Zeitschrift: Technische Mitteilungen / Schweizerische Post-, Telefon- und Telegrafenbetriebe = Bulletin technique / Entreprise des postes, téléphones et télégraphes suisses = Bollettino tecnico / Azienda delle poste, dei telefoni e dei telegrafi svizzeri

Herausgeber: Schweizerische Post-, Telefon- und Telegrafenbetriebe

Band: 73 (1995)

Heft: 3

Artikel: Offene verteilte Systeme. Teil 2, Open distributed processing

Autor: Zweiacker, Marc

DOI: https://doi.org/10.5169/seals-875928

Offene verteilte Systeme
Teil 2: Open Distributed Processing

Marc ZWEIACKER, Bern

Zusammenfassung

Offene verteilte Systeme
Teil 2: Open Distributed Processing

Das Referenzmodell für Open Distributed Processing (RM-ODP) definiert einen Rahmen für die Architektur offener verteilter Systeme. Der Beitrag gibt einen Einblick in einige wesentliche Aspekte des Frameworks. Dieses definiert fünf unterschiedliche Spezifikationsschwerpunkte für verteilte Systeme, *Viewpoints* genannt. Der Computational und der Engineering Viewpoint werden intensiver betrachtet, weil die Definition der darin enthaltenen Konzepte am weitesten fortgeschritten ist. Darunter fällt u. a. die Idee der Objektinteraktionen mittels Interfaces, die Objekt-Bindung, die Interface-Typisierung und das Trading. Den Verteilungs-Transparenzen – eines der populärsten Konzepte für verteilte Systeme – wird in diesem Bericht ebenfalls grosse Aufmerksamkeit geschenkt. Eine Betrachtung zum Thema verteilte Plattform rundet den Bericht ab.

Résumé

Systèmes distribués ouverts
Deuxième partie: traitement distribué ouvert

Le modèle de référence pour le traitement distribué ouvert (RM-ODP) définit un cadre pour l'architecture des systèmes distribués ouverts. L'article donne un aperçu de quelques aspects essentiels de ce cadre. On définit cinq temps forts des spécifications pour systèmes distribués, appelés viewpoints. Un examen plus détaillé du «computational» et de l'«engineering» viewpoint s'imposait, car la définition des concepts qu'ils englobent est la plus développée. En fait notamment partie la notion d'interactions-objets au moyen d'interfaces, la liaison objet, la typisation d'interface et le trading. On attache aussi de l'importance dans cet article aux transparences de distribution, l'un des concepts les plus courants des systèmes distribués. L'auteur décrit en conclusion la notion de plate-forme distribuée.

Riassunto

Sistemi distribuiti aperti
2a parte: Open Distributed Processing

Il modello di riferimento per l'Open Distributed Processing (RM-ODP) definisce l'architettura di sistemi distribuiti aperti. L'autore presenta alcuni aspetti importanti del Frameworks. Quest'ultimo definisce cinque diversi punti chiave di specificazione per sistemi distribuiti, chiamati Viewpoints. Il Computational Viewpoint e l'Engineering Viewpoint vengono illustrati più dettagliatamente poiché la definizione dei principi in essi contenuti è la più aggiornata. Fanno parte di questi principi fra l'altro l'idea delle interazioni degli oggetti mediante interfacce, i legami degli oggetti, la tipizzazione dell'interfaccia e il trading. L'autore dà risalto alla trasparenza della distribuzione – uno dei principi più popolari dei sistemi distribuiti. Egli termina l'articolo con una riflessione sul tema della piattaforma distribuita.

Summary

Open Distributed Systems
Part 2: Open Distributed Processing

The reference model for Open Distributed Processing (RM-ODP) defines a framework for the design of open distributed systems. The article provides an introduction to some of the more important aspects of the framework. This defines five different crucial specifications for distributed systems, known as *viewpoints*. The computational and engineering viewpoints are examined in greater detail, because the definition of concepts contained therein is the most advanced. Amongst other things it includes the idea of object-interactions by means of interfaces, object-linking, interface-type-designation and trading. The report also takes a close look at distribution transparencies – one of the most popular concepts for distributed systems. The article concludes with a study of the distributed platform.

Einleitung

Mit dem Vorhaben, *offene Architekturen für verteilte Systeme* zu standardisieren, ist eines der komplexesten Gebiete der Informatik Gegenstand einer Normung geworden. Die Vielfalt von Aspekten, die für verteilte Systeme charakteristisch sind, dürfte mit ein Grund dafür gewesen sein, dass sich die in der Telekommunikation führenden Normengremien ISO und ITU-T (vormals CCITT) zu einer Kooperation zusammengefunden haben, um dieses ambitiöse Ziel gemeinsam anzustreben. Aus dieser Zusammenarbeit ist das *Reference Model of Open Distributed Processing (RM-ODP)* entstanden, welches als X.900 Standard in die Normung eingehen wird. Zurzeit ist das Referenzmodell als Draft International Standard (DIS) verfügbar und wird allen Erwartungen nach im Frühjahr 1995 zum internationalen Standard (IS) erhoben.

Das Referenzmodell

Dieses Kapitel gibt einen Einblick in den Begriff des *ODP-Standards*. Die Dokumente, welche das RM-ODP (im folgenden «Referenzmodell» genannt) beschreiben, sind mit einer kurzen Inhaltsangabe aufgelistet.

Einführung in das Referenzmodell

Die Elemente eines verteilten Systems (Rechner, Netzwerke, Betriebssysteme usw.) bilden in der Regel ein heterogenes Umfeld, weil sie nicht für die Kooperation gebaut worden sind, die ein verteiltes System auszeichnet. Der Hauptgrund liegt in der Vielfalt der Hersteller für Hard- und Software. Das Referenzmodell überwindet diese Hürde, indem es den Elementen bestimmte Gemeinsamkeiten auferlegt, die aus dem heterogenen System ein nunmehr homogenes, *offenes* machen. Damit künftige technologische Fortschritte mit dem Referenzmodell verträglich bleiben, muss die erworbene Offenheit von maschinennahen Eigenschaften gänzlich unabhängig sein; das Referenzmodell ist daher als Richtlinie für die *Architektur verteilter Systeme* zu verstehen und nicht etwa als Vorschrift über technische Einzelheiten von Hardware-Komponenten. Es liefert einen Formalismus zur Spezifikation von *Server*- und *Client*-Komponenten in einem offenen verteilten System. Daneben schafft das Referenzmodell die wichtigste aller Voraussetzungen für offene Systeme: die Trennung der Systemspezifikation von der Implementation.

ODP-Standards

Das Referenzmodell definiert einen Rahmen für die Architektur (eine Menge von Regeln für die Analyse und Spezifikation) verteilter Systeme, damit diese sogenannt *ODP-konform* sind. Das Hauptaugenmerk liegt auf den Regeln, nach denen verteilte Applikationen entworfen und gebaut werden. *Figur 5* verdeutlicht diesen Zusammenhang: Die Applikationen (A bis D) sind nach bestimmten Regeln aufgebaut, die als Architektur I oder II zusammengefasst werden. Die Freiheit bei der Definition der Architekturen ist durch das Referenzmodell eingeschränkt, d. h. es sind bestimmte Vorschriften einzuhalten, damit die Architekturen als *ODP-Standards* bezeichnet werden können. Da jede Applikation einem ODP-Standard entspricht, ist eine Kooperation zwischen ihnen – trotz teilweise unterschiedlicher Architektur – grundsätzlich gesichert, d. h. obwohl jede der Applikationen unabhängig von den anderen erstellt worden ist, sind alle mit gemeinsamen Mechanismen ausgestattet, die beispielsweise einen Datenaustausch unter ihnen ermöglichen. Das Referenzmodell zwingt jeder ODP-konformen Architektur eine Menge von Eigenschaften auf, damit die daraus abgeleiteten Applikationen zusammengeschaltet werden könnten.

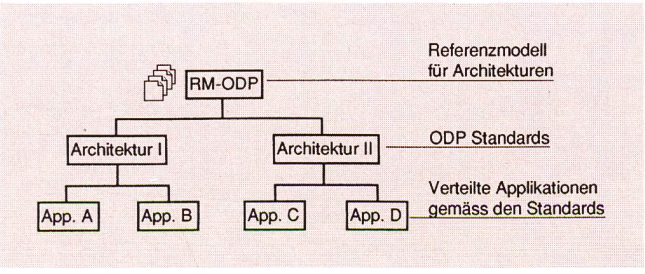

Fig. 5 Referenzmodell, ODP-Standards und Applikationen

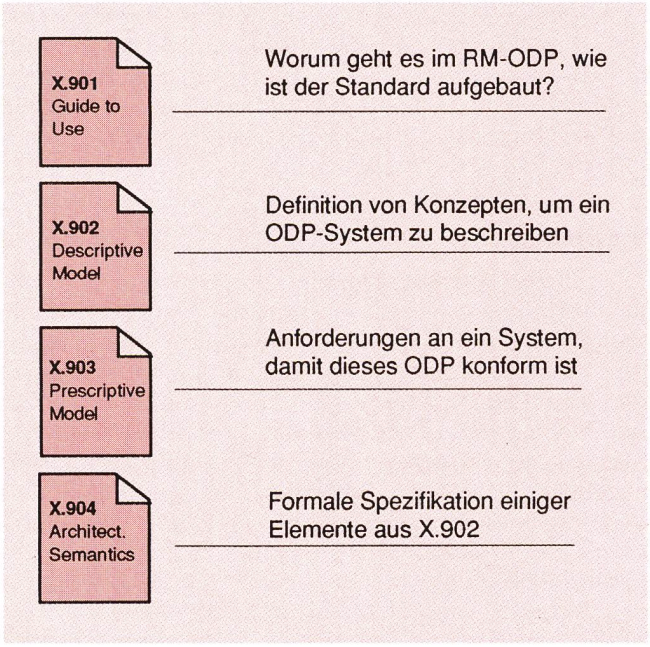

Fig. 6 ODP-Dokumente

ODP-Dokumente

Das Referenzmodell umfasst die folgenden vier Dokumente *(Fig. 6)*:

- *Overview and Guide to Use* [6] führt in das Referenzmodell ein und liefert eine informelle Beschreibung von Konzepten, wie beispielsweise das Objektmodell, die Viewpoints usw. Das Dokument erklärt die Interpretation und die Anwendung des Referenzmodells für die Definition neuer ODP-Standards und Architekturen. Es ist als einziges Dokument nicht verbindlich (not normative).
- *Descriptive Model* [7]: Es definiert die Konzepte und das analytische Gerüst, welches für die Beschreibung von (beliebigen) verteilten Applikationen angewendet wird, sowie den Begriff der *Übereinstimmung* von Architekturen mit dem Referenzmodell (ODP Conformance).
- *Prescriptive Model* [8]: Es enthält eine genaue Beschreibung der erforderlichen Eigenschaften für ein *offenes* verteiltes System. Diese müssen für jedes ODP-System erfüllt sein.
- *Architectural Semantics* [9]: Es enthält eine formale Spezifikation einiger Konzepte aus dem Descriptive Model.

Die Dokumente Descriptive Model und Prescriptive Model haben seit Frühjahr 1994 den Status eines Draft International Standards (DIS). Die beiden anderen Dokumente sind zurzeit in der Phase des Committee Draft (CD). Dieses Nachhinken hängt damit zusammen, dass das Erstellen des User Guide und der Architectural Semantics eine gewisse Stabilität der übrigen Dokumente erfordert.

Konzepte des Referenzmodells

Das Referenzmodell definiert ein *Objektmodell,* das zur Analyse beliebiger verteilter Systeme herangezogen werden kann [7]. Es zu erläutern würde den Rah-

men dieses Beitrags sprengen, da es zu umfangreich ist, als dass man es in Kürze abhandeln könnte. Im folgenden werden einige Konzepte des Referenzmodells vorgestellt, die mit Hilfe des Objektmodells präziser formulierbar wären, deren Kern jedoch auch informal beschrieben werden kann.

Viewpoints

Die vollständige Spezifikation eines beliebigen verteilten Systems beinhaltet eine grosse Menge an Informationen. Es ist in der Regel unmöglich, alle relevanten Aspekte in eine einzige Beschreibung einzubringen. Die meisten Analyse- und Design-Methoden zielen deshalb darauf ab, *mehrere zusammenhängende Modelle* zu verwenden, die jeweils eine bestimmte Facette des Systems abdecken. Das Referenzmodell definiert zu diesem Zweck die *Viewpoints:* Ein Viewpoint beschreibt das System aus einem speziellen Blickwinkel, wobei die Regeln für die Beschreibung des Systems aus diesem bestimmten Viewpoint als *Viewpoint Language* bezeichnet wird. Viewpoints erlauben eine partielle Sicht auf die gesamte Systemspezifikation. Einige Aspekte können mitunter in mehreren Viewpoints auftauchen, weshalb die *Konsistenz* zwischen unterschiedlichen Viewpoint-Beschreibungen gewährleistet werden muss, d. h. die Spezifikation eines Systems in einem bestimmten Viewpoint darf derjenigen in einem anderen Viewpoint nicht widersprechen.

Viewpoints lassen sich mit verschiedenen Photographien eines dreidimensionalen Gegenstandes vergleichen: Jedes Bild enthält Informationen, die auf keinem anderen sichtbar sind, nebst solchen, die auch auf anderen Bildern ausgemacht werden können. Erst die koordinierte Betrachtung aller Bilder lässt den Gegenstand schliesslich vollständig erkennen.

Das Referenzmodell definiert die fünf Viewpoints *Enterprise, Information, Computational, Engineering* und *Technology:*

- Der *Enterprise Viewpoint* beschreibt die Bedingungen, herrührend von der Geschäftspolitik und dem Management, unter welchen das System zu bestehen hat. Die Rolle des Benützers und seine Wechselwirkung mit dem System werden hier festgelegt. Der Begriff Enterprise bedeutet nicht, dass man sich auf ein einziges Unternehmen beschränkt. Die Mittel der Enterprise Language sind geeignet, um Beziehungen zwischen mehreren Organisationen zu modellieren.
- Im *Information Viewpoint* liegt das Schwergewicht auf Informationen und Informationsflüssen. Hier wird die Struktur der Information festgelegt sowie die damit verbundene Semantik, was in einem Modell für den Datenaustausch innerhalb des Systems resultiert (sog. Information Model).
- Der *Computational Viewpoint* beschreibt das System als eine Menge von Objekten (Computational Objects), die dem Client-Server-Prinzip entsprechend interagieren.
- Der *Engineering Viewpoint* macht alle Einzelheiten der Verteilung des Systems sichtbar und definiert eine Menge von Infrastruktur-Objekten, um den Datenverkehr zwischen Servers und Clients zu ermöglichen.
- Der *Technology Viewpoint* beschreibt die physikalischen Komponenten und die verwendeten Technologien für die Implementation (Netzwerke, Betriebssysteme usw.).

Enterprise und Information Viewpoint führen zu einer Systemspezifikation, die von der Verteilung abstrahiert, d. h. die Beschreibungen lassen nicht erkennen, dass sie auf ein *verteiltes* System angewendet werden. Es sind zwei Sichten, welche höhere, managementnahe Ansprüche an ein System ausdrücken können. Die Viewpoints Engineering und Technology spezifizieren die Verteilung von Komponenten und die Mittel, wie die Verteilung realisiert ist. Dazwischen siedelt sich der Computational Viewpoint an, mit dem durch die Trennung in Server- und Client-Objekte ein erster Schritt in Richtung Verteilung unternommen wird.

Computational Objects

Gemäss Teil 1 des Artikels besteht ein verteiltes System aus einer Menge von kooperierenden funktionalen Einheiten. Diese Darstellung beschreibt weitgehend die Sicht des Computational Viewpoint, wo Objekte eine definierte Funktionalität aufweisen. Die Systemspezifikation im Computational Viewpoint konzentriert sich ausschliesslich auf die *Computational Interfaces.* Ein Interface modelliert den Zugriff auf und die Interaktion mit einem Objekt. Ein Objekt kann mehrere Interfaces besitzen, wobei jedes eine bestimmte Funktionalität (einen Dienst) des Objekts freilegt.

Alle relevanten Informationen bezüglich eines Objekts verbergen sich in der Spezifikation des am Interface angebotenen Dienstes. Es können mehrere Objekte den gleichen Service anbieten. Welche Algorithmen, Datenstrukturen, Programmiertechniken und -sprachen dabei für die Implementation eines bestimmten Computational Objects verwendet werden, ist nicht von Belang. Es bleibt dem Programmierer überlassen, wie er die Interface-Spezifikation im Objektinnern umsetzt.

In ODP werden Interfaces mittels der Interface Description Language (IDL) spezifiziert. *Figur 7* zeigt ein Objekt, welches, entsprechend seinen zwei Interfaces, zwei unterschiedliche Dienste anzubieten hat. Beide Interfaces müssen in IDL beschrieben sein. Ein Beschreibungs-Fragment in einer IDL-nahen Form ist für den Echo-Service als Beispiel angeführt. Es lässt er-

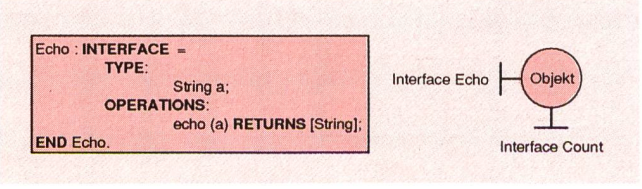

Fig. 7 Interfaces und Interface-Beschreibung in IDL

kennen, dass ein Dienst namens Echo angeboten wird, der einen Datentyp String definiert, sowie eine Operation Echo mit dem String als Eingabeparameter. Aus der IDL-Beschreibung ist jedoch nicht zu erkennen, was die Auswirkungen der Echo-Operation sind, d. h. IDL-Beschreibungen erlauben keinen Rückschluss auf die Semantik von Diensten.

Jedes Computational Object, das den Echo-Dienst anbietet, definiert sein Interface mit derselben IDL-Beschreibung. Man kann auch sagen: Echo entspricht einer bestimmten Sorte von Service.

Typisierung im Computational Viewpoint

In ODP werden die unterschiedlichen Dienstsorten in eine *Typenhierarchie* gegliedert. Jede IDL-Beschreibung – man spricht auch von *Interface-Signatur* – definiert einen bestimmten Servicetyp und somit einen Interfacetyp. Der Nutzen der Typisierung von Diensten liegt in der Möglichkeit, Abhängigkeiten zwischen unterschiedlichen Diensten zu beschreiben, wobei das *Subtyping* die wichtigste darstellt. *Figur 8* zeigt eine Typenhierarchie in Form einer Baumstruktur für die Services in einem ODP-System.

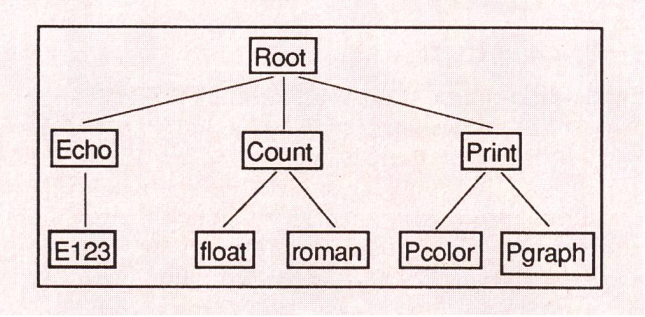

Fig. 8 Typenhierarchie

Die Figur zeigt drei unterschiedliche Typen: Echo, Count und Print. Es sind alles Grundtypen und daher einzig von Root abstammend, d. h. es sind *Subtypen* zu Root. E123 ist ein Subtyp zu Echo, float und roman sind Subtypen zu Count, Pcolor und Pgraph sind Subtypen zu Print. Der Subtyp ist also ein Servicetyp, der im Typenbaum unter einen anderen Servicetyp (den *Supertyp*) zu liegen kommt.

Der Zusammenhang zwischen Supertyp und Subtyp ist folgender: Ein Subtyp bietet den *gleichen* oder einen *erweiterten* Dienst an wie der Supertyp. Print ist ein Dienst für das Ausdrucken von ASCII-Dateien. Die Subtypen Pcolor und Pgraph sind eigenständige Dienste mit entsprechend eigenständigem Servicetyp. Pcolor bietet einen Farbdruck an, Pgraph erlaubt das Ausdrucken von Graphiken. Die beiden Services haben allerdings die Eigenschaft, dass sie auch ASCII-Dateien ausdrucken können, d. h. sie erfüllen alle Eigenschaften des Servicetyps Print.

Für verteilte Systeme bedeutet Subtyping, dass jederzeit ein Subtyp eines Dienstes verwendet werden kann, von dem der eigentliche Servicetyp nicht erhältlich ist. Ein Client könnte somit den Pgraph-Service benutzen, um ASCII-Dateien zu drucken.

Computational Interface Identifier

Die Kenntnis des Servicetyps reicht für einen Client nicht aus, um sich mit dem Interface eines Server-Objekts zu verbinden. Jedes Interface in einer verteilten Applikation ist darum über einen systemweit eindeutigen Namen, den *Interface Identifier*, ansprechbar. Ein Interface Identifier kann zwischen Objekten ausgetauscht werden. Dieses für ODP grundlegende Prinzip ist in *Figur 9* dargestellt. Objekt A benutzt den Print Service, d. h. es kennt den Namen eines Interfaces, das diesen Dienst ermöglicht. Der Interface Identifier 104 wird von A nach B weitergegeben, was letzterem ermöglicht, auf *denselben* Print Service zuzugreifen.

Falls mehrere Print Services im System vorhanden sind, ist es auch denkbar, dass B seinen eigenen Print Service findet (ohne die Hilfe von A), der dann nicht den Identifier 104 haben muss.

Trading

In der Designphase wird u. a. definiert, welche Interfacetypen am verteilten System beteiligt sind. Die Objekt-Instanzen hingegen sind noch nicht bestimmt. Für ein ODP-System heisst das, dass die Interface-Typen und ihre Interaktionen bekannt sind, nicht aber die Interface-Identifier der beteiligten Objekte. Diese müssen zur Laufzeit erst ermittelt werden. Ein Beispiel: In einem verteilten System werde gelegentlich der Print Service angefordert. Die Hardware (Drucker, Plotter usw.), die für solche Aufgaben verwendet werden soll, ist zur Zeit der Programmierung gänzlich unbekannt. Der Servicetyp dagegen ist sehr wohl bekannt. Damit tatsächlich gedruckt werden kann, wird sich das System zur Laufzeit so konfigurieren, dass ein Drucker (Hardware) den Print Service unterstützt und das Interface einen Identifier zugewiesen bekommt, damit Client-Objekte darauf zugreifen können.

Damit die Laufzeitbindung von Interfaces möglich wird, existiert ein spezieller Dienst, der zu jedem Servicetyp einen oder mehrere Interface Identifier kennt, die diesen Service anbieten. Es ist eine Datenbank für

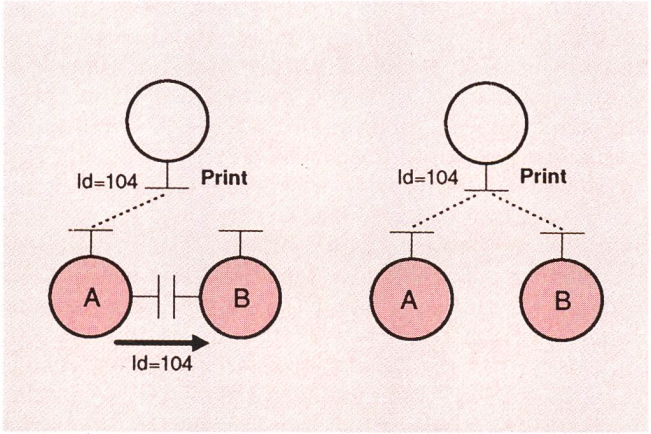

Fig. 9 Interface Identifier bezeichnet einen Service eindeutig
links: A teilt B den Identifier für den Print Service mit
rechts: B kann nun auf den Service zugreifen

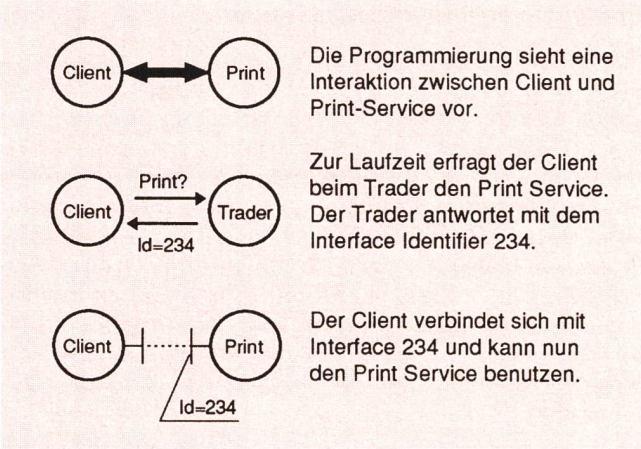

Fig. 10 Servicetypen im Design, Interface-Instanzen zur Laufzeit

verteilte Dienste und nennt sich *Trader.* Der Trader ist das Bindeglied zwischen Client- und Server-Objekten. Server-Objekte lassen sich beim Trader registrieren, Clients fragen ihn nach den registrierten Diensten ab. Der Trader bietet die Dienste also nicht selber an, sondern vermittelt sie, indem er jeweils einen Interface Identifier für den gewünschten Dienst bekanntgibt.

Figur 10 verdeutlicht den Zusammenhang zwischen Design, Trading und Echtzeitbindung: Die Konfiguration von Objekttypen ist das Resultat der Software-Entwicklung. Hier besteht sie aus einem Client-Objekt, das auf einen Service vom Typ Print zugreift. Zur Laufzeit erfragt der Client den Trader nach dem Print Service und erhält als Antwort den Interface Identifier 234. Dieser bezeichnet einen Print Server, der nun kontaktiert werden kann. Die im Applikationsdesign vorgesehene Verbindung zwischen Client und Print Service ist also erst zur Laufzeit zustande gekommen, was mit dem Begriff *Late Binding* treffend bezeichnet wird.

Objekt-Bindung

Ein Client-Objekt, das den Identifier eines Services kennt, muss sich mit dessen Interface verbinden, um auf den Dienst zugreifen zu können. Dafür existiert ein Konzept im Computational Viewpoint, das sich *Binding Object* nennt und von der Netzwerk-Topologie und -Technologie abstrahiert. Es handelt sich um ein spezialisiertes Computational Object, das alleine der Interface-Bindung von Applikations-Objekten dient. *Figur 11* erläutert das Prinzip: Client und Server verbinden ihre Interfaces zum Binding Object. Alle Interaktionen zwischen den beiden Applikations-Objekten erfolgen durch das Binding Object, das als abstrakte Datenstrecke betrachtet wird. Diese kann über das Control Interface in ihren Charakteristiken des Quality of Service *(QoS)* beeinflusst werden. Die Modellierung der Kommunikation als Binding Object löst selbstverständlich keine Probleme des Netzwerks; es verschiebt sie lediglich auf eine andere Ebene, namentlich in den Engineering Viewpoint (s. Begriff «Channel» im nachfolgenden Unterkapitel).

Die Einführung des Binding Objects erlaubt dem Applikations-Designer, die Eigenschaften einer Datenübermittlung beeinflussen zu können. Darin liegt die grosse Stärke dieses Konzepts: Die Anforderungen einer Applikation an die Datenstrecke können bereits in einer hohen Abstraktionsebene – im Computational Viewpoint – spezifiziert werden. Diese direkte Beeinflussung der Interface-Bindung nennt sich *Explicit Binding* und bedeutet, dass ein Applikations-Objekt ein Binding-Objekt anfordert und die Kontrolle der gesamten Kommunikation übernimmt (Auf- und Abbau der Verbindung, Modifikation der QoS-Parameter).

Nicht immer sind die Vorteile dieses Mechanismus für den Programmierer auch wirklich von Nutzen. Gerade im Falle von RPC-(Remote-Procedure-Call-)Interakt ionen sind die Charakteristiken der Kommunikationsstrecke meist von geringer Bedeutung, weshalb die explizite Einführung des Binding Objects überrissen erscheint. Viel einfacher wäre es, wenn man in solchen Situationen gänzlich darauf verzichten könnte. Aus diesem Grund ist das *Implicit Binding* eingeführt worden: Das Binding Object ist auch hier unverzichtbar, allerdings übernimmt die verteilte Plattform nun die Kontrolle über die Bindung.

Engineering Objects

Im Computational Viewpoint kann sich der Programmierer auf die Zusammenhänge unter den Objekten konzentrieren und muss sich nicht um Details der Implementation oder des Datenverkehrs kümmern. Letzteres wird im Engineering Viewpoint untersucht, wo Netzwerk-Protokolle, Algorithmen usw. definiert werden. Die *Engineering Objects* sind ausführbare Code-Segmente und somit von der Hardware abhängig. Etwas vereinfacht kann man sagen, dass sie die lauffähige Repräsentation der Computational Objects sind.

Im Engineering Viewpoint kommt der Technik des Datenaustausches zwischen Objekten grosse Bedeutung zu. Anders als im Computational Viewpoint kann hier nicht mehr von einer abstrakten Interaktion gesprochen werden, sondern die Mechanismen des Datenverkehrs müssen spezifiziert werden. In ODP wurde dafür der *Channel* definiert. Ein Channel ist eine Konfiguration von speziellen Engineering Objects, die

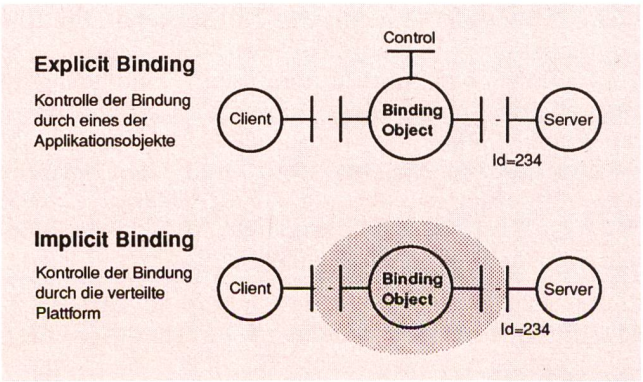

Fig. 11 Explicit und Implicit Binding

zum Zwecke des Datenverkehrs im Referenzmodell definiert worden sind. Die Zusammensetzung eines Channels aus den Grundobjekten *Stub, Binder* und *Protocol* Object ist im Referenzmodell genau vorgeschrieben. Die Spezifikation dieser Objekte definiert die technischen Eigenschaften (Datenprotokolle usw.) einer Kommunikationsstrecke.

Ein Channel modelliert den Kommunikationspfad zwischen zwei Engineering Objects, wenn diese nicht über andere, lokale Mechanismen interagieren, wie beispielsweise Inter-Prozess-Kommunikation. Das im Computational Viewpoint definierte Binding Object erhält im Engineering Viewpoint die Form eines Channels. Dieser besteht aus einer Kette von zweimal drei Engineering Objects *(Fig. 12):*

- Der *Stub* besitzt als einziges Channel-Objekt den direkten Zugang zum Applikationsobjekt, und ermöglicht diesem die Anwendung von stets denselben Mechanismen für den Zugriff auf entfernte Objekte. Das ist darum nicht selbstverständlich, weil diese anderen Objekte unter einem anderen Betriebssystem laufen und eine unterschiedliche Hardware als Basis besitzen können. Eine weitere Hauptaufgabe besteht in der einheitlichen *Datenpräsentation,* die von den Eigenheiten einer bestimmten Hardware-Architektur abschirmt (little endian vs. big endian, unterschiedliche Floating-Point-Formate und Wortbreiten usw.).
- Der *Binder* ist für die Datenintegrität und die Quality of Service über einem Kommunikationspfad zuständig. Es ist dasjenige Objekt, das den Channel kontrolliert (z. B. Verbindungsauf- und -abbau).

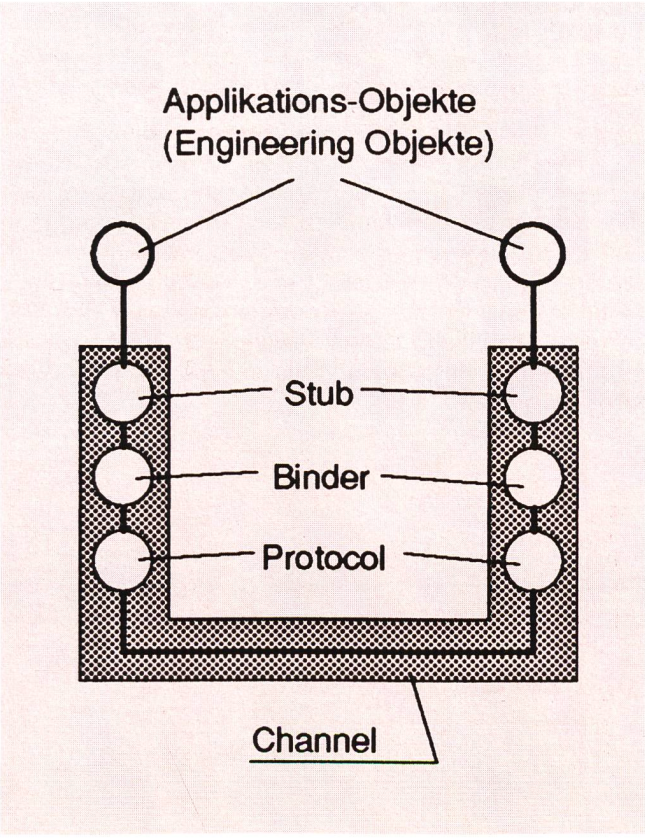

Fig. 12 Engineering Channel

- Das *Protocol* Object bietet die eigentliche Kommunikationsfunktion an. Es greift auf das Netzwerk zu und unterstützt mindestens ein Kommunikationsprotokoll.

Jedes Engineering-Objekt, das einen Datenaustausch über das Netzwerk vornehmen muss, benötigt seine eigene Konfiguration von Stub, Binder und Protocol-Objekten. Dadurch besteht ein Channel immer aus zwei derartigen Ketten, je eine pro Applikationsobjekt. Die beiden Konfigurationen stehen in gegenseitiger Abhängigkeit. So müssen die Protocol Objects auf jeder Seite das gleiche Kommunikationsprotokoll verwenden (z. B. TCP/IP), die Binder müssen sich über den Channel-Aufbau einigen, und die Stubs müssen die gleiche Präsentation der Applikationsdaten vornehmen (z. B. Transfersyntax BER).

Transparencies

Der Programmierer einer verteilten Applikation sollte sich möglichst nicht um Details der Kommunikation und Verteilung von Komponenten im Netzwerk kümmern müssen. Das setzt voraus, dass er eine Programmierumgebung vorfindet, die ihn bereits dahingehend unterstützt, dass die Belange der Datenkommunikation und Verteilung von ihm ferngehalten werden. Eine solche Umgebung nennt man *verteilte Plattform.* ODP definiert eine Zahl von ODP-Funktionen (ODP Functions), die eine verteilte Plattform anbieten kann (aber nicht muss).

ODP Functions

ODP Functions reduzieren die Komplexität einer verteilten Applikation für den Programmierer erheblich. Die folgende unvollständige Liste von ODP Functions soll das Verständnis für dieses wichtige Instrumentarium beim Bau verteilter Systeme fördern:

- *Node Management Function:* kümmert sich u. a. um die Bereitstellung der Mechanismen, damit Channels konfiguriert werden können.
- *Group Function:* behandelt Gruppen von Interfaces so, als würde ein einziges Interface angesprochen. Diese Funktion erleichtert daher den gleichzeitigen Datenaustausch an mehrere Adressaten in einer Applikation (Broadcasting).
- *Replication Function:* Dies ist eine besondere Form der Group Function. Alle Interfaces einer Gruppe bieten den gleichen Service an. Dadurch werden *Redundanzen* in Form von bereitstehenden Ersatz-Diensten geschaffen (sog. *Replicas*). Die Replication Function ist ein Hilfsmittel für den Bau fehlertoleranter Systeme.
- *Migration Function:* Diese Funktion koordiniert den Wechsel eines Objekts von einem Ort an einen anderen (Migration). Das wird erreicht, indem am neuen Ort eine Replica des Services bereitsteht, die zu gegebener Zeit aktiviert wird. Die Migration Function macht somit Gebrauch von der Replication Function.
- *Relocation Function:* Die an einer Migration beteiligten Services erhalten am Zielort neue Interface Identifier. Wenn ein Client, der mit einem migrierten Service in Verbindung steht, vom neuen Identifier Kenntnis erhält, so ermöglicht dies den unterbre-

chungsfreien Ablauf der Server-Client-Interaktion. Damit muss die Applikation im Falle von Migrationen nicht unterbrochen werden. Dieser für ein verteiltes System grundlegende Vorgang wird durch die Relocation Function unterstützt.
- Die *Type Repository Function* baut eine Datenbank auf, in der Service-Typen in einen Hierarchiebaum eingebunden werden.
- *Trading Function:* Sie vermittelt zwischen Service-Angeboten und anfragenden Clients. Ein Server kann seine Dienste in eine Datenbank eintragen lassen. Die Eintragung besteht im wesentlichen aus dem Interface Identifier für diesen Dienst sowie die Eingliederung in die Typenhierarchie (Type Repository Function). Klienten können die Datenbank nach Diensten abfragen und erhalten als Resultat einen Interface Identifier. Der Trader ist eine Verkörperung der Trading Function.

Die obenerwähnten Funktionen decken nur einen Bruchteil der im Referenzmodell definierten ab. Die ODP-Funktionen sind Sonderdienste, die von der verteilten Plattform angeboten werden. Ihre Anwendung erleichtert die Implementation des Systems erheblich und kann die Robustheit und Fehlertoleranz erhöhen.

ODP Distribution Transparencies

Jede ODP-Funktion bedeutet für den Programmierer eine Abschirmung von Problemen der Verteilung. Soll beispielsweise eine Nachricht an viele Objekte zugleich gesandt werden, so kann dazu die Group Function verwendet werden. Die ODP-Funktionen ermöglichen gewisse Vereinfachungen beim Bau eines verteilten Systems, die man Transparenzen oder *Distribution Transparencies* nennt. Transparenzen bauen auf den ODP-Funktionen auf, oder anders gesagt: Die Anwendung von ODP-Funktionen führt zu Transparenzen. Es gibt allerdings auch Formen von Transparenz, die nicht durch ODP-Funktionen realisiert sind. Die folgende Liste nennt einige Transparenzen:

- Die Zugriffs-Transparenz *(Access Transparency)* ist die grundlegendste von allen, weil sie die Hürde der *Heterogenität* überwindet. Sie sagt aus, dass auf alle Objekte eines Systems ein einheitlicher Zugriffsmechanismus anwendbar ist. Die Bedeutung dieser Anforderung wird deutlich, wenn man sich vorstellt, dass Objekte miteinander kommunizieren, die auf *unterschiedlicher Hardware* laufen. Die Zugriffs-Transparenz wird durch das Stub-Objekt im Channel erfüllt.
- Die zuvor erwähnte Replication Function lässt sich so einsetzen, dass ein fehlerhaftes Objekt gegen eine funktionstüchtige Replica ersetzt wird. Wenn dieser Ersatz ohne das Zutun der Applikation selber geschieht, so hat diese nie von seinem fehlerhaften Verhalten erfahren, und man spricht von Fehler-Transparenz *(Failure Transparency).*
- Wenn ein Objekt von einem Knoten auf einen anderen verlegt wird, so kann diese Verschiebung für das Objekt selber verborgen bleiben. Diese Abschirmung nennt sich Verschiebungs-Transparenz *(Migration Transparency)* und macht von der Migration Function Gebrauch.
- Ein verschobenes Objekt bekommt von der verteilten Plattform für seine Interfaces neue Identifiers zugewiesen. Die Relocation Function erlaubt, dass Clients weiterhin mit diesem neu benannten Interface kommunizieren können, als hätte nie eine Verschiebung stattgefunden. Dieses Phänomen nennt sich *Relocation Transparency.*

Die einzige zwingende Transparenz, die jedes ODP-System anbieten muss, ist der einheitliche Zugriff, die Access Transparency. Ein Objekt kann durch seine Interfaces jederzeit auf andere zugreifen und deren Dienste in Anspruch nehmen, ohne seine eigene Darstellung von Applikationsdaten zuerst an unterschiedliche System-Software oder Hardware anpassen zu müssen.

ODP in der Praxis

Obwohl das Referenzmodell noch nicht als internationaler Standard verabschiedet worden ist, gewinnen die darin enthaltenen Ideen für die Computerwelt zunehmend an Bedeutung. Der Einfluss von ODP lässt sich u. a. in der Telekommunikation erkennen.

ODP in der Telekommunikation

In der Telekommunikation kommt der Kontrolle und Überwachung von Netzwerk-Verbindungen grosse Bedeutung zu. Die Mechanismen des Explicit Binding verleihen dem Referenzmodell besonders auf dem Gebiet des Connection Management grosse Stärken. Forschungsprogramme für Intelligente Netze, wie TINA-C und EURESCOM, integrieren ODP von Anfang an in ihre Architekturen. Darauf aufbauend soll das offene Intelligente Netz der Zukunft geplant und realisiert werden.

Plattformen für ODP

Dieser Bericht hat sich an mehreren Stellen auf die verteilte Plattform bezogen. Viele Konzepte aus dem Referenzmodell fliessen in die Architektur und den Bau der verteilten Plattform ein. Ihre Hauptaufgabe besteht einerseits in der Integration heterogener Elemente in ein homogenes offenes System, anderseits bietet sie ein erweitertes Betriebssystem für den Bau verteilter Applikationen an. In *Figur 13* sind einige der Einflüsse des Referenzmodells auf die verteilte Plattform dargestellt:

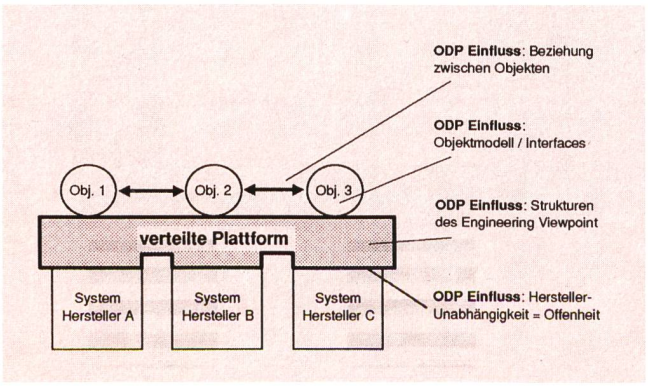

Fig. 13 Verteilte Plattform

- *Beziehung zwischen Objekten:* Der Programmierer ist an das ODP-Interaktions-Schema (Objekte mit Interfaces, Interface-Typisierung usw.) gebunden.
- *Objektmodell und Interfaces:* Die verteilte Plattform bietet die Trading Function an, wodurch die Einführung einer Typenhierarchie erst Sinn macht. Die objektorientierte Programmierung sowie die Ausführung von Objektinstanzen muss ebenfalls durch die Plattform unterstützt werden.
- *Konzepte aus dem Engineering Viewpoint* schränken die Spezifikation der verteilten Plattform auf ODP-Konformität ein.
- Die Adaptierung der verteilten Plattform an die unterschiedlichen Systeme (von unterschiedlichen Herstellern) unterstützt die *Offenheit* der Applikationen.

Dem Autor ist nur ein einziges Produkt bekannt, das als ODP-Plattform bezeichnet werden kann. Es handelt sich um ein Softwarepaket, das durch die Firma *APM Ltd.* in England erstellt worden ist das den Namen *ANSAware* trägt. APM ist die Geburtsstätte mancher Idee, die sich im Referenzmodell niedergeschlagen hat; das ODP Framework hat seinen Ursprung nämlich in der ISA-Architektur (ISA = Integrated Systems Architecture) für verteilte Objekte, die von eben dieser Organisation stammt.

Bibliographie

[1] *Geihs K.* Infrastrukturen für heterogene verteilte Systeme, Informatik-Spektrum (1993) 16, pp. 11–23.

[2] *Bosco P. G.* et al. A Distributed Object-oriented Platform Based on DCE and C++, Proceedings ICODP Berlin, September 1993.

[3] *Elbert B., Martyna B.* Client/Server Computing: Architecture, Applications, and Distributed Systems Management, Artech House 1994, ISBN 0-89006-691-4.

[4] *Pabrai U. O.* UNIX Internetworking, Artech House 1993, ISBN 0-89006-685-X.

[5] APM Ltd. ANSAware Version 4.1 Manual Set, 1993.

[6] X.901, ODP Reference Model Part 1, *Overview and Guide to Use.*

[7] X.902, ODP Reference Model Part 2, *Descriptive Model.*

[8] X.903, ODP Reference Model Part 3, *Prescriptive Model.*

[9] X.904, ODP Reference Model Part 4, *Architectural Semantics, Specification Techniques and Formalisms.*

Marc Zweiacker (Jahrgang 1963) erwarb nach einer Lehre als Elektroniker sein Diplom als Elektroingenieur HTL an der Ingenieurschule Burgdorf. Anschliessend studierte er an der Eidgenössischen Technischen Hochschule ETH in Zürich Informatik. Nach Abschluss seiner Studien als dipl. Informatikingenieur ETH trat er in die Direktion Forschung und Entwicklung, Fachabteilung für Kommunikation und Netzwerke, der Telecom PTT ein. Sein Tätigkeitsfeld umfasst die Untersuchung von verteilten Plattformen, Mitwirkung in europäischen Forschungsprojekten und aktive Beteiligung an der Normung offener verteilter Systeme (Open Distributed Processing, ITU-T X.900 Series Standard).