Erschienen in: Praxis der Informationsverarbeitung und Kommunikation. 11. Jg. 1988, Heft 1, S. 52-56

# Integrierte Informationssysteme: Konventionelle Modelle und Perspektiven objektorientierter Kommunikation

## Illustriert durch Beispiele in C und Smalltalk

Ulrich Frank, GMD

### Gegenstand und Anliegen

Die zunehmende Durchdringung von Organisationen mit Informationstechnologie und die daneben zu verzeichnende Abkehr von rein proprietären Lösungen führt zu einer Reihe von Problemen für das DV-Management. Um nur einige zu nennen: Der Datenaustausch oder allgemeiner die Kommunikation zwischen heterogenen Einheiten ist häufig mit erheblichem Aufwand und Verlust an Information verbunden, die Einbindung und ggfs. Anpassung von Standardsoftware unterliegt zumeist schwerwiegenden Restriktionen, die Vielzahl von Anwendungen unterschiedlicher Architektur erschwert die Pflege, aber auch die Planung von Informationssystemen beträchtlich. Es besteht also ein Bedarf nach Vereinheitlichung der Vielfalt bzw. nach Reduktion von Komplexität durch Systematisierung. Dieser Bedarf spiegelt sich in einem Schlagwort, das in der Liste der Qualitätsmerkmale von Informationssystemen i.d.R. weit oben rangiert: Integration. Im Unterschied zu anderen Schlagworten scheint es sich hier um eine Orientierung zu handeln, die einen substantiellen Beitrag zur Überwindung der aufgezeigten Probleme zu leisten verspricht. Der inflationäre Gebrauch dieses Begriffs - kaum eine neue Software, die nicht in irgendeiner Weise Integration zu unterstützen verspricht, auch die Vielzahl der CI-Konzepte ist bemerkenswert - hat nicht eben zu einem differenzierten Begriffsverständnis beigetragen. Wir werden deshalb zunächst - gleichsam als Bezugsrahmen - Merkmale integrierter Informationssysteme näher betrachten, um darauf aufbauend verschiedene Stufen der Integration und Konzepte zu ihrer Umsetzung zu verdeutlichen. Dabei steht die Interprozeß-Kommunikation im Vordergrund.

Objektorientierte Softwareentwicklung ist seit einiger Zeit ein wichtiges Thema - in der Forschung wie auch zunehmend in der Praxis. Die damit verbundenen Konzepte sind geeignet, wenn nicht euphorisch, so doch optimistisch zu stimmen. Auch wenn dabei i.d.R. die Erhöhung von Entwicklungsproduktivität und Softwarequalität im Vordergrund stehen, ermöglicht objektorientierte Software auch die Integration von Informationssystemen auf einem besonders hohen Niveau. Die Nutzung der Integrationspotentiale objektorientierter Software erfordert jedoch die Überwindung schwerwiegender Probleme, die selten thematisiert werden. Es handelt sich dabei nicht um unüberwindbare Hürden, sie sollten allerdings rechtzeitig als wesentliche Herausforderungen begriffen werden. Dies gilt umsomehr, als mancher Anwender schon heute in seiner strategischen DV-Planung über integrierte Entwicklungsumgebungen (wie z.B. AD-Cycle) oder den zukünftigen Einsatz objektorientierter Datenbanksysteme nachdenken muß.

### Anforderungen an integrierte Informationssysteme

Das Zusammenfügen der Teile eines Informationssystems zu einem einheitlichen Ganzen - eben seine Integration - ist mit einer Reihe von Anforderungen verknüpft. Um einige der wichtigeren zu nennen:

- ❏ Verfügbarkeit

- ❏ Transparenz
- ❏ Flexibilität
- ❏ Integrität

Ein wichtiger Indikator für den Grad der Integration ist die Fähigkeit der einzelnen Komponenten miteinander zu kommunizieren. Integration nimmt hier mit der Annäherung an das Ideal der Transparenz zu. Ein Programm, das physisch auf eine Menge von Daten und Programmen (bei näherer Betrachtung handelt es sich fast immer um Programme) zugreifen kann, sollte dies über eine angemessene Schnittstelle (API) tun können - unabhängig vom physischen Ort der Speicherung bzw. Ausführung. Was ist unter einer angemessenen Schnittstelle zu verstehen? Betrachten wir dazu ein Beispiel. Eine grafische Umsatzstatistik, die mit Hilfe eines Präsentationsprogramms erstellt wurde, soll in ein Dokument übernommen werden, das mit einem Textverarbeitungsprogramm erstellt wurde. Im einfachsten Fall wird der die Grafik darstellende Vektor von Punkten übertragen. So schön eine solche Funktion angesichts ihrer mangelnde Verfügbarkeit in manchen gegenwärtig genutzten Systemen auch sein mag, ihre Nachteile sind kaum zu übersehen. Hier ist einerseits an die fehlende Unabhängigkeit von der jeweils verwendeten Hardware zu denken: ein Pixel-Array, das von einem System mit einer bestimmten Bildschirmauflösung stammt, kann auf einem anderen System mit geringerer Auflösung nur verzerrt wiedergegeben werden. In diesem Zusammenhang ist auch an die Schwierigkeit zu denken, die Grafik weiterzuverarbeiten. Änderungen der Größe der Grafik sind - eben mit gewissen Verzerrungen - durchführbar. Problematischer wird es, wenn auf einzelne Teile der Grafik (z.B. Flächen, Beschriftungen) zugegriffen werden soll: sie sind in dem übertragenen Vektor nicht als solche ausgezeichnet. Eine Abbildung der Grafik mit Hilfe einer geeigneten Beschreibungssprache hilft hier weiter. Anstelle von Punkten dienen dabei aggregierte Konstrukte (Rechtecke, Kreise, Zeichen ...) sowie deren relative Ausrichtung in einem virtuellen Ausschnitt der Darstellung. Darüber hinaus ist es denkbar, die Semantik der Darstellung durch entsprechende Bezeichner in der Nachricht zu kennzeichnen: wenn dem empfangenden Programm bekannt ist, daß es sich hier um die Darstellung einer Umsatzstatistik handelt, wird eine Weiterverarbeitung auf diesem Niveau möglich. Beispiel: “Suche alle Dokumente, in denen Umsatzstatistiken als Kuchendiagramme dargestellt sind”.

Das Beispiel macht deutlich, daß es vorteilhaft sein kann, ein hohes Abstraktionsniveau für die Kommunikation zwischen Programmen zu wählen. Wileden u.a. schlagen in diesem Sinne eine Differenzierung nach “Representation Level Interoperability” und “Specification Level Interoperability” vor [17]. Während auf dem “representation level” lediglich für eine Kompatibiliät elementarer Datentypen oder Strukturen derselben gesorgt wird, vollzieht sich “specification level”-Kommunikation durch Referenzen auf Konstrukte, die den jeweiligen Anwendungsbereichen zuzuordnen sind - eben auf Objekte oder abstrakte Datentypen. Dabei ist allerdings zu berücksichtigen, daß die Unterscheidung zwischen elementaren abstrakten Datentypen durchaus nicht eindeutig ist, sondern allenfalls maschinenabhängig durchgeführt werden kann. Grundsätzlich gilt: Interprozeßkommunikation kann auf semantische Konstrukte referieren, die ihren Ursprung in dem weiten Spektrum zwischen Hardware und Anwendungsdomäne haben. Die oben formulierten Anforderungen an integrierte Informationssyteme legen es nahe, Kommunikation auf einem möglichst hohen Abstraktionniveau zu etablieren. Diese Feststellung ist allerdings zu relativieren. Es reicht hin, sich auf ein Abstraktionsniveau zu beschränken, das soviel Semantik der übertragenen Daten vorsieht, wie der Empfänger für eine mögliche Weiterverarbeitung benötigt. Er sollte also nicht genötigt werden, Teile der Semantik

zu rekonstruieren. Andererseits ist eine Anreicherung mit nicht benötigter Semantik für den Empfänger buchstäblich sinnlos. So ist es für einen Window-Manager unerheblich, welchen Sachverhalt eine auszugebende Grafik repräsentiert.

Um zu verdeutlichen, welche Abstraktionsniveaus unter welchen Voraussetzungen möglich sind, betrachten wir zunächst zwei softwaretechnische Varianten: Remote Procedure Calls und die Versendung von Objekten in Smalltalk.

**Remote Procedure Calls**

Die Konzeption der remote procedure calls ist weitgehend an den Prozeduraufrufen in höheren Programmiersprachen orientiert - mit dem wesentlichen Unterschied, daß die aufgerufene Prozedur in einem anderen Prozeß und ggfs. auf einer anderen Hardware-Plattform läuft: “When a remote procedure is invoked, the calling environment is suspended, the parameters are passed across the network to the environment where the procedure is to execute ..., and the desired procedure is invoked there. When the procedure finishes and produces its results, the results are passed back ...” [2] Es sind im wesentlichen zwei Unterschiede zu einfachen Prozeduraufrufen zu berücksichtigen. So kann die aufgerufene Prozedur nicht mit Hilfe eines üblichen Prozedurnamens identfiziert werden: da sie grundsätzlich in einem entfernten Arbeitsspeicher residieren kann, kann der Linker eine Referenz auf sie nicht befriedigen. Daneben - und das steht für unsere Betrachtung im Vordergrund - ist dafür zu sorgen, daß die übertragenen Parameter an der aufrufenden Stelle die gleiche Semantik haben wie an der aufgerufenen, denn aus dem genannten Grund kann der Compiler keinen Test auf Typkompatibilität durchführen.

Ein weit verbreitetes Protokoll für remote procedure calls ist RPC (Remote Procedure Call) von Sun [15], das wiederum auf TCP/IP (Transaction Control Protocol/Internet Protocol) aufbaut [3]. Es ist an einem Server/Client-Modell mit synchronisierter Kommunkation orientiert. Die Auswahl einer Prozedur auf einer entfernten Maschine erfolgt über eine Prozedurnummer. Um an den Server zu gelangen, der die zugehörige Prozedur verwaltet, wendet sich das Client-Programm zunächst an den sog. Portmapper, bei dem jeder Server die angebotenen Dienste registrieren lassen muß. Dazu verwendet es einerseits die Internet-Adresse der Host-Maschine, andererseits die auf allen Maschinen gleiche Port-Nr. des Portmappers. Falls der gewünschte Dienst verfügbar ist, stellt der Portmapper dem Client die Port-Nr. zur Verfügung:

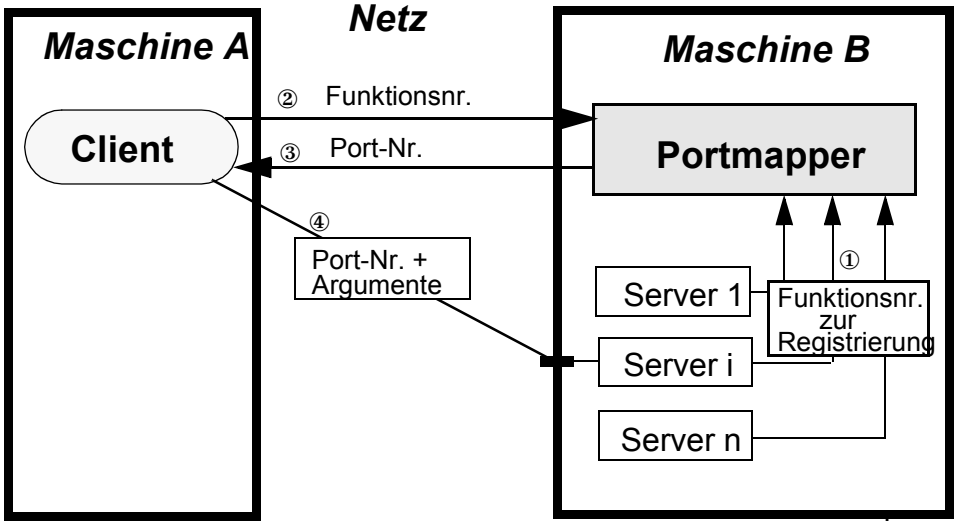


**Abb. 1:** Modell der RPC-Kommunikation nach [15]

Um das Abstraktionsniveau von Remote Procedure Calls zu veranschaulichen, greifen wir im folgenden auf Code-Fragmente eines Systems zurück, das vor einiger Zeit in der GMD entwickelt wurde [6]. Es handelt sich dabei um ein Volltext-Retrieval-System zur Literatur und Dokument-Recherche. Der Retrieval-Server läuft auf einer UNIX-Maschine, die Clients entweder auf UNIX- oder MS/DOS-Maschinen. Wie bereits erwähnt ist ein remote procedure call vergleichbar mit einem Prozeduraufruf in einer höheren Programmiersprache. Auch wenn grundsätzlich zusätzliche Angaben wie Host-Name, Funktionsnr. (statt Funktionsname) spezifiziert werden müssen, können diese Details am Ort des Funktionsaufrufs weitgehend verborgen bleiben. Der folgende Funktionsaufruf veranlaßt einen remote procedure call an den Server. Als Ergbnis wird die Gesamtzahl der abgelegten Titel, die Zahl der Treffer und ggfs. der erste gefundene Titel zurückgeliefert.

```
send_query (query, &satz, &Error, &gesamtzahl, &titelzahl)
```

Der Client erhält jeweils nur einen Titel, den er beim Server über eine relative Titelnummer anfordern kann. Wie wird nun dafür gesorgt, daß Client und Server die als Argumente verwendeten Konstrukte sowie die jeweils verwendete Funktionsnummer in einheitlicher Weise interpretieren? Die Struktur der Schnittstelle wie auch die Zuordnung von Funktionen zu Funktionsnummern wird in Header-Files definiert, die in identischer Form im Server wie in den Clients verwendet werden sollten:

```
#define  ANSWER_QUERY   ((u_long)151000)     /*  receive and dispatch query       */
#define  GET_RECORD   ((u_long)151002)     /*  provide one record of database       */

typedef struct
 {
        char  machine_passw [PASSLENGTH];
        unsigned short  userID;
        unsigned short error;
        char  query [MAX_QUERY_LEN];
```

```
unsigned int Trefferzahl;
unsigned int Gesamtzahl;
char autor[120];
char titel[255];
char zeitschrift[30];
char editor[120];
```

Da RPC für den Einsatz auch in heterogenen Netzen konzipiert ist, muß sichergestellt sein, daß die in der Definition der Schnittstelle spezifizierten Datentypen in konsistenter Weise verwendet werden. Dazu dient die sog. *External Data Representation* (XDR). Sie bietet eine Menge von Datentypen sowie einfacher Datenstrukturen (Arrays). Zum Funktionsumfang von RPC gehören Konvertierungsroutinen, die die Datenrepräsentation der jeweiligen Maschine in die Repräsentation der Protokollebene transformieren - et vice versa. Die XDR-Routinen bieten über die Transformation hinaus einen gewissen Schutz gegen Inkonsistenzen: sie liefern einen Fehlercode zurück,wenn - z.B. nach einer einseitigen Änderung der Schnittstelle - Datentypen von Server und Client nicht mehr einander entsprechen.

```
bool_t queryconv (xdrsp, sf)
XDR *xdrsp;
QUERYSTRUCT *sf;
{
    char *sp;
    sp = sf->machine_passw;
    if (!xdr_string (xdrsp, &sp, PASSLENGTH)) return 0;
    if (!xdr_u_short (xdrsp, &sf->userID)) return 0;
    if (!xdr_u_short (xdrsp, &sf->error)) return 0;
    sp = sf->query;
    if (!xdr_string (xdrsp, &sp, MAX_QUERY_LEN)) return 0;

    .
    .return XDR_ERROR;
```

Zusammenfassend läßt sich feststellen, daß remote procedure calls Kommunikation auf einem Abstraktionsniveau bieten, das von Funktions- bzw. Prozeduraufrufen in konventionellen Programmiersprachen bekannt ist: die Semantik der Schnittstellen ergibt sich aus elementaren Datentypen sowie daraus gebildeter einfacher Strukturen (Records, Arrays). Im Hinblick auf Integrität sind allerdings Abstriche zu machen. So kann die Konsistenz der Schnittstellen nicht ohne weiteres garantiert werden: Modifikationen sind an mehreren Stellen in übereinstimmender Weise durchzuführen. Im Falle der Festlegung von Funktionsnummern geht es dabei nicht allein um Abmachungen zwischen Server- und Clientseite, vielmehr müssen auch die Server auf einer Maschine disjunkte Mengen von Funktionsnummern verwenden, da sonst der Eintrag eines Servers im Portmapper von einem anderen überschrieben werden kann.

### Interprozeßkommunikation in Smalltalk

Angesichts der Forderung, Kommunikation zwischen Programmen auf einem möglichst hohen Abstraktionsniveau durchzuführen, erscheint ein objektorientierter Ansatz sehr attraktiv. Anders als beim konventionellen Datenaustausch können mit Objekten Konstrukte transferiert werden, die sehr viel mehr Semantik beinhalten.[1] Sie enthalten eben nicht nur Daten, sondern auch Methoden. Da in konsequent objektorientierten Systemen wie Smalltalk keine Unter-

scheidung zwischen Daten und Objekten gibt, sollte man wohl besser sagen: ein Objekt enthält ein oder mehrere andere Objekte, die in ihrer Gesamtheit den Zustand des Objekts angeben sowie Methoden, die festlegen, nach welchen Regeln Zustandsveränderungen durchgeführt werden können. Ein für unsere Betrachtung wesentlicher Unterschied zu konventionellen Prozeduren ist dabei, daß Objekte Instanzen von Klassen sind: ähnlich wie bei konventionellen Daten ist ihre Semantik auf einer Metaebene festgelegt.

Die Versendung von Objekten anstelle von Daten öffnet die Perspektive auf eine erheblich komfortablere und leistungsfähigere Kommunikation. Der Empfänger, der ein Objekt erhält, muß nicht mehr seine eigenen - u.U. unangemessenen - Interpretationsverfahren anwenden, um die Nachricht in den jeweiligen Verarbeitungskontext zu integrieren. Vielmehr kann er dem empfangenen Objekt eine Nachricht senden, um den gewünschten Zustand zu erhalten. Dazu muß er nur das für ihn bedeutsame Protokoll des Objekts, also die Menge der Nachrichten, auf die das Objekt antwortet, kennen. Eine rudimentäre Variante dieses Vorgangs ist von manchen Betriebssystemen (z.B. Mac-OS) bekannt: ein Objekt, z.B. ein Dokument, reagiert auf Anklicken mit der Maus, indem es sich - durch Aufruf eines geeigneten Programms - sichtbar macht. Dieses Verfahren ist für den Benutzer sehr komfortabel, aber u.U. auch lästig: weil durch das Anklicken nur eine einzige Nachricht gesendet werden kann, ist das Objekt schlimmstenfalls überhaupt nicht zu verwenden, wenn die zugehörige Methode (eben das Programm) nicht verfügbar ist.

Die Funktionsweise objektorientierter Kommunikation soll im folgenden mit einem Beispiel illustriert werden, das in Smalltalk erstellt wurde. Smalltalk bietet sich dazu m.E. nachhaltig an, da es die Vorzüge, aber auch die Schwierigkeiten einer Kommunikation auf Objektebene besonders deutlich macht. Um einen Vergleich zum RPC-Beispiel zu erhalten, haben wir wiederum ein Server/Client-Modell realisiert. Der Server führt auf Anfragen Recherchen in Literaturbeständen durch. Der Client stellt die erhaltenen Ergebnisse für den Benutzer dar. Soweit entspricht die Funktionsweise dem RPC-Beispiel. Die Kommunikation erfolgt allerdings auf einem sehr viel höheren Abstraktionslevel. Nachdem der Server ein Retrieval durchgeführt hat, aktualisiert er ein Objekt, hier derVerwalter genannt, indem er ihm die Liste der Treffer zuschickt, den Query-Ausdruck, die Gesamtzahl der Treffer, die Zahl der gefundenen Aufsätze, Monografien usw. Zudem verwaltet derVerwalter die Historie der Queries. Mit Hilfe geeigneter Methoden kann er dann Auskunft über bestimmte Häufigkeiten geben. Darüber hinaus verfügt der Verwalter über Methoden zur grafischen Darstellung der von ihm geführten Statistiken. Auf diese Weise ergibt sich folgendes Kommunikatonsmodell:

1. Diese Feststellung soll nicht darüber hinwegtäuschen, daß es gewiß nicht trivial ist, ein komparatives Maß für Semantik zu entwickeln. Für unseren auf formale Systeme eingeschränkten Kontext ist es an dieser Stelle m.E. hinreichend, eine Vorstellung rationalistischer Wissenschaftstheorie zu übernehmen: die Semantik eines formalen Systems ist umso größer, je mehr denkmögliche Interpretationen ausgeschlossen werden.

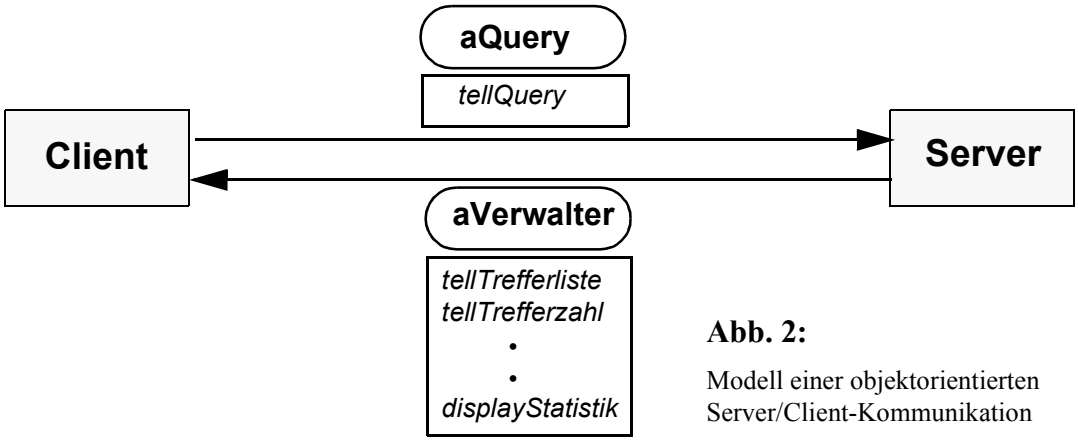


**Abb. 2:**

Modell einer objektorientierten Server/Client-Kommunikation

Zur Realisation der Kommunikation zwischen Server und Client wurde auf die Smalltalk Klasse *SharedQueue* zurückgegriffen. Die Synchronisation erfolgt über ein Objekt der Klasse *Semaphore*. Die Namen der versendeten Objekte sind nur auf der Seite ihres Entstehens bekannt (*theQuery* auf der Client-Seite, *theVerwalter* auf der Server-Seite). Der jeweilige Empfänger erhält also jeweils eine Instanz, auf die er nicht namentlich referieren kann. Vielmehr wendet er sich an die entsprechende *SharedQueue* und fordert das nächste Objekt an. Er kann dann allerdings die Klasse des Objekts feststellen, da jedes Smalltalk-Objekt eine Methode enthält, die über seine Klasse Auskunft gibt. Nachdem der Client auf Anfrage eine Instanz der Klasse *Verwalter* erhalten hat, kann er sie über das ihm bekannte Protokoll veranlassen, bestimmte Methoden auszuführen. Für die Darstellung der Titel (s. Abb. 3) haben wir einen Ansatz gewählt, der immer noch eine gewisse Ähnlichkeit zum RPC-Beispiel aufweist: das für die Bildschirmausgabe zuständige Objekt des Clients fordert vom Verwalter-Objekt die Treffer (als Liste von Records, bzw. in Smalltalk-Terminologie Arrays) um sie dann in bestimmter Weise in dem von ihm verwalteten Bildschirmkomponenten auszugeben. Ein Unterschied, auf den wir noch eingehen werden, ist allerdings unverkennbar: während im RPC-Beispiel jeweils nur ein Element der Liste transferiert wurde, wird hier dem Client die gesamte Liste zur Verfügung gestellt. Bei der grafischen Darstellung statistischer Werte wird in unserem Beispiel der objektorientierte Ansatz besonders deutlich: die zuständige Client-Methode schickt an das empfangene Verwalter-Objekt allein die Nachricht *displayStatistik*. Daraufhin veranlaßt dieses Objekt die Öffnung eines Fensters, in dem die gewünschte Grafik erscheint:

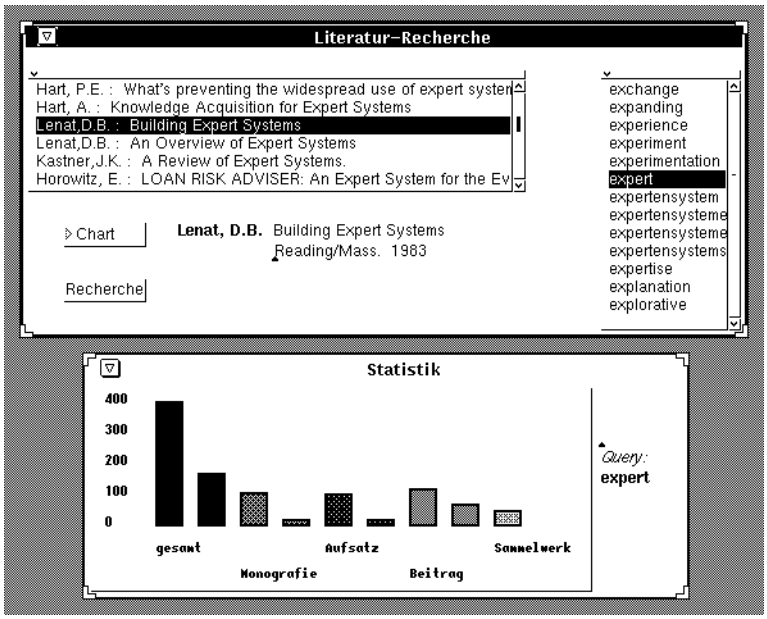


**Abb. 3:**

das obere Fenster wird von dem Client-Prozeß erzeugt. Er präsentiert dort die Daten, die er vom Verwalter-Objekt erhält. Das untere Fenster wird vollständig von dem empfangenen Objekt erzeugt.

Der Zugriff auf das übermittelte Verwalter-Objekt erfolgt auf Client-Seite nach Freigabe durch den Semaphore in der Methode *holVerwalter*. Die Zuweisung der im Pool-Dictionary (ein Verzeichnis, das von mehreren Objekten zur Verwaltung gemeinsamer Objekte verwendet werden kann) eingetragenen *SharedQueue* an eine lokale Variable dient allein der Anschaulichkeit.

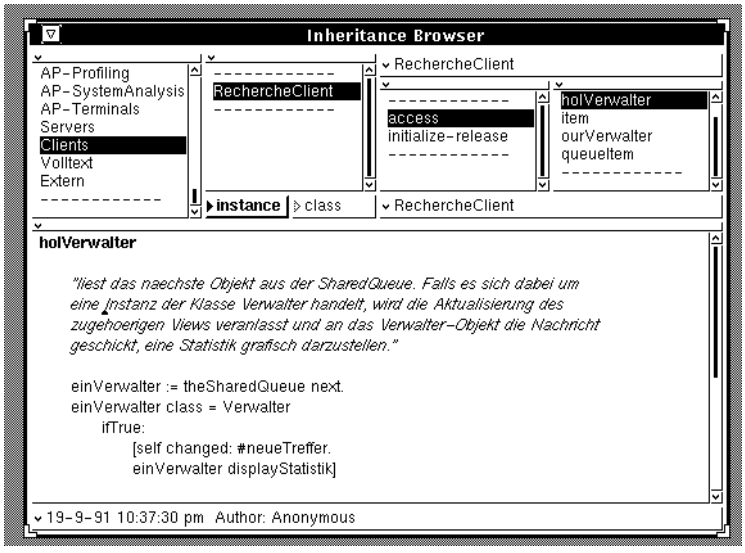


**Abb. 4:**

Definition der Methode *holVerwalter* im Smalltalk 80 Browser (es handelt sich dabei um eine modifizierte Version, den sog. "Inheritance-Browser", der Bestandteil des "Application Organizers" von *Instantiations, Inc.* ist

Das Abstraktionsniveau der Kommunikation wird hier eindrucksvoll verdeutlicht. Es kann allerdings nicht übersehen werden, daß dieses Abstraktionsniveau an gewisse Voraussetzungen

geknüpft ist, deren Erfüllung in einer verteilten heterogen Umgebung eine Reihe von Problemen aufwirft. Im Unterschied zu dem zuvor dargestellten RPC-Beispiel haben wir uns bei Smalltalk darauf beschränkt, zwei Prozesse auf einem Rechner - oder genauer: in einem Smalltalk -Image - zu betrachten. Ein Image in Smalltalk besteht aus der Beschreibung einer i.d.R. einige hundert Klassen umfassenden Hierarchie zusammen mit einer Menge aktiver Objekte. Es ist damit vergleichbar mit einem Ausschnitt des Arbeitsspeicher. Um allerdings Portabilität zu ermöglichen, erfolgt die Abbildung in eine vom Smalltalk-Compiler erzeugte Zwischenrepräsentation, auf der dann der Interpreter ansetzt[1] (eine umfassende Beschreibung der Funktionsweise und Implementierung von Smalltalk findet sich in [7]).

Für unseren Blickwinkel ist dabei von besonderer Bedeutung, daß Smalltalk ein außerordentlich hohes Integrationsniveau bietet. Dies wird vor allem durch zwei Instrumente erreicht - die miteinander in Zusammenhang stehen. Da ist zum einen die *konsequente Vermeidung von Redundanz*. Dadurch, daß Klassen auch Methoden enthalten, müssen in den aus ihnen instantiierten Objekten diese Methoden nicht neu abgelegt werden. Es genügt eine Referenz auf die Methode der zugehörigen Klasse. Vererbung verbessert zudem die Möglichkeit, vorhandene Methoden zu nutzen ohne sie zu kopieren. Innerhalb eines Images können auch komplexe Objekte von mehreren Prozessen genutzt werden. Auf diese Weise wird die Übertragung der Trefferliste realisiert: das Verwalter-Objekt übergibt allein eine Referenz auf die Liste. Der zweite Punkt ist für die Kommunikation innerhalb des Images von großer Bedeutung: die Klassenhierarchie von Smalltalk stellt ein umfangreiches *semantisches Referenzsystem* dar. Die Beschreibung (Strukturen und Methoden) aller Klassen sind jedem Objekt zugänglich. Auf diese Weise kann jedes Objekt auf das Protokoll eines anderen Objekts über dessen Klassenbeschreibung zugreifen.

Die Beschränkung auf Prozesse innerhalb eines Arbeitsspeichers stellt natürlich eine wesentliche Vereinfachung gegenüber der Kommunikation in verteilten Systemen dar. Hier liegt der Einwand nahe, daß der objektorientierte Ansatz keine wesentliche Erweiterung gegenüber konventionellen Systemen bietet: auch in traditionellen Programmiersprachen (wobei hier C besonders hervorzuheben ist) können zwischen Prozessen Referenzen auf Daten und Funktionen ausgetauscht werden - Unterstützung durch ein Multitasking-Betriebssystem mit geeigneten Verfahren zur Interprozeß-Kommunikation vorausgesetzt. Aber: so wenig sachdienlich eine überzeichnete Darstellung der Möglichkeiten objektorientierter Systeme ist, so falsch wäre es, das Kind mit dem Bade auszuschütten. Das Versenden von Objekten anstatt des Hantierens mit Pointern erlaubt ein höheres Abstraktionniveau und damit letztlich effizienteres Programmieren, höhere Sicherheit und bessere Wartbarkeit.

Wie kann man nun Objekte von einem Smalltalk-Image in ein anderes senden? Da die Semantik eines Objekts in seiner Klasse und den in der Hierarchie übergeordneten Klassen festgelegt ist, müßte die komplette zugehörige Klassenhierarchie mit dem Objekt übermittelt werden - falls das Ziel-Image nicht ebenfalls über die Klasse des Objekts verfügt. Die Übermittlung kompletter Klassenbeschreibungen ist wenig praktikabel: sie erfordert einen hohen Zeitaufwand (u.U. müssen Methoden neu kompiliert werden, in jedem Fall ist eine Einordnung in das bestehende Image vorzunehmen) und birgt die Gefahr von Namenskonflikten, die nicht maschinell bereinigt werden können. Wenn Sender und Empfänger die gleiche Klassenhierarchie

1. Zur Erhöhung der Performance verwendet Smalltalk 80 einen Cache-Speicher, in dem einmal interpretierte Methoden in Maschinencode abgelegt werden können, so daß sie beim nächsten Auruf - hinreichende Größe des Cache-Speichers vorausgesetzt - direkt ausgeführt werden können.

aufweisen (als gemeinsames semantisches Referenzsystem), müssen nicht mehr Objekte mit all ihren Methoden und den Klassenbeschreibungen ihrer Instanzvariablen transferiert werden, es reicht vielmehr, den jeweiligen *Zustand* des Objekts zu übermitteln. Dies ist zwar nicht trivial: Objekte können eine komplexe, z.B. rekursive Struktur aufweisen (vgl. Lamersdorf), aber wesentlich leichter zu realisieren als die Übermittlung von Klassen mitsamt ihrer Methoden. Wir haben in der GMD eine solche Kommunikation zwischen zwei Images prototypisch realisiert. Der Zustand eines Objekts (und der Objekte auf die es referiert) wird zusammen mit Referenzen auf Klassen auf einen Stream abgebildet. Dieser Stream wird über die C-Schnittstelle auf ein Array of Byte abgebildet und an die Zielmaschine geschickt, wo aus dem Stream der Objektzustand rekonstruiert wird. Auch wenn dieses Verfahren grundsätzlich funktioniert, sind doch gravierende Nachteile unverkennbar. So nehmen Konstruktion und Rekonstruktion der Streams sehr viel Zeit in Anspruch. Daneben ist das Risiko hoch, das Images im Zeitverlauf inkompatibel werden: Smalltalk ist als grundsätzlich offenes System konzipiert, d.h. der Anwender kann nahezu sämtliche Klassen verändern. Es ist also nicht einmal garantiert, daß gewisse Basisklassen (die als solche in Smalltalk ohnehin nicht ausgezeichnet sind) in zwei verschiedenen Images die gleiche Bedeutung haben.[1] Letztlich werden durch einen solchen Ansatz wesentliche Vorteile von Smalltalk wie Redundanzminimierung oder ein zentrales Ressourcenmanagement aufgegeben.

Solche Integritätsprobleme sind nicht neu. Sie sind auch für konventionelle RPC-Modelle nicht zu vermeiden - gleichwohl in einem objektorientierten System wie Smalltalk eine sehr viel höhere Komplexität zu verzeichnen ist. Ein wichtiges Instrument zur Förderung der Integrität von Informationssystemen sind Datenbankmanagement-Systeme (DBMS). Im Zusammenhang mit objektorientierten Ansätzen ist hier vor allem an sog. objektorientierte DBMS (OODBMS) zu denken.

### Objektorientierte Datenbankmanagement-Systeme

Datenbankmanagement-Systeme leisten einen großen Beitrag zur Integration von Informationssystemen. Dies gilt vor allem für die Reduktion von (Daten-) Redundanz. Aber auch das zentrale im Data-Dictionary verwaltete Datenschema ist von erheblicher Bedeutung. Es beschreibt u.a. (wenn auch mit eingeschränkten Mitteln) die Semantik der gespeicherten Daten. Damit kann es für die beteiligten Anwendungen die Rolle eines Referenzsystems übernehmen. So können Anwendungen (z.B. in einer Vorgangsbearbeitung) einander anstatt der zu verarbeitenden Daten Referenzen auf dieselben zusenden - unter Rückgriff auf den einheitlichen Namensraum des DBMS. Zudem können die im Data Dictionary abgelegten Strukturbeschreibungen zur Koordination der Anwendungsentwicklung genutzt werden. Referenzsysteme sollten nicht nur für alle Betroffenen verfügbar sein, sie sollten zudem stets in einem konsistenten Zustand sein. Die Integrität von Datenbankschemata kann mit Hilfe mechanischer Schutzmechanismen und dedizierter Zugriffsregelungen (hier ist an die Rolle des Datenbankadministrators zu denken) wirksam unterstützt werden.

Ähnlich wie DBMS in konventionellen Informationssystemen versprechen OODBMS Inte-

1. Sobald eine Kommunikation mit konventionellen Systemen intendiert ist, kann ohnehin nicht auf dem Niveau von Objekten kommuniziert werden. In diesem Fall sind Smalltalk-Objekte bzw. deren Zustände in Instanzen geeigneter Datentypen der konventionellen Sprache zu transformieren. So bietet Smalltalk 80 z.B. eine Schnittstelle zu C

grationsprobleme in verteilten objektorientierten Systemen zu überwinden. Die Voraussetzungen dazu scheinen auf den ersten Blick hervorragend. So kann mit der zentralen Ablage von Objekten grundsätzlich die Redundanz innerhalb eines Informationssystems weiter verringert werden als dies bei der Beschränkung auf Daten der Fall ist. Darüber hinaus könnte das Schema eines OODBMS, nämlich eine Klassenhierarchie, genutzt werden, um Kommunikation in verteilten Systemen zu unterstützen: ähnlich wie Prozesse innerhalb eines Smalltalk-Images auf eine gemeinsame Klassenhierarchie referieren können, könnten hier Anwendungen in einer verteilten Umgebung auf die Klassen eines zentralen Klassen-Schemas referieren. In beiden Fällen ist allerdings ein Problem zu überwinden, das selten thematisiert wird: Anwendungen können nicht ohne weiteres auf Objekte in einem OODBMS zugreifen - auch wenn dies eine scheinbar inhärente Funktion eines solchen Systems ist. So weckt ein Anbieter in seinem Prospekt den Eindruck, der Zugriff auf Objekte sei allenfalls bei der Verwendung unterschiedlicher Sprachen in den Anwendungen ein Problem - das man aber auch überwunden hat: “For example, a C++ program can read and write objects entered into the database by a Smalltalk program.”[1] Dazu könnte man lakonisch anmerken: “Und - was passiert dann?” oder aber nach den Voraussetzungen fragen. Dabei steht man vor einer Schwierigkeit, die bereits im Zusammenhang mit Smalltalk deutlich wurde: Objekte enthalten Referenzen auf Methoden. Wie sollen diese Referenzen beim Empfänger eines Objekts befriedigt werden? Anders formuliert: wenn der Empfänger (wie in unserem Smalltalk-Beispiel) dem Objekt eine Nachricht schickt, wo soll die entsprechende Methode ausgeführt werden? Angesichts dieses Problems verwundert es wenig, daß es die meisten OODBMS gar nicht erlauben, ein Objekt in seiner Gesamtheit abzulegen. Vielmehr können nur Zustände gespeichert werden. Um den Zustand eines Objekts korrekt abzulegen, muß seine Klasse im Schema des OODBMS beschrieben sein. Aber auch die Anwendung muß zur Instantiierung des Objekts über eine Beschreibung seiner Klasse verfügen, da nur dadurch die Referenzen auf Methoden befriedigt werden können. Für OODBMS, die Anwendungen in einer heterogenen Umgebung Zugriff auf abgelegte Methoden erlauben, bleiben im wesentlichen zwei Varianten. Die Beschreibung der Klasse eines Objekts kann in maschinenunabhängiger Form übermittelt werden. Sie muß dann vom Empfänger in geeigneter Weise in seine Umgebung integriert werden (mit dem bereits erwähnten Risiko, daß hier Konflikte auftreten können). Angesichts der i.d.R. unzureichenden Performance kommt dieses Verfahren allenfalls im Rahmen der Software-Entwicklung in Betracht. Daneben gibt es die Möglichkeit, Methoden auf Anforderung innerhalb des OODBMS auszuführen. Dies aber setzt voraus, daß der Anwendung wenigstens die Klassenbeschreibungen der jeweils zu übertragenen Parameter bekannt sind. Solche Einschränkungen sind allerdings wenig erbaulich, da sie den Programmierer auf ein niedrigeres Abstraktionsniveau zwingen.

Der sinnvollste Ansatz bei verteilter Verarbeitung in heterogenen Systemen ist m.E. zur Zeit darin zu sehen, auf jedem Rechner im Netz eine Kopie des zentral geführten Klassenschemas zu halten. Die damit verbundene Redundanz läßt sich in akzeptabler Weise kontrollieren, indem z.B. allein das OODBMS für die konsistente Aktualisierung der lokalen Kopien zuständig ist. Daneben ist ein anderer Ansatz zu berücksichtigen, der auch von einigen Anbietern explizit empfohlen wird: sämtliche Anwendungen eines Informationssystems laufen innerhalb des OODBMS oder aber wenigstens auf dem gleichen Rechner ab. Also zurück zum alten Mainframe-Ansatz. Die Kommunikation mit den verteilten Arbeitsplätzen könnte dann z.B. über X-Window-Server erfolgen. Eine m.E. durchaus reizvolle Vorstellung, die allerdings u.a. wegen

1. Eine selbstkritische Denkschrift zu den häufig unseriösen Verlautbarungen der Branche findet sich in [8].

der mangelhaften Unabhängigkeit vom Hardwaresystem sorgfältig geprüft werden sollte. Die Abbildungen 5 und 6 veranschaulichen die beiden praktikablen Basismodelle für die Nutzung von OODBMS bzw. die Ermöglichung von Kommunikation auf Objekt-Niveau. Es sind natürlich auch Mischformen denkbar.

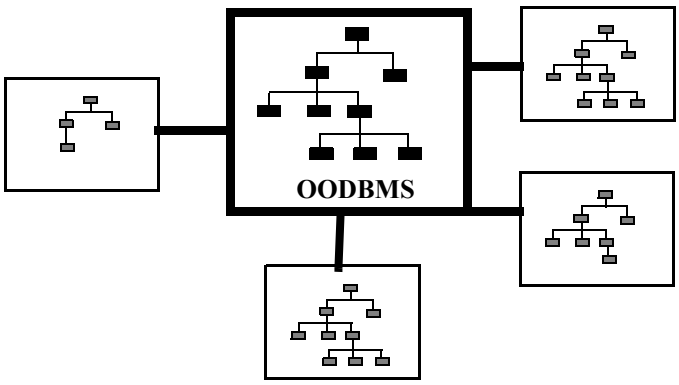


**Abb. 5:**

Die Klassenhierarchie wird zentral vom OODBMS-Administrator gepflegt. Die Anwendungsrechner erhalten auf Veranlassung des Administrators eine Kopie des gesamten oder von Teilen des Schemas. Die Referenzen auf Methoden werden so innerhalb der Anwendungsrechner befriedigt.

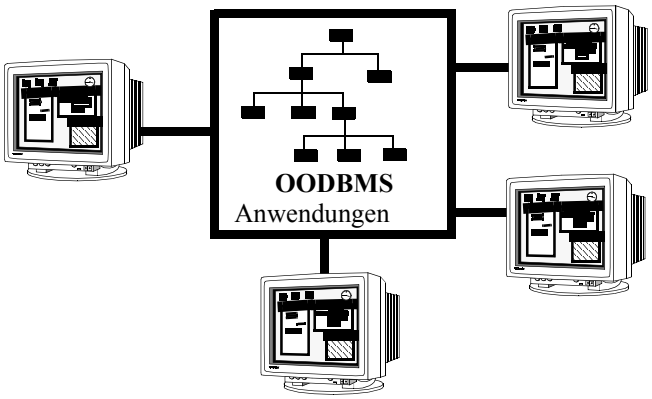


**Abb. 6:**

Die Klassenhierarchie wird zentral vom OODBMS-Administrator gepflegt. Sämtliche Anwendungen laufen ebenfalls auf dieser Maschine. Die Rechner an den Arbeitsplätzen sind allein für die Realisation der Benutzerschnittstelle zuständig (z.B. X-Window-Server).

Zusammenfassend läßt sich feststellen, daß ein OODBMS tatsächlich ein ausgezeichnetes Instrument zur Integration der Anwendungen eines Informationssystems darstellt. Neben der Bewältigung von Synchronisation- und Performance-Problemen ist es wesentlich, ein einheitliches Klassenschema für alle zu integrierenden Anwendungen zu entwickeln. Dies ist ein Unterfangen, daß die Komplexität des gewiß nicht trivialen Entwurfs unternehmensweiter Datenmodelle erheblich übersteigt. So ist es u.a. erstrebenswert, die Klassenhierarchie so aufzubauen, daß einzelne Klassen von möglichst vielen Anwendungen verwendet werden können (entweder unmittelbar oder über Vererbung und Spezialisierung). Daneben sind Klassen auf einem hohen Abstraktionsniveau - also mit möglichst viel Anwendungssemantik - wünschenswert.

### Abschließende Bemerkungen

Die Versendung von Objekten erlaubt Kommunikation auf einem sehr hohen semantischen Niveau. Sie eröffnet die Perspektive auf eine Vision, die Tsichritzis [16] “Objectworld” nennt: anstelle von Daten, die vom Sender zum Empfänger geschickt werden, migrieren Objekte durch das Informationssystem, suchen u.U. selbst ein geeignetes Zielobjekt, mit dem sie wiederum kommunizieren. Der Umstand, daß wir dieser Vision heute noch nicht näher gekommen

sind, hat sicherlich mit der zögerlichen Verbreitung objektorientierter Software-Technologie zu tun. Vor allem im Bereich objektorientierter DBMS herrscht selbst in Fachkreisen ein hohes Maß an Verwirrung[1] - nicht zuletzt durch die unseriösen Verlautbarungen der Anbieter verursacht. Die Unsicherheit wird zudem dadurch erhöht, daß wesentliche Funktionen von OODBMS von zukünftigen objektorientierten Betriebssystemen [10, 13] übernommen werden könnten und die Verlautbarungen großer Anbieter über zukünftige Software-Entwicklungsstrategien noch wenig konkret sind (z.B. IBM mit AD/Cycle [12]). Von entscheidender Bedeutung für die Umsetzbarkeit der skizzierten Vision aber sind weniger Aspekte der technischen Realisierung (auch wenn es hier noch viel zu tun gibt), sondern vielmehr die Entwicklung geeigneter Referenzsysteme in Form von Klassenhierarchien. Es gibt gute Gründe, solche Referenzsysteme nicht auf ein Unternehmen zu beschränken (als dessen zentrales "Objektmodell"), sondern sie für einen möglichst großen Kreis von Anwendern zu entwerfen. Rechnergestützte Kommunikation findet mehr und mehr auch zwischen Unternehmen statt. Darüber hinaus ergibt sich so die Chance einer geradezu phantastischen Verbesserung des Preis-Leistungsverhältnisses von Software: durch vielfache *Wiederverwendung* sorgfältig entworfener und implementierter Klassen. Es liegt auf der Hand, daß dazu eine Vereinheitlichung bzw. Standardisierung nötig ist. Erste Aktivitäten dieser Art sind seit einiger Zeit zu verzeichnen. So gibt es Konsortien, deren Ziel es ist, umfangreiche Klassenbibliotheken zu entwickeln [11], daneben sind Vereinbarungen über Architekturen, Sprachen und Schnittstellen Gegenstand von Standardisierungsbemühungen [1, 14]. Diese Aktivitäten beziehen sich allerdings noch auf systemnahe Bereiche. Sie schließen die Anwendungsebene weitgehend aus.[2] Gerade dort sind jedoch gegenüber der heutigen Situation besondere Vorteile sowohl für Kommunikation als auch für Wiederverwendbarkeit zu erwarten. Die Entwicklung von Anwendungs-Referenzsystemen setzt die Modellierung des jeweiligen Anwendungsbereichs voraus. Dies ist nicht allein eine wissenschaftliche, sondern auch eine kulturelle Herausforderung: es geht letztlich darum, sich auf eine einheitliche *Sprache* zur Beschreibung von Realitätsausschnitten zu einigen.

In der GMD läuft seit einem knappen Jahr ein Projekt mit dem Ziel, ein prototypisches Unternehmens-Referenzmodell zu entwickeln [4]. Um eine aussichtsreiche Grundlage für Generalisierbarkeit und Akzeptanz zu schaffen, versuchen wir dabei u.a. die objektorientierte Rekonstruktion etablierter terminologischer Bezugsrahmen der Betriebswirtschaftslehre (ein herausragendes Beispiel dafür ist das Rechnungswesen). Zur Modellierung verwenden wir Smalltalk - nicht zuletzt wegen der Notwendigkeit häufiger Modifikationen.

## Literatur


**1 Atwood, T.:** Why the OMG Object Request Broker should mean good news for object databases . In: Journal of Object-Oriented Programming, Vol. 4, No. 4, July/August 1991 , S. 8-11

**2 Birrell, A.D.; Nelson, B.J.:** Implementing Remote Procedure Calls. In: ACM Transactions on Computer Systems, Vol. 2, No. 1, Feb. 1984 , S. 39-59


1. So tauchen in der news-Konferenz "comp.object" immer wieder Anfragen der Art "somebody out there who can tell me what an OODBMS really is or should be" auf.
2. Von den gegenwärtigen Datenübertragungs-Standards kommt ODA/ODIF [5] dieser Vorstellung nahe. Zwar wird hier nicht notwendig eine objektorientierte Software-Umgebung unterstellt, aber die in ODA dokumentierte Architektur ist objektorientiert.

**3 Comer, Douglas E.:** Internetworking with TCP/IP - principles, protocols, and architecture. London u.a. 1988

**4 Frank, U.; Klein, S.:** Computer Integrated Enterprise oder: Zur Bedeutung anwendungsnaher Referenzmodelle. Sankt Augustin 1991

**5 Frank, U.:** Anwendungsnahe Standards der Datenverarbeitung: Anforderungen und Potentiale. Illustriert am Beispiel von ODA/ODIF und EDIFACT. In: Wirtschaftsinformatik, Heft 2 , 1991, S. 100-111

**6 Frank, U.:** Ein Literatur- und Dokument-Retrievalsystem für den Einsatz in heterogenen Netzen. Sankt Augustin 1991

**7 Goldberg, A.; Robson, D.:** Smalltalk-80 - the language. Reading, Mass. 1989

**8 Ingari, F. :** The object database market: stranger than it needs to be? In: Journal of Object-Oriented Programming, Vol. 4, No. 4, July/August 1991 , S. 16-18

**9 Lamersdorf, W. :** Communicating recursive objects. Heidelberg 1986

**10 Makpangou, M.; Shapiro, M.:** The SOS object-oriented communcation service. Institut National de Recherche en Informatique e en Automatique. Rocquencourt 1988

**11 Meyer, B.:** Lessons from the Design of the Eiffel Libraries. In: Communications of the ACM, Sept. 1990, S. 68-89

**12 Schussel, G. :** The Promise and the Reality of AD/Cycle. In: Datamation Sept. 15 , 1990, S. 69-73

**13 Shapiro, M.:** Prototyping a distributed object-oriented operating system on UNIX. Institut National de Recherche en Informatique e en Automatique. Roquencourt 1989

**14 Soley, R.M.:** Object Management Architecture Guide 1.0 . Framingham, Ma. 1990

**15 SUN Microsystems:** OS 4.0, Network Programming, Writing Device Drivers. Mountain View 1988

**16 Tsichritzis, D.:** Objectworld. In: Tsichritzis, D. (Hg.): Office Automation. Berlin, Heidelberg u.a. 1985 , S. 379-398

**17 Wileden, J.C.; Wolf, A.L.; Rosenblatt, W.R.; Tarr, P.L.:** Specification-Level Interoperability. In: Communications of the ACM, May 1991, S. 72-87