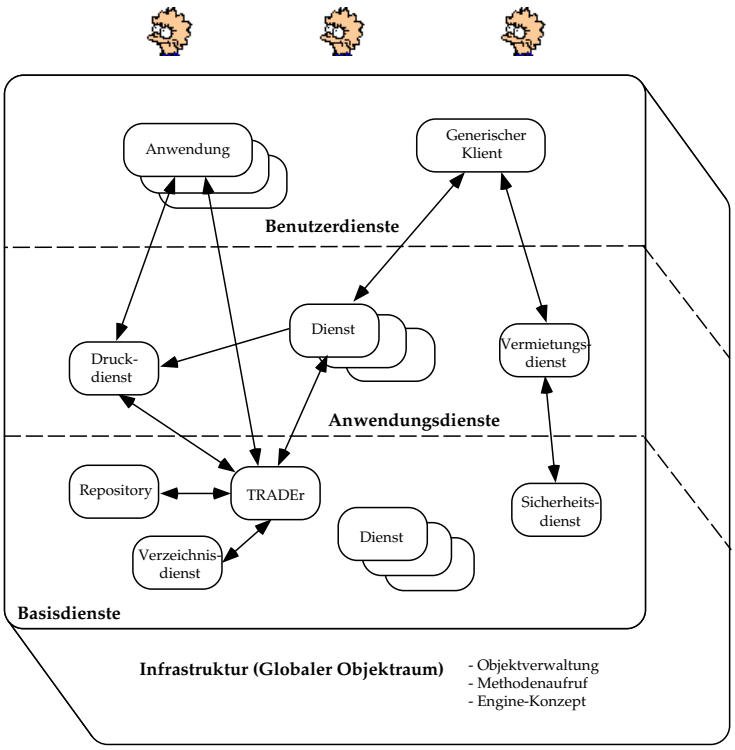

Fig. 1. Architekturmodell für die Systemunterstützung eines offenen Dienstemarktes

sind, ist eine geeignete Unterstützung erforderlich, welche dem Benutzer erlaubt, vor der Nutzung interaktiv Informationen über Schnittstelle und Semantik des entfernten Dienstes zu erlangen. Zu diesem Zweck werden durch den Benutzer geeignete Vermittlungsdienste in den Auswahlprozeß einbezogen, die es ihm erlauben, Dienstbeschreibungen anhand strukturierter Merkmale zu ermitteln und zu inspizieren. Aufgrund dieser Beschreibung erhält der Benutzer schließlich unmittelbaren und interaktiven Zugang zum gewünschten Dienst.

Zentrales Element dieser Dienstbeschreibung ist bei derartigen Anwendungen die sog. *Dienstrepräsentation*, welche neben einer Beschreibung der operationalen Schnittstelle weitere Komponenten umfaßt. Hierzu gehören die Definition der grafischen Benutzerschnittstelle und der dazugehörigen Dialogsteuerung, die Definition eines Schnittstellenprotokolls, welche zulässige Aufruffolgen auf der Basis endlicher Automaten festlegt und eingebettete lokale Zustandsvariablen, welche einen Dialogzustand festhalten können. Die Dienstrepräsentation kann um zusätzliche Komponenten erweitert werden, so daß inkrementell der Spezifikationsumfang vergrößert werden kann. Gegenstand der aktuellen Entwicklung sind hierbei z. B. die Integration von Kosteninformationen für angebotene Ope-

rationen sowie eine Erweiterung der Diensttypbeschreibungen für eine automatisierte Dienstvermittlung, wenn für den betreffenden Dienst ein entsprechender Typ definiert und standardisiert ist.

Eine Dienstrepräsentation wird vom Diensterbringer erzeugt und potentiellen Klienten bereitgestellt. Zum Bindungszeitpunkt wird sie an den Generischen Klienten übertragen, so daß nach Interpretation der Beschreibungsinformation für den Benutzer ein Bildschirmformular generiert werden kann, welches das Modifizieren lokaler Datenwerte und schließlich den Aufruf entfernter Server–Operationen erlaubt. Dienstrepräsentationen können hierbei als Datenobjekte zwischen heterogenen Rechnersystemen versendet und auch lokal zwischengespeichert werden, so daß spezialisierte Server deren Verwaltung und Vermittlung wiederum als gesonderten Dienst erbringen können. Derartige Vermittlungsdienste können hierbei selbst als unklassifizierte Diensterbringer auftreten, so daß sie ihrerseits über den bestehenden Mechanismus der Dienstrepräsentation in generischer Weise zugreifbar sind.

3.2 Anwendungsdienste

Anwendungsdienste stellen den Benutzern bzw. Benutzerdiensten Dienste zur Verfügung, die *anwendungsspezifische* Aufgaben erfüllen. So kann zum Beispiel ein Textverarbeitungsprogramm (aus der Ebene der Benutzerdienste) zum Ausdrucken von Dokumenten sich eines Druckdienstes bedienen, der in der Lage ist, den entsprechenden Dokumententyp auszudrucken. Neben solchen relativ dedizierten, einfachen Diensten lassen sich auf dieser Ebene jedoch auch beliebig komplexe Anwendungsdienste ansiedeln, wie z. B. spezifische Flugbuchungsdienste oder datenbankbasierte Informationsdienste, die selbst wiederum andere Dienste der Anwendungsebene nutzen. Die Nutzung bzw. der Zugriff auf diese Dienste erfolgt für den Benutzer durch die entsprechenden Benutzerdienste der höheren Ebene. Anwendungsdienste nutzen bei der Erfüllung ihrer Aufgabe oftmals mehrere, generische Basisdienste der darunterliegenden Ebenen. Z. B. können die schon oben erwähnten Dienstvermittler (Trader) verwendet werden, die eine generische Dienstvermittlung und –verwaltung beliebiger Anwendungsdienste realisieren, um andere benötigte Anwendungsdienste zu finden.

3.3 Basisdienste

Basisdienste unterscheiden sich gegenüber den Anwendungsdiensten hauptsächlich in der Art der von ihnen angebotenen Dienste. Im Gegensatz zu anwendungsorientierten Diensten stellen sie *generische, anwendungsunabhängige* Dienste bereit, die sich für eine große Anzahl von Anwendungsdiensten nutzen lassen. Beispiele hierfür sind Verzeichnisdienste, Autorisierungs– und Authentisierungsdienste sowie allgemeine Datenbankdienste. Das *Distributed Computing Environment (DCE)* [Fou92] stellt z. B. solche Dienste zur Verfügung, wie sie auch in anderen Plattformen zu finden sind [GH90].

Zusätzlich zu den zur Zeit etablierten Basisdiensten stellen *Vermittlungsdienste* eine wichtige, neuartige Klasse von Basisdiensten in offenen verteilten

Dienstemärkten dar. Aufgabe dieser Dienste ist die Vermittlung und Verwaltung von Anwendungsdiensten. In der Literatur werden derartige Dienste oft als *Trading–* oder *Brokerdienste* bezeichnet [ISO94b]. Im folgenden gehen wir auf zwei Varianten des Tradings und entsprechende Realisierungsansätze näher ein.

Diensttypbasiertes Trading In dieser Variante des Tradings erlaubt der Trader die Vermittlung und Verwaltung von Diensterbringern auf Basis eines formalisierten Diensttypbegriffes. Die zentrale Bedeutung für ein gemeinsames Verständnis von Diensttypen bildet hierbei die Diensttypbeschreibung, wie sie in Abschnitt 2.3 bereits dargestellt wurde. Sie beeinhaltet die grundlegende Beschreibung der Art des Dienstes, die Menge der angebotenen Operationen an seiner Schnittstelle sowie Diensteigenschaften, die den Dienst über syntaktische Aspekte des Schnittstelle hinausgehend anhand von Diensteigenschaften näher charakterisieren. Eine standardisierte Diensttypbeschreibung eines spezifischen Diensttyps bildet die notwendige Voraussetzung für eine strukturierte, automatische Vermittlung und Verwaltung von entsprechenden Dienstangeboten. Mit der Kenntnis einer Diensttypbeschreibung ist es Dienstnehmern und Diensterbringern nun möglich, auf typsichere und semantisch korrekte Art und Weise miteinander zu interagieren.

Abbildung 2 verdeutlicht die Interaktion von Dienstanbietern und Dienstnehmern mit einem Trader am Beispiel des im COSM/TRADE–Projekt an der Universität Hamburg entwickelten und implementierten *TRADErs* (TRADE–Traders) [MJM94, MML94b, MJML95] und stellt seine funktionalen Bausteine vor. Nachdem der definierte Diensttyp „PrintService“ dem TRADEr in Form einer Diensttypbeschreibung bekanntgegeben wurde, ist es speziellen Druckdiensterbringern (Exporters) möglich, sich mit ihren Dienstangeboten unter diesem Diensttyp registrieren zu lassen. Hierfür muß der Diensterbringer neben der Angabe seiner Diensttypbeschreibung — die konform zu der im TRADEr definierten „PrintService“–Diensttypbeschreibung sein muß — einen Namenskontext angeben, in den das Dienstangebot exportiert werden soll. Außerdem müssen die Dienstattribute des angebotenen Dienstes, z. B. „costPerPage“, mit entsprechenden Werten belegt werden, die die Diensteigenschaften charakterisieren.

Potentielle Dienstnehmer (Importers) eines Druckdienstes können nun beim TRADEr unter Angabe einer entsprechenden Diensttypbeschreibung Dienstangebote erfragen und sich — je nach spezifizierter Auswahlstrategie, z. B. „random_choice“ — das am besten geeignete vermitteln lassen. Grundsätzlich bietet der TRADEr verschiedene Strategien zur Auswahl von Diensterbringern an, die sich in der Qualität bzgl. der Wahl des „am besten geeigneten“ Dienstangebotes unterscheiden [WT90]. Die Auswahl erfolgt hierbei unter anderem über die Diensttypkonformität [Car89], die aktuellen Werte der Dienstattribute der Angebote sowie einem Suchkontext, der einen Startpunkt für eine Dienstangebotssuche im globalen Namensraum spezifiziert. Neben statischen Attributen werden bei der Dienstauswahl auch dynamische Dienstattribute, z. B. „queueLength“, berücksichtigt, die aktuelle Zustandsinformationen der Diensterbringer bereitstellen. Sollte ein entsprechendes Dienstangebot durch den TRADEr in seiner

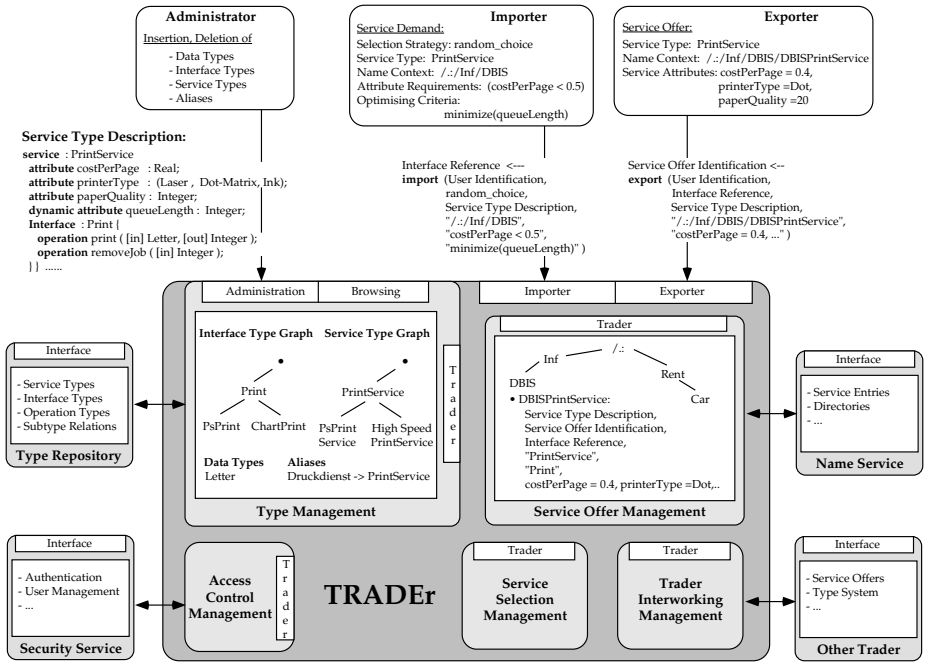

Fig. 2. Architektur des TRADErs

Verwaltungsdomäne nicht gefunden werden, so kann er — je nach vorheriger Spezifikation des Dienstnehmers und der gewählten TRADEr–Konfiguration — auch Trader anderer Verwaltungsdomänen in seine Dienstangebotssuche mit einbeziehen. Diese Form der Interaktion bzw. Interoperabilität von Tradern wird als sog. *Trader Interworking* bezeichnet [ISO94b]. Nach der erfolgten Dienstauswahl durch den TRADEr erhält der Dienstnehmer die notwendigen Bindungsinformationen zum Zugriff auf den entsprechenden Diensterbringer zurück.

Kernbausteine des TRADErs sind das *Typmanagement* (type management), die *Dienstangebotsverwaltung* (service offer management), die oben beschriebene *Dienstauswahl* (service selection management), das *Trader–Interworking* (trader interworking management) sowie die *Zugriffskontrolle* (access control management). Das Typmanagement als für das diensttypbasierte Trading wichtigste Komponente ist für die Verwaltung des gesamten Typsystems des TRADErs verantwortlich und verwaltet insbesondere die Konformitätsbeziehungen der definierten Diensttypen (in der Abbildung als Schnittstellentyp– und Diensttypgraphen dargestellt). Die Einordnung neu definierter Diensttypen erfolgt unter *expliziter* Angabe der entsprechenden Sub– bzw. Supertypbeziehung und beruht primär auf der *Namensäquivalenz* von Dienst– bzw. Schnittstellentypen. Zur Zeit erfolgt der Ausbau des Typsystems hinsichtlich einer automatischen, *impliziten* Diensttypeinordnung und –überprüfung, welche auf der *Strukturäquivalenz*

basiert, wie sie z. B. im Emerald–System [BHJ+87] implementiert ist.

Die konkrete Verwaltung der Dienstangebote durch den TRADEr erfolgt mit Hilfe der Dienstangebotsverwaltung, die eine globale Strukturierung der Dienstangebote erlaubt. Hierbei wird insbesondere die Ablage in hierarchisch strukturierten Namensräumen, im speziellen dem DCE Cell Directory Service und dem DCE Global Directory Service (X.500 Implementation) [Fou92], unterstützt, die einen weltweit zugreifbaren Namensraum realisieren. Zur Zeit werden auch andere, spezialisierte Ablagedienste, wie z. B. Datenbanken oder transaktionsgesicherte Dateisysteme, hinsichtlich ihrer Nutzbarkeit und Effizienz für die Dienstangebotsablage untersucht.

Eine weitere wichtige Komponente des TRADErs ist das Trader–Interworking. Wie schon oben angedeutet, realisiert diese die Zusammenarbeit, das sog. Interworking, zwischen Tradern verschiedener, verteilter Verwaltungsdomänen, wobei die Trader grundsätzlich in heterogenen Ungebung existieren und auch mittels verschiedener, heterogener Implementationstechnologien realisiert sein können. Die Entwicklung dieser Komponente befindet sich in der aktuellen Entwicklung im Rahmen des offenen, weltweiten Kooperationsprojektes *IWT (Interworking Traders)* [Vog94], in dem im speziellen das Interworking von Tradern gleicher, aber auch verschiedener Middleware–Architekturen untersucht und entwickelt wird. Ziel dieses Projektes ist die Demonstration des Interworkings verschiedener Trader–Implementierungen, unter anderem dem TRADEr, dem DCE–Trader vom DSTC [BB94] sowie dem ANSA–Trader [Cam91].

Sämtliche Interaktionen externer Klienten (Administratoren, Dienstanbieter und Dienstanfrager) mit dem TRADEr erfolgen unter Einbindung einer Zugriffskontrolle. Hiermit ist es möglich, speziellen Klienten dedizierte Rechte für spezielle Operationen des TRADErs zuzuordnen. Dieses ist z. B. insbesondere für das Einfügen, Lesen, Modifizieren und Löschen von Diensttypbeschreibungen sowie für sämtliche Funktionen des Exportierens, Importierens und Löschens von Dienstangeboten von Wichtigkeit. Die Realisierung der Zugriffskontrolle im TRADEr erfolgt hierbei unter Nutzung eines externen Autorisierungs– und Authentisierungsdienstes, im speziellen des DCE Security Services [Fou92].

Trading auf Basis von deklarativen Diensttypen Das in diesem Abschnitt vorgestellte Tradingkonzept ist als Ergänzung zu dem zuvor beschriebenen diensttypbasierten Trading zu verstehen. Die Diensttypkonformität wird dabei um *Regeln* erweitert, die sich aus der Hinzunahme von Hornklauseln zu der Typdefinition ergibt. Dadurch wird die bereits erwähnte Dualität der Typbeschreibungen in offenen verteilten Systemen für sogenannte *Typfamilien* gelöst. Eine Typfamilie sei hier informal als eine Menge von Diensttypen charakterisiert, welche sich durch „verwandte Anwendungskontexte“ auszeichnen. Als Beispiel diene die Typfamilie der Container, deren Schnittstelle durch Methoden für das Ablegen und Auslesen von Dateneinheiten definiert ist. Unterschiede zwischen einzelnen Mitgliedern der Typfamilie drücken sich auf einer semantischen Ebene aus (wie beispielsweise LIFO oder FIFO Eigenschaften).

Bei einer rein dienstbasierten Typdefinition *müssen* sich diese Unterschiede

auf einer syntaktischen Ebene widerspiegeln, da hier Typbeschreibungen keine semantischen Informationen beinhalten. Die deklarative Diensttypbeschreibung erweitert die ursprünglich IDL-basierte Typdefinition um eine Semantik, bestehend aus einer Menge von Hornklauseln. Die Art der semantischen Erweiterung unterscheidet sich von den Vor– und Nachbedingungen des Hoare–Kalküls. Die Vorteile der Hornklauseln liegen zum einen in der syntaktischen Trennung der eigentlichen Diensttypbeschreibung, zum anderen in der deklarativen Semantik, wie im folgenden erläutert.

Während die auf dem Hoare-Kalkül aufbauenden Vor– und Nachbedingungen explizit Bezug auf die Signatur einer Methode nehmen, abstrahiert eine separate Hornklauselmenge von der vollständigen Semantik. Die semantische Spezifikation eines Dienstes muß nur noch so aussagekräftig sein, daß sie Unterschiede zu anderen Mitgliedern der *gleichen* Typfamilie „erklären" kann. Durch die den Hornklauseln zugrundeliegende deklarative Semantik wird darüber hinaus eine Abschwächung der zuvor erwähnten *a priori* Definition von Diensttypen erreicht. Eine „Standardisierung" ist nur noch auf der Ebene von Typfamilien notwendig, d.h. für die „generische" Schnittstelle aller Mitglieder einer Typfamilie muß weiterhin ein gemeinsamer Konsens im Sinne einer *a priori* Definition bestehen. Eine detaillierte Beschreibung der deklarativen Diensttypen ist in [Pud94, PMGG95] zu finden.

3.4 Infrastruktur

Der *Global Object Space (GOS)* ist eine Infrastruktur, die die auf dem Objektmodell basierende Kooperation zwischen Dienstanbietern und Dienstnutzern in einem verteilten System unterstützt. Der GOS realisiert konzeptionell gesehen einen globalen Objektraum, in dem Objekte instanziiert werden können (Vergleiche dazu Abbildung 3). Primäres Ziel ist die Schaffung einer Programmierumgebung in Form einer Klassenbibliothek, auf deren Basis sich die gemeinsame Nutzung von Objekten durch andere Objekte (Object Sharing) realisieren läßt. Die Architektur und die Funktionalität des GOS tragen dabei den Erfordernissen Rechnung, die aus dem Objektmodell und dessen Übertragung in eine heterogene verteilte Umgebung resultieren.

Jedes Objekt im GOS verkörpert einen speziellen Dienst. Der Objekttyp (im objektorientierten Sinne) entspricht dabei dem Diensttyp bzw. dem Dienst, weshalb wir fortan diese drei Begriffe synonym verwenden werden. Eine Instanz, die mit dem GOS interagiert, kann eine von drei Rollen einnehmen: Klient(C), Server(S) oder Engine(E). Klienten erwerben Referenzen auf Objekte im GOS und rufen Methoden darauf auf. Sie sind die Dienstnutzer der bereitgestellten Dienste. Server instanziieren (Create) und deinstanziieren (Destroy) Objekte im GOS und legen damit fest, welche Dienste via GOS angeboten werden. Engines hingegen verkörpern Aktivitätsträger, die das Verhalten der Objekte des GOS implementieren und somit als Diensterbringer auftreten. Die getroffene Unterscheidung zwischen Dienstanbieter und Diensterbringer wird im folgenden motiviert.

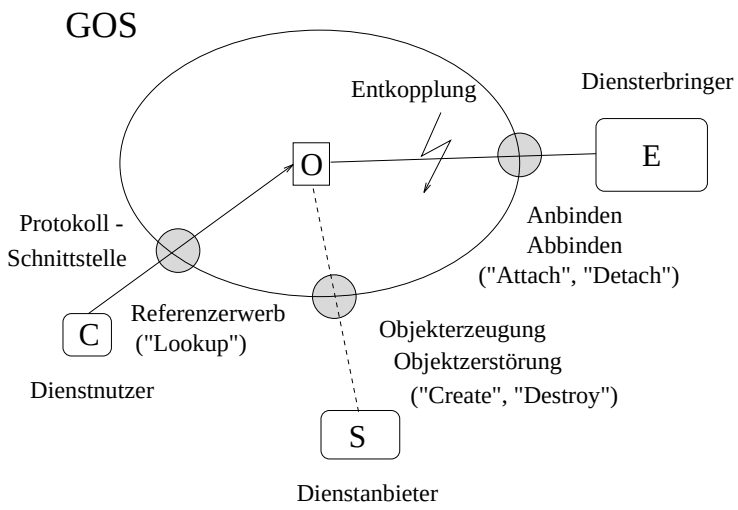

Fig. 3. Architektur des GOS

Wie schon oben erwähnt gehören zu den wesentlichen Eigenschaften offener verteilter Dienstemärkte die weitgehende Autonomie einzelner Rechnerknoten und Software–Komponenten sowie die Dynamik bezüglich der verfügbaren Objekte, wobei eine zeitliche und räumliche Entkopplung, die die Beziehungen zwischen Dienstanbietern und Dienstnutzern unabhängiger und flexibler macht, wünschenswert ist. Unsere grundsätzlichen Vorstellungen von zeitlicher und räumlicher Entkopplung stammen hierbei von Linda [CG86]. Linda realisiert einen globalen, verteilten Speicher für Datentupel, den sogenannten „Tuplespace“. Der Zugriff auf diesen Speicher erfolgt mittels vier Operationen (*in*, *out*, *read*, *eval*), die in eine Programmiersprache eingebettet sind. Die Tupel werden assoziativ über ihre aktuellen oder formalen Datenparameter adressiert. Die Nichtvergabe expliziter, eindeutiger Adressen entspricht der räumlichen Entkopplung (Loslösung von einem konkreten Namensraum). Prozesse, die mittels der Operation *out* Tupel in den Tupelraum schreiben, und Prozesse, die mittels *in* diese Tupel lesen, müssen nicht zeitlich zusammentreffen. Der Konsument wartet, falls sein gewünschtes Tupel noch nicht existiert, bis der Produzent ein Tupel generiert hat. Die Unabhängigkeit der Kooperation vom zeitlichen Zusammentreffen der Operationen *in* und *out* wird als zeitliche Entkopplung bezeichnet.

Die Systemunterstützungsumgebung, für die wir eine Entwicklungsunterstützung bieten wollen, unterscheidet sich in einigen wichtigen Punkten von derjenigen, die Linda zugrundeliegt. Wir betrachten lose gekoppelte, verteilte Rechnersysteme, die durch Heterogenität in Soft- und Hardware und eine weitgehende Autonomie der beteiligten Rechnerknoten gekennzeichnet sind, während Linda für eng gekoppelte, homogene Systeme entworfen wurde. Wir greifen daher das Prinzip der Entkopplung heraus und übertragen es auf die Anforderungen,

die unsere Zielumgebung stellt.

Kombinieren wir das Objektmodell mit der Notwendigkeit der Entkopplung (zu dieser Thematik siehe auch [Pol93]), tritt folgende Problematik auf: Einerseits erfordert das Objektmodell die Vergabe einer eindeutigen Referenz, um ein Objekt über mehrere Zugriffe hinaus identifizieren zu können, da der Dienst eines Objektes gewöhnlich als Sequenz von Methodenaufrufen in Anspruch genommen wird. Andererseits widerspricht gerade dies der Eigenschaft der räumlichen und zeitlichen Entkopplung, da eine feste Referenz auf ein Objekt über ein längeres Zeitintervall besteht. Um dennoch Objektmodell und Entkopplung integrieren zu können, führen wir das *Engine-Konzept* ein. Eine Engine ist eine ausführungsbereite Implementation, die den Dienst eines Objektes erbringt. Sie nimmt Zugriff auf den Zustand eines Objektes und führt die von den Dienstnutzern aufgerufenen Methoden aus. Die Entkopplung wird, wie Abbildung 3 zeigt, durch die An- und Abkopplung von Engines erzielt. Dieses vollzieht sich vollkommen transparent für den Klienten. Aus seiner Sicht referenziert er weiterhin Objekte, die dem dargelegten Objektmodell genügen.

Jede Engine besitzt einen Typ. Voraussetzung für das Anbinden einer Engine an ein Objekt ist die Typkompatibilität zwischen Engine- und Objekttyp. Der Typ der Engine muß ein Subtyp des Objekttyps sein, d. h. die Engine muß mindestens den vom Objekt angebotenen Dienst erbringen können. Insofern wird vom GOS auch auf Diensterbringerseite Polymorphie unterstützt (zu den Begriffen Typ und Polymorphie siehe auch [CW85], [LW93a] und [LW93b]).

Die Typkompatibilität allein reicht jedoch nicht aus. Da die Engine auf dem Objektzustand operiert, muß sie diesen lesen und interpretieren können. Die Auswirkungen, die sich aus der Verknüpfung von Typ und Zustand ergeben, werden derzeit untersucht.

4 Zusammenfassung

Ein offener, verteilter Dienstemarkt ist gekennzeichnet durch eine große Vielfalt der Dienste, die Dynamik des Dienstangebots und die Dynamik der Diensterbringung. Daraus leiten sich neue, in existierenden Plattformen nur unzureichend berücksichtigte Anforderungen an eine adäquate systemtechnische sowie benutzerorientierte Unterstützung ab. In diesem Beitrag werden einige der dadurch neuen Problemstellungen analysiert und es wird eine Betrachtungsgrundlage in Form eines generellen Architekturmodells für offene, verteilte Dienstemärkte vorgestellt. Diese bietet die Grundlage für die konkrete Beschreibung vorhandener bzw. in Entstehung befindlicher, prototypischer Realisierungen von wesentlichen Teilfunktionen einer derartigen Systemplattform, wie z. B. des in diesem Beitrag dargestellten Generischen Klienten, des TRADErs und des Global Object Spaces (GOS).

Die Arbeiten der an diesem Beitrag beteiligten Forschungsgruppen an den Universitäten Hamburg und Frankfurt konzentrieren sich hierbei hauptsächlich auf die technische Dimension von generellen Unterstützungsmechanismen für einen offenen verteilten Dienstemarkt. Hinsichtlich einer darüberhinausgehen-

den globalen Bedeutung eines „Offenen Verteilten Dienstemarktes“ hat dieser Themenkomplex aber auch eine soziologische und politische Dimension mit entsprechendem eigenen Forschungsbedarf.

References

[BB94] A. Beitz und M. Bearman. An ODP Trading Service for DCE. In *Proceedings of the First International Workshop on Services in Distributed and Networked Environments (SDNE)*, S. 42–49, Prague, Czech Republic, Juni 1994. IEEE Computer Society Press.

[BHJ+87] A. Black, N. Hutchinson, E. Jul, H. Levy und L. Carter. Distribution and abstract types in Emerald. *IEEE Transactions on Software Engineering*, Band 13, Heft 1, S. 65–76, 1987.

[Boo91] G. Booch. *Object Oriented Design.* The Benjamin/Cummings Publishing Company, Inc., Redwood City, California, 1991.

[Cam91] APM Ltd. Cambridge. *ANSAware 3.0 Implementation Manual*, 1991. Document RM.097.00.

[Car89] L. Cardelli. Typeful programming. Technical report, DEC SRC Research Report No. 45, 1989.

[CG86] N. Carriero und D. Gelernter. The S/Net's Linda Kernel. *ACM Transactions on Computer Systems*, Band 4, Heft 2, S. 110–129, Mai 1986.

[CW85] L. Cardelli und P. Wegner. On understanding types, data abstraction and polymorphism. *Computing Surveys*, Band 17, Heft 4, Dezember 1985.

[EF86] W. Effelsberg und A. Fleischmann. Das ISO–Referenzmodell für offene Systeme und seine sieben Schichten. *Informatik Spektrum*, Band 9, Heft 5, S. 258–286, Oktober 1986.

[Fou92] Open Software Foundation. *Introduction to OSF DCE.* Prentice–Hall, Englewood Cliffs, New Jersey, 1992.

[GH90] K. Geihs und U. Hollberg. A Retrospective on DACNOS. *Communications of the ACM*, Band 33, Heft 4, 1990.

[GHH93] K. Geihs, R. Heite und U. Hollberg. Protected Object References in Heterogeneous Distributed Systems. *IEEE Transactions on Computers*, Band 42, Heft 7, Juli 1993.

[ISO94a] ISO/IEC JTC 1/SC 21. Information Technology – Open Distributed Proessing – Basic Reference Model of Open Distributed Processing – Part 1: Overview and Guide to Use. Committee Draft 10746–1, September 1994. ISO/IEC JTC 1/SC 21 N8218.

[ISO94b] 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.

[LW93a] B. Liskov und J. Wing. A new definition of the subtype relation. In *ECOOP'93: Object–Oriented Programming.* Springer, 1993.

[LW93b] B. Liskov und J. Wing. Specifications and Their Use in Defining Subtypes. *OOPSLA'93*, Band 28, Heft 10, S. 16–28, Oktober 1993.

[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 im Rahmen des 13th World Computer Congress '94, August 1994. Springer–Verlag.

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

[ML93] M. Merz und W. Lamersdorf. Cooperation support for an open service market. In *Proceedings of the IFIP TC6/WG6.1 International Conference on Open Distributed Processing*, S. 329–340. North–Holland, Elsevier Science Publishers B.V., 1993.

[MML94a] 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.

[MML94b] 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 1994.

[PMGG95] A. Puder, S. Markwitz, G. Gudermann und K. Geihs. AI–based Trading in Open Distributed Processing. Erscheint in: Third International Conference on Open Distributed Processing (ICODP '95), Februar 1995.

[Pol93] A. Polze. The Object Space Approach: Decoupled Communication in C++. In *Proceedings of Technology of Object–Oriented Languages and Systems (TOOLS)*, Santa Barbara, USA, August 1993.

[Pud94] A. Puder. A Declarative Extension of IDL-based Type Definitions within Open Distributed Environments. In *OOIS'94: Object–Oriented Information Systems*. Springer–Verlag, 1994.

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

[WT90] A. Wolisz und V. Tschammer. Service provider selection in an open services environment. In *Proceedings of the 2nd IEEE Workshop on Future Trends on Distributed Computing in the 1990's*, S. 229–235, Cairo, Egypt, September 1990. IEEE.

This article was processed using the LaTeX macro package with LLNCS style