Objektorientierte Entwicklung verteilter Applikationen

Andres Koch
Object Engineering GmbH
Birmensdorferstrasse 32
CH-8142 Uitikon Waldegg
Tel +41 (0) 44 400 47 00
Fax +41 (0) 44 400 47 07
email: info@objeng.ch

Erschienen in
"Offene Systeme" (1995) Band 4/ Nr. 2, Mai 95, S. 70-74 Springer Verlag International Heidelberg

Einleitung

Verteilte Systeme sind auf breiter Basis im Gespräch und Client/Server-Applikationen, so bekommt man den Eindruck, seien nicht nur das "Gelbe vom Ei" sondern heute das Normalste vom Normalen. Macht man sich aber mehr Gedanken über die richtige Verteilung und wie man solche Applikationen überhaupt entwirft und realisiert, dann stellt man fest, dass man mehr oder weniger auf sich selbst gestellt ist und das "Normalste vom Normalen" zum Novum wird.

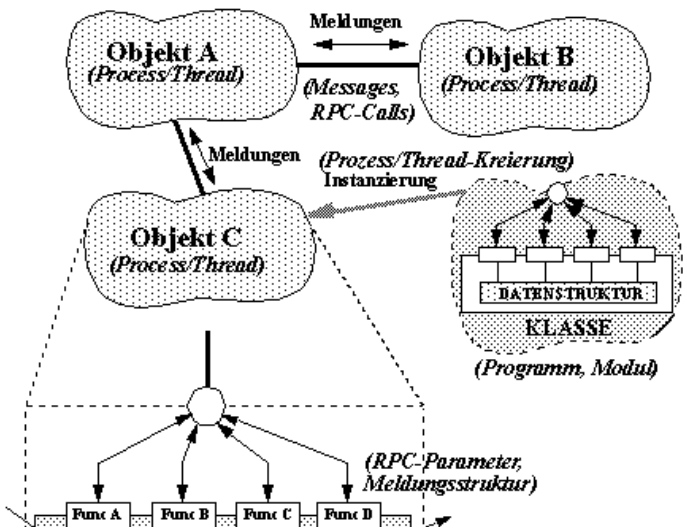
Entwickeln wir Applikationen nach konventionellen Kriterien als Monolythen, dann stehen uns entweder bekannte strukturierte Methoden oder in neuerer Zeit sogar objektorientierte Methoden zur Verfügung. Wie steht es aber bei Applikations-Programmen in verteilten Systemen, müssen wir auf den Methodik-Propheten aus fernen Landen oder auf sonst einen Guru warten, der uns eine neue wundervolle Methode bringt? Und mit der neuen Methode und dazugehöriger Notation wird unser Turm von Babylon wieder um eine Sprache reicher. Im vorliegenden Artikel beabsichtigt der Autor einen Weg aufzuzeigen, auf dem man mit einer herkömmlichen, bekannten Methode für die Entwicklung von Applikationen für verteilte Systeme zu Hilfe nehmen kann und damit den Notations-Wirrwarr in Grenzen hält.

Als Methode wird hier die objektorientierte Entwurfsmethode von Booch verwendet, welche einerseits durchgängig bis zur Implementation ist und anderseits alle notwendigen Notations-Elemente enthält, die für die Darstellung der Designs notwendig sind.

Analogie zwischen Objekten und Prozessen

Betrachtet man eine in Prozesse aufgeteilte Applikation und eine in Objekte aufgeteilte Applikation, kann man leicht eine Analogie zwischen objektorientierten Systemen und prozessorientierten Systemen aufzeigen.

Abb. 1 Analogie-Modell von OO-Prozessen



Prozesse (Tasks, Threads) gehören zu den wichtigsten Komponente einer verteilten Applikation. Weiter ist natürlich die früher in diesem Kreis [4] behandelte Interprozesskommunikation eine zentrale Komponente. In Tabelle 1 [Tab. 1] wird der Zusammenhang zwischen Objekten und Prozessen aufgezeigt. Darin sieht man, dass die Kapsulierung von Objekten natürlich auch bei Prozessen in noch viel stärkerem Mass, nämlich durch das Betriebssystem gesichert, vorhanden ist. Jeder Prozess hat auch private Datenbereiche und wenn es sich um einen Server handelt, dann kapsuliert er ja die von ihm verwaltete Ressource ab. Das zeigt dass wir Prozesse eigentlich als Objekte mit Nebenläufigkeit (active Objects gemäss Booch [1,2] darstellen können. Dies insbesondere wenn wir auch die Kommunikation zu Objekten anschauen, welche in einem objektorientierten System in der Regel aus einem Prozeduraufruf der entsprechenden Methode abläuft.

Tabelle 1 Vergleich Objekt mit Prozess

Objektorientiertes System	Prozessorientiertes System
Klasse	Programm und Datenstrukturdes Prozesses
Objekt (Instanz einer Klasse)	Gestarteter Prozess (Prozess-Instanz, Task, Thread)
Methode	Meldungs-Typ und -Struktur und die damit verbundene Funktion innerhalb des Prozesses
Aktivierte Methode	Meldung mit bestimmtem Typ resp. Struktur an den Prozess senden, der diese in der zugehörigen Funktion verarbeitet und das Resultat als Meldung zurücksendet.
Polymorphismus	Entsprechende Meldungstypen, welche an den gleichen Prozess gesendet werden.
Vererbung	Prozess leitet Meldungenan einen Basis-Prozess weiter, von welchem er logisch abgeleitet ist Evtl. kann dieser Basisprozess gestartet werden,wennnoch nicht vorhanden (Dynamik):

Im prozessorientierten System wird dem Prozess entweder auf synchrone Art eine Prozedur mit dem sogenannten Remote Procedur

Call aktiviert, was natürlich dem Methodenaufruf des Objektes sehr nahe wenn nicht sogar gleich kommt. Wie wir wissen sind aber auch RPCs auf Meldungen aufgebaut und so wird für den asynchronen Betrieb auch der "Message-Passing"-Mechanismus das analoge Mittel zum Aktivieren der Methode des angesprochenen Objektes (Prozess) verwendet. Diese Analogie wird sogar durch die Pioniersprache der Objektorientierten - Smalltalk - untermauert, in welcher man vom "Senden einer Meldung an das Objekt" spricht und damit die Aktivierung einer Methode des Objektes meint. Während bei den eigentlichen Methodenaufrufen der Objekte und bei RPC die Selektion der richtigen Methoden-Prozedur implizit durch den Mechanismus geschieht, muss bei Message-Passing die Selektion der richtigen Methode auf der Prozess-Seite explizit aufgrund des Message-Types geschehen. Was uns hier einerseits etwas expliziten Mehraufwand gibt, bringt uns auf der andern Seite wiederum zusätzliche Flexibilität.

Polymorphismus funktioniert auch in verteilten Systemen mit Prozessen. So würden wir zum Beispiel in einer Grundklasse die Methode 'getStatus' definieren, was uns erlaubt von jedem je von dieser Klasse abgeleiteten Subklasse und der daraus instanzierten Objekten eine Status-Abfrage zu machen. Dasselbe funktioniert natürlich auch bestens für Prozesse und bekommt eben als "dynamische Bindung" auch eine grosse Bedeutung. Die Implementation würde über eine Meldung 'getStatus' gemacht, die von allen dieser Grundklasse angehörenden Prozesse verstanden wird. Dies wird in der Praxis sehr wichtig für Versionen-Abfragen, Diagnostik, Statusabfragen und andere Funktionalitäten.

Die Vererbung scheint auf den ersten Blick vielleicht nicht so offensichtlich, wird dann aber bei näherer Betrachtung sich um so stärker zeigen, um was es sich auch im Objektorientierten handelt. Bei Objekten wird eine Aufgabe (Funktion), die einer Oberklasse angehört auch vom entsprechenden Code bearbeitet, welcher im direkten Zusammenhang mit der Oberklasse definiert und implementiert wurde. Bei einem Prozess wird eine ankommende Meldung, die eine Funktion eines vererbten Prozesses anspricht, welche aber im Superprozess gelöst ist, einfach an diesen weitergeleitet. Das heisst der Superprozess ist auch als eigener Prozess im (oder irgendeinem) System vorhanden. Der Subprozess wirkt in diesem Fall als ein Verteiler (Dispatcher) und wird selbstverständlich auch mit Mehrfachvererbung fertig, indem er die Meldung an den richtigen Prozess weiterleitet.

Wenn man "Objektorientiert" hört denkt man heute vielleicht zuerst an Objektorientierte Programme. Man könnte Objekte, die in objektorientierten Sprachen spezifiziert und durch Compiler in die Maschinensprache übersetzt werden, als Mikro-Objekte bezeichnen.

In der hier gezeigten Analogie ist aber die Ebene von Makro-Objekten zu verstehen, welche nicht auf die Granularität von Listenelemente reduziert werden sondern sich eher auf Services und Client-Prozesse abstrahiert. Diese Art ist heute natürlich nicht weniger aktuell. Denn mit der "Common Object Request Broker Architecture" (CORBA) [6] der Object Management Group (OMG) sind wir voll und ganz mit dem Gebiet von verteilten Objekten konfrontiert. Und was sind diese verteilten Objekte in Tat und Wahrheit? Klar es sind Prozesse, die auf irgendeinem Rechner im System ihren Lebenszyklus durchlaufen.

Wenn wir an Wiederverwendung von Objekt-Implementationen (Klassen) denken, dürfen wir uns keiner Euphorie oder Utopie hingeben. Auf Basis von Mikro-Objekten hat es sich nämlich in der Praxis gezeigt, dass noch zuviel Individualität und Unorganisiertheit vorhanden ist, und dass die angebotenen Klassenbibliotheken viel zu verschieden aufgebaut sind, um sie ohne Umschweife in das eigene, bereits auf andere Klassenbibliotheken basierte Applikation einzubauen.

Das hat auch OMG (Object Management Group) erkannt und sich statt auf die Normierung von Objekten auf Programmierebene haben sie sich auf die Definition einer übergeordneten Architektur von Makro-Objekten konzentriert und dafür eine Architektur (CORBA) normiert, die die Schnittstellen und nicht zu vergessen die Semantik zwischen Objekten festlegt. Dieses Konzept ist eine weise Voraussicht auf das, was in den nächsten Jahren auf uns zukommen wird. Die ersten Resultate von CORBA zeigen eine wesentlich bessere Effektivität und bessere Verbreitung, als das z.B. OSF/DCE in einem verwandten Gebiet vorweisen kann. Object

wesentlich bessere Effektivität und bessere Verbreitung, als das z.B. OSF/DCE in einem verwandten Gebiet vorweisen kann. Object Request Brokers, die nach CORBA-Spezifikationen entwickelt sind, werden die Entwicklung verteilter Anwendungen stark beeinflussen und es ist auch sinnvoll darauf zu bauen, um von anderen Ressourcen (Services in Form von Objekten) Gebrauch zu machen.

Wie aber steht es mit den Entwurfs-Verfahren solcher Applikationen. Entweder verwendet man eine adhoc-Methode aus eigener Feder, was grundsätzlich nicht schlecht sein muss, doch handelt man sich dabei ein, dass andere Entwickler die eigene Notation lernen müssen. Dabei läuft man Gefahr, dass nur wenige oder eben niemand ihr Applikations-Entwurf mangels Notationsklarheit verstehen kann oder will. Auch hilft das Warten auf unser akademischen Päpste nicht, die vielleicht mit einem Glückstreffer in naher Zukunft eine dazu geeignete Methode und Notation erfinden, die nach langer Polemik und philosophischem Gerangel in 10 Jahren die Industrie davon überzeugt, das Richtige zu sein. Die objektorientierte Technologie hat schlussendlich auch erst vor 25 Jahren ihre Geburtsstunde erlebt.

Aus Erfahrung mit der Entwicklung verteilter Anwendungen ist der Autor überzeugt, dass eine herkömmliche objektorientierte Methode die Aspekte für den Entwurf verteilter Anwendungen weitgehend abdeckt. Die in verschiedenen Projekten eingesetzte Booch-Methode hat hier sehr gute Dienste geleistet. Natürlich könnte auch OMT von Rumbaugh oder die Methode von Schlaer/Mellor eingesetzt werden, doch soll nachfolgend hauptsächlich auf die Entwurfsmethode respektive auf die Notation von Grady Booch eingegangen werden.

Grobentwurf

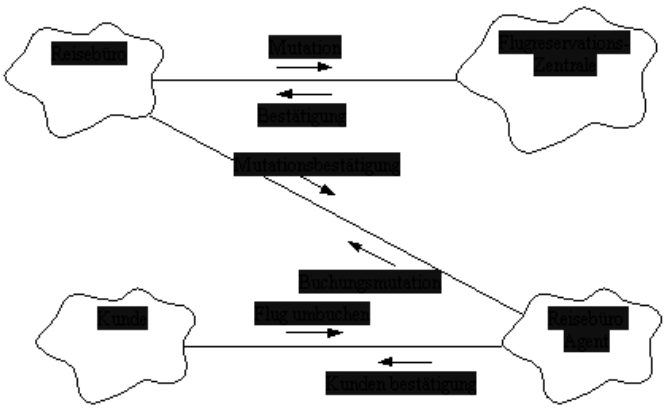
Vorausgesetzt die Analyse, welche ja noch wenig auf Verteilung einzugehen hat, liege vor dann geht man zum Entwurf der Implementation über. Gemäss Booch bestimmt man in diesem Schritt die Initial-Architektur.

Wenn man beabsichtigt einen ORB nach CORBA einzusetzen, dann stellt dies bereits einen relevanten Teil der Architektur dar, womit wir bereits auf der Basis vom Design eine Wiederverwendung erreichen.

Ohne ORB würde man sich eine Architektur zurechtlegen, welche die Aufteilung der Aufgaben in mehrere logische evtl. verteilte Module manifestiert. Diese Module, welche in der Regel eigene Prozesse (oder Hauptprozesse) sind, können mittels einem Objektszenario dargestellt werden. Der Leser möge sich daran erinnern, dass wir uns auf Ebene von Makro-Objekten (also Prozessen) und nicht auf der Basis von deren Mikroorganismen konzentrieren. Letztere werden dann ganz "normal'' mit der eigentlichen OO-Entwurfsmethode gelöst.

Wir zeichnen die Prozessmodule also als Objekte auf und zeichnen die Verbindungen zwischen diesen Modulen auf .

Abb. 2 Grobentwurf mittels Objekt-Szenario



Schnittstellen

Einer der wichtigsten Merkmale der objektorientierten wie auch bereits der objektbasierten Methoden sind neben Vererbung und Polymorphismus die Bildung von klaren Schnittstellen. Ohne Schnittstellen keine Kommunikation (sowohl auf logischer wie auch physikalischer Ebene). Mit den richtigen Schnittstellen sichern wir uns Flexibilität, Modifikationsfähigkeit und Wartbarkeit in Zukunft zu. Also ein wichtiges Kapitel. Dazu können wir uns auch für verteilte Anwendungen wiederum Regeln der objektorientierten Entwicklung zu Hilfe nehmen. Bertrand Meyer [5] hat in seinem Buch "Object Oriented Software Construction'' dazu im Kapitel 2.2 Modularität dazu unter anderem folgende Hinweise gegeben:

- Wenige Schnittstellen: Jedes Modul sollte mit möglichst wenig andern kommunizieren.
- Schmale Schnittstellen (Lose Kopplung): Wenn zwei Module überhaupt miteinander kommunizieren, sollten sie sowenig Informationen wie möglich austauschen.
- Explizite Schnittstellen: Wenn zwei Module A und B kommunizieren, dann muss das aus dem Text A oder B oder beiden hervorgehen.

Hand aufs Herz geehrte Leser, haben sie sich beim Entwurf einer rein objektorientierten Applikation darüber Gedanken gemacht. Wohl waren sie mit anderen wichtiger scheinenden Problemen beschäftigt.

Bei verteilten Applikationen, wo zwischen den "Objekten'' unter Umständen Prozess oder sogar Rechner- und Netzwerkgrenzen überschritten werden müssen, sind diese obigen Empfehlungen von grösserer Bedeutung und können sich stark auf die Performance des Applikationsverhaltens durchschlagen.

Die Schnittstellen sind natürlich am Aehnlichsten zu objektorientierten Sprachen auf RPC (Remote Procedure Calls) für verteilte Applikationen umzusetzen. Methodenaufrufe können direkt auf RPCs abgebildet werden. Wie besprochen ist die Synchronität dabei nicht immer ein gewünschte Einschränkung, wenn man hohe Parallelität erreichen will. Dafür ist Message Passing geeigneter, was sich auch in den Programmen als einfacherer Mechanismus für das Erreichen von Nebenläufigkeit zeigt. Man kann das einfache Prinzip aus dem Administrations-Leben, den Post-Eingangs-, Pendenzen- und Ausgangs-Korb realisiert durch verschiedene Listen (oder Queues) realisieren.

Die Umsetzung eines auf Meldungen basierten Designs bedingt, dass man im Design die verwendeten Meldungen auch aufführt. Dies

kann übrigens elegant via Meldungs-Hierarchien im OO-Entwurf mit einer entsprechender Klassen-Hierarchie gemacht werden.

Bewegt man sich in einer CORBA-Umgebung, dann drängt sich für die genaue Spezifikation der Schnittstellen natürlich die CORBA-IDL (Interface Definition Language (nicht mit RPC-IDL von OSF zu verwechseln) auf. Es ist zu erwarten, dass in absehbarer Zukunft auch OO-CASE-Tools in der Lage sein werden diese Schnittstellen-Sprache generieren zu können.

Die Schnittstellen sind ein wichtiger Faktor, doch was über diese Schnittstellen abläuft wird durch das Protokoll beschrieben.

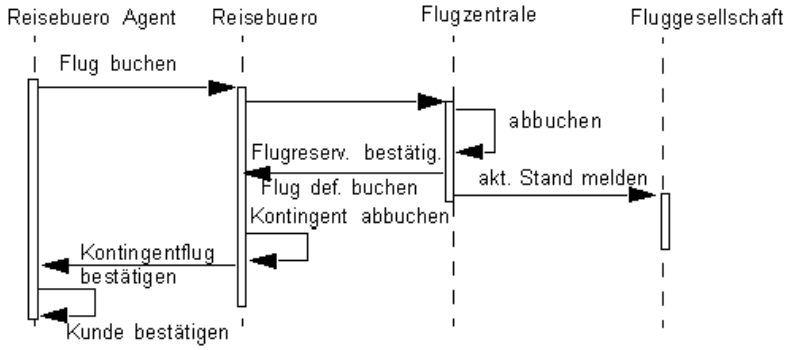
Protokolle

Das Protokoll beschreibt den genauen "Wortwechsel" - wie übrigens auch bei diplomatischen Gesprächen - zwischen kommunizierenden Prozessen.

Die Booch-94-Notation gibt uns zwei Werkzeuge, womit wir das Protokoll sehr gut beschreiben können. Das eine ist das Objektszenario [Abb 2], wie bereits oben erwähnt, worin man mit dem Spezifizieren von Messages, die über die einzelnen Verbindungen geschickt werden, dies nicht nur dokumentieren kann, sondern auch herausfinden kann, ob eine Meldung, die man braucht noch nicht spezifiziert ist.

Um das chronologische Ablaufverhalten besser darstellen zu können, eignet sich das von Jacobson [3] in die Booch-Methode eingeflossene Interaktions-Diagramm. Diese ist eine Methode, die häufig für die Beschreibung der Protokolle zwischen Netzwerkknoten Verwendung findet.

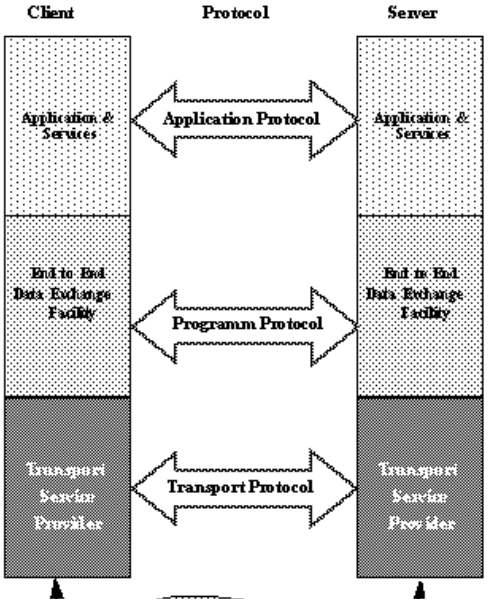
Abb. 3 Interaktions-Diagramm

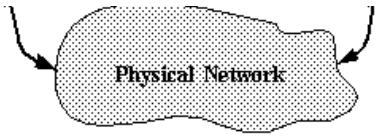


Dieser Teil ist natürlich sowohl bei synchroner wie bei asynchroner Kommunikation wichtig.

In diesem Bereich müssen wir uns im Klaren sein, dass standardisierte Protokolle auf der Ebene von Applikation und Services (vergleiche Abb. 4) noch wenig vorhanden ist. Bis auf Ebene 5 bis 6 des OSI-Modells sind wir recht gut bedient. Bis Level 4 wird heute üblicherweise TCP/IP eingesetzt. RPC, Sockets, Message Passing u.a. decken den Bereich Interprozesskommunikation (Ebene 5 Session und teilweise 6 Präsentation) ab.

Abb. 4 Verschiedene Protokoll-Ebenen





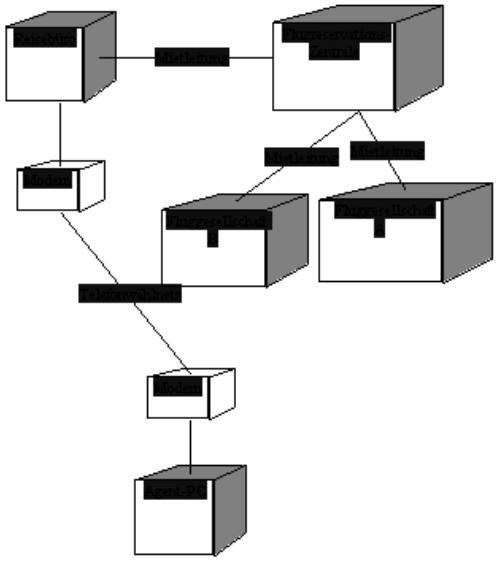
Die Ebene 7 (Applikation) muss aber noch mehr standardisiert werden. Hier wird natürlich einiges durch CORBA abgedeckt, aber um wirklich "verbindbar" zu sein und generell kommunizieren zu können, muss noch einiges definiert werden. Diese OSI-Ebene 7 muss als nach oben offene Skala betrachtet werden, und es ist zu hoffen, dass OMG hier auch noch etwas vorgeben wird.

Klassenbildung

Aufgrund der Modulaufteilung, der Schnittstellen und der Protokolle können die Prozesse spezifiziert werden. Dabei geht man wie bei der OO-Methode sonst vor, achtet aber darauf, dass man Vererbung nicht im Uebermass braucht, da quasi eine Oberklasse bei der Instanzierung durch einen eigenen Prozess repräsentiert wird. In Abbildung 2 als Beispiel kann aber gezeigt werden, dass Meldungen von der Agentur ans Reisebüro zur Reservation eines Fluges teilweise direkt an die Flugreservationszentrale weitergeleitet werden können.

Prozessverteilung

Die Methode von Booch stellt für die Grobdarstellung der Systemarchitektur die Prozess-Darstellung (wäre evtl. besser als System-Darstellung zu bezeichnen) zur Verfügung, welche aufzeigt wie das darunterliegende System hardwaremässig (grobes Blockdiagramm) aufgebaut und verbunden ist. Dies kann natürlich vorallem für verteilte Systeme interessant sein, wenn man damit auch die Verteilung der Prozess-Objekte (Prozesse) aufzeigen kann. Die Notation selbst ist nicht allzu glücklich gewählt und könnte etwas weniger schwerfällig dargestellt sein. Dafür wäre etwas mehr Darstellungsmöglichkeiten für die eigentliche Zuordnung von Prozessen auf den einzelnen Komponenten gewünscht. Dies ist aber, wenn man die ganze Methode und Notation betrachtet ein fast vernachlässigbarer Punkt.



Zusammenfassung

Im vorliegenden Artikel wurde ein Ansatz aufgezeigt, wie man mit einer eingeführten Methodik objektorientiert eine Applikation für verteilte Systeme entwickeln kann. Es wurde dabei mehr Wert auf die Ideenvermittlung als auf Vollständigkeit gelegt. Die Erfahrungen mit der schrittweisen Anwendung dieser Technik waren sehr gut und in Anbetracht des Aufkommens von Konzepten wie CORBA auch sehr zukunftsversprechend.

Literaturverzeichnis

G.Booch, "Object Oriented Analysis And Design with Applications", The Benjamin/Cummings Publishing Company, Inc. New York, 2nd Ed, ISBN 0-80535340-2, 1994.

G.Booch, "Objektorientierte Analyse und Design",Addison Wesley Publishing Company, Bonn, Paris, Reading, Mass u.a. ISBN 3-89319-673-0 (Deutsche Übersetzung von [6]).

Jacobson Ivar etal.,Object Oriented Software Engineering, Addison Wesley,
ISBN 0-201-54435-0, 1993 4th Edition.

Koch, A., "Interprozesskommunikation in verteilten Systemen", Offene Systeme (1993) Band 2/ Nr. 4, November 93, S. 201-209, Springer Verlag International Heidelberg.

Meyer Bertrand, Objektorientierte Softwareentwicklung, Carl Hanser München, Prentice Hall, ISBN 3-446-15773-5, 1990

Object Management Group, "The Common Object Request Broker: Archtecture and Specification", Rev. 1.1, John Wiley&Sons, Inc, New York, OMG Document 91.12.1, ISBN: 0-471-58792-3.