Schnittstellenspezifikation in offenen Systemen

Michael Merz und Winfried Lamersdorf

Universität Hamburg, Fachbereich Informatik, AB DBIS - Datenbanken und Informationssysteme
Vogt-Kölln-Str. 30, D-22527 Hamburg; eMail: [merz|lamersd] @ dbis1.informatik.uni-hamburg.de

KURZFASSUNG

In den immer größeren und immer weiter verbreiteten, offenen verteilten Systemumgebungen steht im Prinzip auch eine immer größere Zahl von Diensten zur Unterstützung der Kooperation bei verteilten Anwendungen zur Verfügung. Deren praktische Verwendungsmöglichkeit ist in derartigen Umgebungen jedoch häufig eingeschränkt, da der Zugang zu diesen Diensten ohne Kenntnis sowohl ihrer Existenz als auch ihrer jeweiligen Schnittstelle undurchführbar wird. Deshalb sollte eine Infrastruktur zur Unterstützung der Entwicklung verteilter Anwendungen in einem solchen 'offenen Markt von Diensten' [WoTs90, MeLa93b] u.a. auch Mechanismen zur Beschreibung von entfernten Anwendungsdiensten und ihren jeweiligen Schnittstellen vorsehen.

Im vorgelegten Diskussionsbeitrag wird dazu zunächst ein eigener Ansatz einer Schnittstellenbeschreibungssprache für anwendernahe Dienste in offenen Umgebungen vorgestellt. Anhand einer erweiterten Notation einer *Service Interface Description Language* (SIDL) soll so für den Benutzer von Diensten in offenen Umgebungen der Zugang und Aufruf dieser entfernten Dienste einheitlich ausgedrückt und strukturiert werden können. Dabei bietet die SIDL zudem auch Möglichkeiten, Dienstbeschreibungen etwa auch für die automatische Generierung von grafischen Benutzerschnittstellen zu verwenden.

In dem geplanten Beitrag soll auch kurz darauf eingegangen werden, wie die Ausdrucksfähigkeit einer derartigen Beschreibungssprache z.B. durch Verwendung von Konzepten aus dem Bereich moderner Programmiersprachen (z.B. erweiterte Typkonzepte) geeignet erhöht werden kann. Schließlich soll gezeigt werden, wie grundlegende Komponenten einer ersten SIDL-Version *prototypisch implementiert* sowie ausgewählte Dienst- und Systemkomponenten einer verteilten, heterogenen Systemumgebung exemplarisch realisiert worden sind.

1. EINLEITUNG

Im Gegensatz zu früheren monolithischen Softwaresystemen, welche alle realisierten Datenverarbeitungsaspekte - wie z.B. Benutzerschnittstellen, Datenspeicherung und Anwendungsprogramme - in einer einheitlichen Umgebung integrierten, ist auf der Basis heutiger Client/Server-Technologie im Prinzip die *Verteilung* dieser funktionalen Komponenten auf ganz unterschiedliche Knoten einer offenen Systemumgebung möglich.

Auf der Basis inzwischen verfügbarer Datenkommunikationsdienste gewinnen dabei nun wiederum Fragen nach einer *Vereinheitlichung* (d.h. Standardisierung) von funktionsübergreifenden Aspekten der Datenkommunikation und der Kooperation in heterogenen Umgebungen zunehmend an Bedeutung [ANSA89]. Derartige Fragen werden u.a. auch im Rahmen der aktuellen Standardisierungsbemühungen zum Thema 'Open Distributed Processing' (ODP) behandelt [Lini91] bzw. im industriellen Umfeld untersucht [OMG91].

Während bisher vor allem funktionsspezifische Schnittstellen - wie z.B. SQL als Datenbanksprache oder 'Remote Database Access' (RDA) [ISO93] als Kommunikationsunterstützung für den Transfer von Datenbankanfragen und -resultaten im Fokus der Betrachtung standen, soll im Rahmen der vorliegenden Arbeit vor allem die *Integration von Schnittstellen verschiedener Funktionsbereiche* in einen einheitlichen, generischen Rahmen näher untersucht werden.

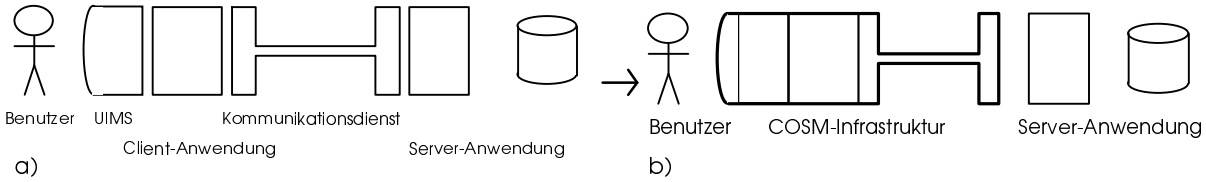


Abb. 1: Die Integration von Schnittstellen verschiedener Funktionsbereiche

Fragestellung

Zusammenfassend lassen sich beim Übergang von zentralisierter Verarbeitung zur offenen Client/Server-Kooperation u.a. folgende zusätzliche Problemkreise näher betrachten:

- Konformitätsaspekte: Sowohl für Clients als auch für Server ist die vormals implizit gewährleistete Konformität bezüglich der Parameter und der Aufrufsequenz beim RPC i.d.R. nicht mehr gegeben, wenn "einander unbekannte" Softwarekomponenten in offenen Umgebungen frei gegenseitig Dienste nutzen können.
- Die *Namenskonventionen* kommerzieller, meist geschlossener RPC-Implementationen erfordern (z.B. durch die willkürliche Vergabe von IDs per Dienst) bereits implizites Wissen über die Semantik der jeweils kooperierenden Anwendungskomponenten.
- Die *Heterogenität* nicht nur der Hardwareplattformen, sondern auch der Betriebssysteme, Kommunikationsprotokolle, Datenspeicherungsmechanismen, Benutzerschnittstellen und schließlich Anwendungssemantiken erfordert eine Vereinheitlichung nicht nur der physischen Repräsentation von Datenwerten, sondern u.U. auch der von Benutzerschnittstellen und der Spezifikation anwendungsspezifischer Protokolle.

Ziele

Der im folgenden präsentierte Ansatz versucht, durch die Integration von Aspekten der Datenkommunikation, Benutzerschnittstelle sowie deren vereinheitlichte Beschreibung,

- Aspekte der in offenen Rechnernetzen i.d.R. immer gegebene Heterogenität für den lokalen Benutzer von entfernten Diensten zu verdecken,

- durch weitgehend lokale Typprüfung bereits vor der Datenübertragung die Anzahl und das Gefahrenpotential fehlerhafter Dienstaufrufe in verteilten Umgebungen zu verringern sowie
- durch einen hohen Grad an Generik den Entwicklungsaufwand für anwendungsspezifische Client-Komponenten zu verringern, und dabei auch die Anfälligkeit des Gesamtsystems bezüglich Programmierfehler zu reduzieren.

Eine Infrastruktur zur Unterstützung offener Client/Server-Kooperation sollte somit in hohem Maße Flexibilität beim Einrichten und Bekanntgeben eines neuen Dienstes bieten. Ebenso sollte für einen Client der organisatorische und programmiertechnische Aufwand für die Erkundung und Interaktion mit neu etablierten Diensten minimiert werden.

2. ANFORDERUNGEN AN EINE SCHNITTSTELLENBESCHREIBUNG IN OFFENEN SYSTEMEN

Im weiteren Verlauf sollen nun Möglichkeiten dargestellt werden, wie durch eine *Erweiterung und Vereinheitlichung von Schnittstellenbeschreibungsmechanismen* einige der aufgeführten Problemkreise bearbeitet werden können.

Wesentliche Bedeutung kommt bei dem hier vorgestellten Ansatz zur einheitlichen Schnittstellenspezifikation in offenen Systemen der Nutzung von *Typinformation* zu, welche bei der Beschreibung einer RPC-Schnittstelle zur typgerechten Konvertierung von Datenwerten aus heterogenen Umgebungen bereits vorgegeben ist. Beim Standard-RPC fließt diese Information *implizit* in die Generierung von typgerechten Code-Segmenten ein und entzieht sich somit einer weiteren Verwendung. Dagegen stellt der Einsatz *interpretierender Stubs* eine Lösung dar, bei der die gegebene, an einen Datenwert gebundene Typinformation *explizit* gemacht und erst zum Aufrufzeitpunkt der entfernten Prozedur aktuell interpretiert wird. Als Vorteil gerade im offenen Systemumfeld erweist sich dabei auch die Möglichkeit zur ad-hoc-Validierung von Datenwerten.

Allerdings setzt die Verwendung generischer Stubs den *Transfer von Typbeschreibungen* zwischen jeweils interagierenden Komponenten voraus: dies kann prinzipiell sowohl durch eine Erweiterung des transferierten Datenwertes (Tagging) als auch durch ein Laden der Typbeschreibung vor dem Bindungszeitpunkt erfolgen. Beide Verfahren erlauben dabei in ihrer Kombination eine zusätzliche *Validierung* übertragener Datenwerte gegen die zuvor erhaltene Typbeschreibung. Ein bestehendes RPC-Protokoll ist für diesen Zweck um zusätzliche Dienste zu erweitern, damit durch diese der Transfer von Typbeschreibungen in standardisierter Weise erfolgen kann.

Die Typinformation, welche beim Parametertransfer dem Zweck der Validierung dient, sollte zudem auch oberhalb des Dienstzugangspunktes zum Transferdienst (dem generischen Stub) nutzbar sein: auch der Client-Anwendung wird damit ebenfalls ein gewisses Potential der Generik bezüglich der Parametertypisierung eingeräumt. Ein in diesem Sinne vollständig ’generischer’ Client wird im folgenden Kapitel vorgestellt.

Neben der Konformität von Parameter- und Resultattypen sollten die Instanzen des spezifischen *Kommunikationsprotokolls* auch (anhand von Information über den Kommunikationszustand der Client/Server-Anwendung) unzulässigen Aufrufsequenzen entfernter Prozeduren vorbeugen. Da die hierfür benötigte Beschreibung von Aufrufsequenzen anwendungsspezifisch ist, stellt die Erweiterung der

Schnittstellenbeschreibung um die Spezifikation zulässiger *Zustandsübergänge* anhand der bereits definierten Server-Prozeduren eine sinnvolle Ergänzung dar.

Ferner sollten im Rahmen der Schnittstellenbeschreibung Informationen zur Auswahl und Identifikation eines Dienstes für dessen potentiellen Benutzer verfügbar sein. Hierbei deuten die bereits skizzierten Probleme auf die Schwierigkeiten hin, die u.a. die *Namensvergabe* in einem globalen, offenen System bereitet. Um hier eine Reduzierung auf attributlistenbasierte Dienstbeschreibungen, welche ihrerseits bezüglich Namenskonvention und Semantik einem langwierigen Standardisierungprozeß unterliegen können, zu vermeiden, sollte im vorgeschlagenen Modell eher der menschliche Benutzer bei der expliziten Auswahl eines Dienstes adäqat unterstützt werden. Dazu dient u.a. die *Erweiterung* der hier vorgeschlagenen *Schnittstellenbeschreibung* um optionale Informationen für den Benutzer, welche ihn bei der Suche nach einem geeigneten Dienst unterstützen.

Weitere Flexibilität bei der Erweiterung von Schnittstellenbeschreibungen bietet die Verwendung polymorpher Typkonzepte aus entsprechenden modernen Programmiersprachen [Matt92]: Wird hierbei eine Schnittstellenbeschreibung als Datenwert aufgefaßt, welcher einem Typ T_1 angehört, und dieser Wert von einer zum Typ T_1 konformen Interpretationsfunktion (z.B. seitens des generischen Clients) akzeptiert, so werden auch alle spezifischeren Datenwerte, deren Typen zu T_1 in einer *Subtyp*-Beziehung stehen, von derselben Interpretationsfunktion akzeptiert. Auf diese Weise können Dienste den Umfang der von ihnen bereitgestellten Schnittstelle individuell und ohne Anpassung der Clients erweitern. Diese Erweiterbarkeit betrifft nicht nur zusätzliche Operationen sondern auch Parametertypen und -anzahl. Zusätzliche Spezifikationselemente, etwa zur Festlegung der Aufrufreihenfolge, stellen eine Subtypisierung der ursprünglichen Schnittstellenbeschreibung dar und sind somit ebenfalls zulässig.

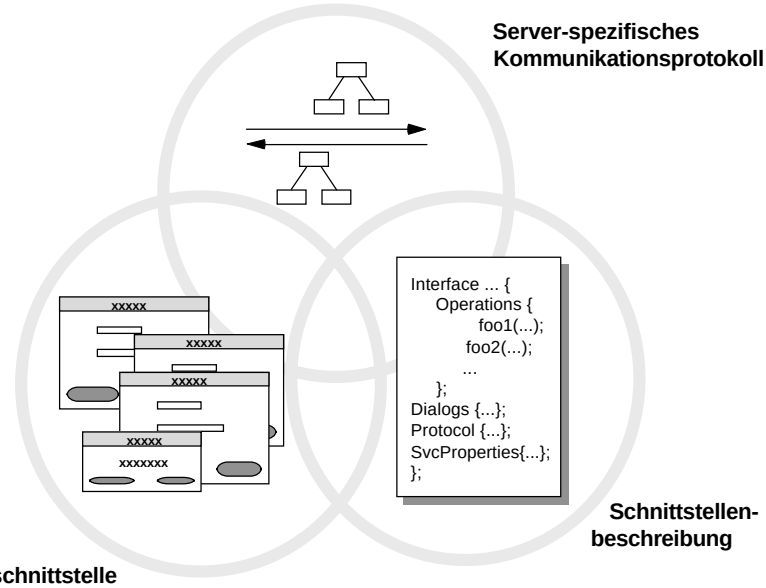


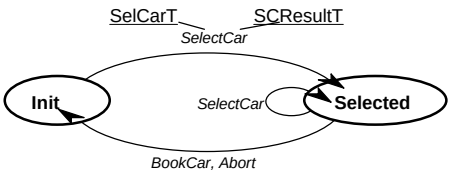
Abb. 2: Aspekte der Schnittstellenbeschreibung und ihrer Notation in SIDL

Der Gewinn der dargestellten Integration von Aspekten der Schnittstellenbeschreibung liegt damit im wesentlichen in der Konzentration der Standardisierungsbemühung auf einen *generischen Dienst*, also auf eine einheitliche Festlegung von Struktur und Semantik

von Protokolldateneinheiten und des Kommunikationsablaufes. Jegliche anwendungsspezifische, individuelle Spezialisierung erfolgt dynamisch durch die Schnittstellenbeschreibung des jeweiligen Dienstes.

SIDL

In folgenden soll nur exemplarisch die Notation der verwendeten Sprache zur Schnittstellenbeschreibung, der *Service Interface Description Language* (SIDL) vorgestellt werden (Näheres siehe auch in [Merz92] und [MeLa93a]):




```
INTERFACE CarRentalService {
    TYPE     SelectCarT RECORD{...};   // see below
    TYPE     SCResultT ...;
    ...
    SERVICE SelectCar {REQUEST SelectCarT; RESULT SCReturnT };
    SERVICE BookCar  { REQUEST BookCarT;   RESULT ResultT };
    SERVICE Abort    { REQUEST BookCarT;   RESULT ResultT };
    STATES {
        INIT:  SelectCar -> S2;
        S2: SelectCar -> S2;
        S2: BookCar -> INIT;
        S2: Abort -> INIT;
        };
    EXPORT {
        Category: "CarRental";
        ChargeMethod: "PerInvocation";
        ChargeAmount: "5";
        ChargeCurrency: "USD";
        ... };
};
```

Abb. 3: Aspekte der Schnittstellenbeschreibung und ihre Notation in SIDL

Die *Typdefinition* in SIDL ist lexikalisch an die abstrakten Syntaxnotation ASN.1 angelehnt und umfaßt i.w. die elementaren Datentypen INTEGER, STRING, FLOAT, BOOLEAN sowie die strukturierten Datentypen RECORD, SEQUENCE, CHOICE, SET. Einen Sonderfall stellt der Typ ANY dar, welcher nicht zu validierende Parameterwerte beschreibt. Dennoch können transferierte Parameterwerte des Typs ANY aufgrund ihres Typpräfixes interpretiert und zwischen verschiedenen physischen Repräsentationen konvertiert werden. Wie auch z.B. beim RDA-Standard für den Zugriff auf entfernte Datenbanksysteme [ISO93] der ASN.1-Typ ANY für dynamische Typen vorgesehen ist, können auch hier derartige, zum Zeitpunkt der Spezifikation noch nicht bekannte Parametertypen zugelassen werden [Rose90; Papp91].

Weiterhin umfaßt die SIDL-Dienstspezifikation eine Beschreibung der Server-seitig ausführbaren *Operationen* mit ihren jeweiligen Parameter- und Resultattypen sowie eine Definition aller erlaubten *Zustandsübergänge* mit Hilfe eines endlichen Automaten. SIDL erlaubt zudem auch die Einstreuung von Textannotationen bei den jeweiligen Komponenten (Typdefinition, Operationen, Zustandsbeschreibung, etc.). Durch geeignete Funktionen des generischen Clients kann der Benutzer sich diese Informationen während der Dienstauswahl oder beim Aufruf einer entfernten Prozedur anzeigen lassen.

Schließlich umfaßt die *Exportbeschreibung* in SIDL solche Informationen, welche zur Klassifikation des Dienstes herangezogen werden können. Im Beispiel aus Abb. 3 z.B.:

```
EXPORT {
     Categorie      : CarRental;
     ChargeMethod   : PerInvocation;
     ChargeAmount   : 5;
     ChargeCurrency: USD;
     };
```

Eine interessante Erweiterung stellt die Hinzunahme des Typs *Service Reference* (SvcRef) als zusätzlichen elementaren Datentyp dar: Parameter und Resultate dieses Typs werden beim generischen Client als Dialogkomponente der Benutzerschnittstelle dediziert dargestellt (Z.B. durch einen "Button"). Dem Benutzer eines generischen Clients bleibt die durch SvcRef beschriebene Bindungsinformation verborgen. Beim Aktivieren einer derartigen Dialogkomponente wird eine Bindung zu dem entsprechenden Server ausgelöst. Auf diese Weise ist der Benutzer in die Lage versetzt, in Abhängigkeit von RPC-Resultaten innerhalb einer ersten Bindung weitere, eingebettete Bindungen einzugehen. Dieses generische Verfahren der Diensteinbettung erlaubt es, flexibel Zugang zu neu im Kommunikationsnetz etablierten Diensten zu erlangen.

3. PROTOTYPISCHE REALISIERUNG AUSGEWÄHLTER SYSTEMKOMPONENTEN

Systemkomponenten

Der Entwurf eines ersten Prototyps zur praktischen Evaluierung der vorgeschlagenen Konzepte umfaßt zunächst eine exemplarische Konfiguration von fünf an der Client/Server-Kommunikation beteiligten Systemkomponenten (Vgl. Abb. 4):

- einem Client, welcher aus einem Laufzeitsystem zur Client/Server-Kommunikation auf der Basis des oben skizzierten generischen Dienstes, einem Basis-Modul zur Service-Auswahl für den Benutzer und dienstspezifisch generierten Komponenten der Benutzerschnittstelle sowie zur Parameterkonvertierung besteht;
- einem *Client Agent* (CAG): ein vom Client entkoppelter Prozeß, welcher Validierungsaufgaben beim Parametertransfer wahrnimmt;
- einem *Server Agent* (SAG): eine dem CAG entsprechende Komponente zur Validierung von Parametern auf der Seite des Servers; sowie
- einem *Server*, welcher den eigentlichen Dienst implementiert.
- Eine ausgezeichnete Komponente stellt schließlich der *'Trader'* dar, welcher zum einen als Server agiert und somit auf der Basis eines generischen Protokolls den Vermittlungsdienst zwischen Client und Server erbringt [BeRa91]. Andererseits stellt er jene Komponente dar, die als Verzeichnisdienst Benutzer bei der Suche ('Browsing') nach entfernten Diensten unterstützt, bzw. an die ein Server während des Startes seine Schnittstellenbeschreibung exportiert.

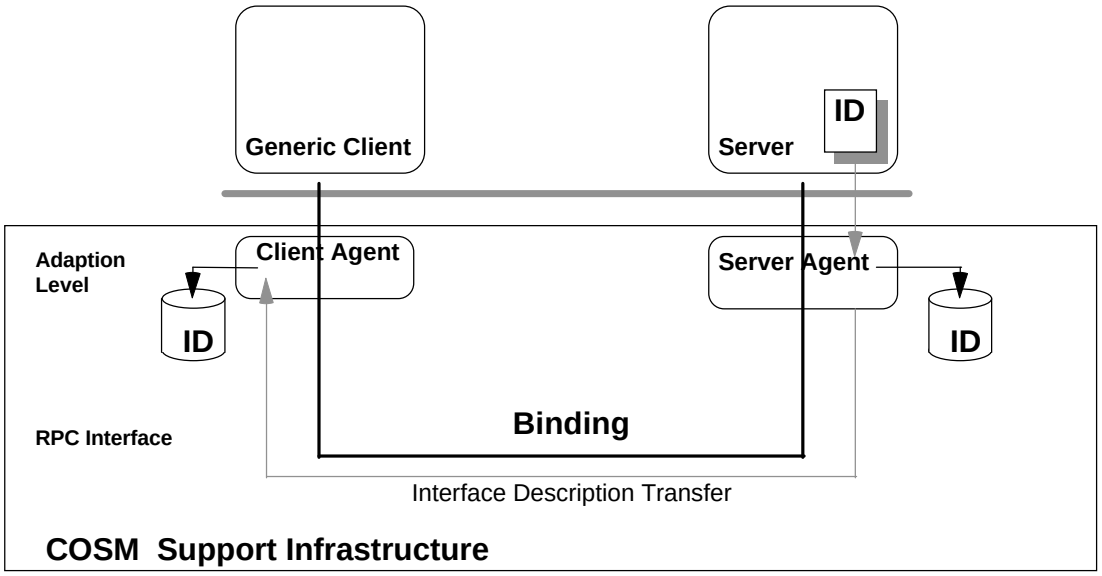


Abb. 4: Systemkomponenten des Prototyps

Diese in Abb. 4 dargestellten Komponenten, sowie aus SIDL-Beschreibungen generierte Benutzerschnittstellen sind in der bisherigen Version der Prototypumgebung bereits implementiert. Bis auf die graphische Benutzerschnittstelle sind die Systemkomponenten frei innerhalb der heterogenen Hardware-Plattformen IBM RS/6000 und Sun SPARC verteilbar. Für den Benutzer besteht Transparenz bezüglich dieser Heterogenität und der Lokation der beteiligten Komponenten.

```
TYPE SelectCarT RECORD {
      INTEGER, LABEL "Mileage", RANGE TINY 50 5000;
      STRING, LABEL "Booking Date";
      INTEGER, LABEL "# Days", RANGE TINY 1 50;
      INTEGER, LABEL "Model",
                  COMM "For a broader range of models
                        consult our service at main branch",
                  RANGE CHOICE 3
                        "BMW 323" "VW Golf"  "Fiat UNO";
      STRING,  LABEL "Customer Name";
      STRING, LABEL "First Name";
      STRING, LABEL "Street";
      STRING, LABEL "City";
      CHOICE {
               INTEGER  LABEL "Visa #";
               INTEGER  LABEL  "MasterCard #";
               INTEGER  LABEL "Amex";
               InvoiceT   LABEL "Invoice";
               }LABEL "Payment";
      } Label "Select Car Form";
```

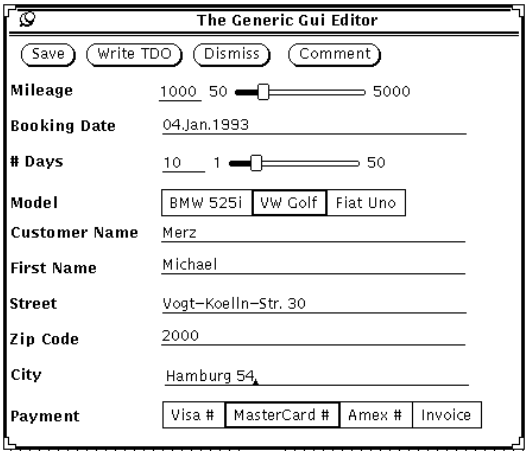


Abb. 5: Generierung eines Formulars zur Parametereingabe aus der SIDL-Spezifikation

4. SCHLUßFOLGERUNG UND AUSBLICK

Als Vorteil der betrachteten Technik generischer Client/Server-Komponenten und einer einheitlichen Spezifikation ihrer Dienstschnittstellen kann vor allem die Konzentration der Entwicklung verteilter Anwendungen auf die Server-Komponente einschließlich ihrer lokal in das System eingebrachten Schnittstellenbeschreibung gewertet werden. Ebenso

erlaubt die Verbindung einheitlicher, transferierbarer Schnittstellenbeschreibungen mit generischen Client-Komponenten eine weitaus flexiblere Einführung neuer, spezialisierter Dienste in den Kontext verteilter Kommunikationsumgebungen. In einer derartigen Infrastruktur, die den Zugang zu einem 'offenen Markt von Diensten' geeignet unterstützt, wird der Entwicklungsaufwand für verteilte Anwendungen wird erheblich reduziert (z.B. durch die Unterstützung von Auswahl, Identifikation und Integration entfernter Dienste durch diese Infrastruktur, oder auch bei der automatischen Generierung von lokalen Benutzerschnittstellen für solche entfernten Dienste).

Die weiteren Arbeiten am Entwurf und an der prototypischen Implementierung einer Systeminfrastruktur für einen 'offenen Markt an Diensten' konzentrieren sich zur Zeit sowohl auf die Weiterentwicklung der SIDL als auch auf die Integration von bestehenden Systemkomponenten für offenen verteilte Umgebungen (z.B. OSF/DCE) in das Prototypsystem.

- Zur Zeit wird im Bereich der prototypischen Testumgebung ein *X.500-Directory--Dienst* um dedizierte Objektklassen erweitert, so daß durch entsprechende Einträge die Identifikation, Speicherung, Verwaltung und netzweite Bereitstellung von Diensten (auf der Basis von SIDL-Spezifikationen) unterstützt wird.
- Vielversprechend erscheint weiterhin die Untersuchung verschiedener, programmiersprachlicher Typsysteme zur Schnittstellenbeschreibung. Hierbei bieten *polymorphe Programmiersprachen* (wie z.B. in [Matt92] vorgestellt), besondere Vorteile, da bei ihnen u.a. aus Signaturen von Dienstschnittstellen und Parameterwerten die Typkompatibilität zwischen beiden dynamisch inferiert werden kann.
- In einer späteren Version der Prototypumgebung wäre es auch denkbar, zusätzliche, sich 'dynamisch' verändernde Informationen über die verwalteten Dienste entsprechend einem Vorschlag von [PoTT91] auf der Basis von Funktionen eines standardisierten, offenen *Systems Management* [ISO90] zu verwalten. Hier sind offensichtlich viele Fragen, insbesondere der Realisierung (z.B. der effizienten Verwaltung sehr großer, verteilter Mengen von heterogenen Daten) erst unzureichend beantwortet [WLS93]. Lösungskonzepte für derartige Probleme sollten auch bereits vorliegenden Ergebnisse berücksichtigen - etwa aus dem Bereich moderner (verteilter) Datenbank-, Objektspeicher- und Informationssysteme oder beispielsweise aus der Standardisierung des 'Remote Database Access' [ISO93].
- Ein weiterer, im Rahmen der bisherigen Betrachtung noch nicht berücksichtigter Bereich liegt in der Verbesserung der *Benutzerschnittstelle*: ausgehend von der aktuellen Implementierung des generischen Clients als lokalen "visuellen RPC-Stub" erscheint eine nähere Untersuchung von Techniken zur Spezifikation der Dialogsteuerung generischer Clients sinnvoll, um die z.B. die benutzerorientierte Dialogverwaltung vom synchronen RPC zu entkoppeln.

LITERATUR

[ANSA89] APM Ltd.: ANSA - An Engineer's Introduction to the Architecture, Cambridge, UK, 1989

[BeRa91] M. Bearman, K. Raymond: Federating Traders: An ODP Adventure, in: Proceedings International IFIP Workshop on ODP, Berlin, 1991

[ISO90] OSI - Systems Management Overview, Information Processing Systems - Open Systems Interconnection, International Standard IS 10040, ISO/IEC JTC1/SC21, 1991

[ISO93] Remote Database Access, Information Processing Systems - Open Systems Interconnection, International Standard 9579, ISO/IEC JTC1/C21 WG3, 1993

[Lini91] P. Linington: Introduction to the ODP Basic Reference Model, in: International IFIP Workshop on ODP, Berlin, 1991

[Matt92] F. Matthes, Generische Datenbankprogrammierung: Sprachliche und Architektonische Grundlagen, Diss., Universität Hamburg, FB Informatik, AB DBIS, 1992

[MeLa93a] M. Merz., W. Lamersdorf: Generic Interfaces to Remote Applications in Open Systems, erscheint in: Procs. Intern. IFIP Workshop on Interfaces in Industrial Systems for Production and Engineering, 1993

[MeLa93b] M. Merz, W. Lamersdorf: Cooperation Support for an Open Service Market, eingereicht für: International Conf. on 'Open Distributed Processing', Berlin, 1993

[Merz92] M. Merz: Generische Unterstützung verteilter Client/ Server-Kooperation in offenen Systemen, Diplomarbeit, Fachbereich Informatik, Universität Hamburg, 1992

[OMG91] The Common Object Request Broker: Architecture and Specification, OMG Document No. 91.12.1, 1991

[Papp91] S. Pappe: Datenbankzugriff in offenen Rechnernetzen, Springer, Berlin Heidelberg, 1991

[PoTT91] R. Popescu-Zeletin, V. Tschammer, M. Tsichholz: 'Y' Distributed Application Platform, IEEE Computer Communication, vol. 14, no. 6, 1991, pp. 366-374

[Rose90] M. Rose, The Open Book, Prentice Hall, 1990

[WLS93] I. Wetzel, W. Lamersdorf, J. Schneider: Systematic Development of Network Management Applications, Procs. Workshop GI FG DB & BS, Heidelberg, März 1993, Datenbank-Rundbrief, GI FG DB, 1993

[WoTs90] A. Wolisz, V. Tschammer: Service Provider Selection in an Open Service Environment, in: 2nd IEEE Workshop on Future Trends of Distributed Computing Systems, Los Alamitos, IEEE Computer Soc. Press, 1990, pp 373-91