Löst CORBA wirklich alle unsere Probleme?

Walter Bischofberger
Takefive Software
Eidmattstrasse
CH-8032 Zurich
bischofberger@takefive.com

Dirk Riehle
Ubilab, UBS
Bahnhofstrasse 45
CH-8021 Zürich
riehle@acm.org

Auf dem CORBA-Standard basierende Middleware ist heute eine Option, an der man beim Entwickeln von verteilten objektorientierten Anwendungssystemen kaum noch vorbei kommt. CORBA verspricht Objektorientierung, transparente Verteilung, Sprachunabhängigkeit und Interoperabilität, unabhängig von einem bestimmten Hersteller. Löst CORBA damit alle unsere Probleme? Viele Publikationen zu diesem Thema machen unterschwellig solche Behauptungen, so daß in der Praxis heute vielfach die Meinung herrscht, daß dies der Fall sei.

In diesem Artikel arbeiten wir deshalb bewußt die Schwachstellen von CORBA heraus. Dabei betrachten wir die folgenden Aspekte: das Objektmetamodell, die Unterstützung durch CORBA im Bereich Softwarearchitektur, sowie pragmatische Überlegungen zur Standardisierung und zum Markt. Wir setzen voraus, daß Sie die wichtigste Terminologie und technischen Ansätze der OMG-Spezifikationen bereits kennen, so daß wir uns auf eine Diskussion der eigentlichen Probleme von CORBA konzentrieren können.

1 Einführung

OMA, die Object-Management-Architektur der OMG, umfaßt die Bereiche Object Request Broker (ORB), allgemeine Dienste (COSS), und Common Facilities. Mit Hilfe des ORBs können Objekte anwendungs- und programmiersprachen übergreifend über in IDL definierte Schnittstellen kommunizieren. Die allgemeinen Dienste spezifizieren Objekteigenschaften und Dienste, die es zusammen ermöglichen objektübergreifende Funktionalität standardmäßig zur Verfügung zu stellen. Mit den Common Facilities hofft man bestimme horizontale oder vertikale Anwendungsbereiche standardisieren.

Hinter diesen verschiedenen Spezifikationen steckt die Vorstellung von verteilten Objektsystemen als eine Menge miteinander kommunizierender Objekte. Dies ist ein sehr verschwommenes Bild der Architektur verteilter Systeme, das keine Vision für bestimmte Anwendungssysteme, sondern vielmehr den kleinsten gemeinsamen Nenner aller möglichen verteilten Objektsysteme repräsentiert. Will man einen breiten Industriekonsens erreichen, muß dies so sein. Gleichzeitig aber besteht die Gefahr CORBA auf eine Art objektorientierten RPC (Remote Procedure Call) zu reduzieren.

In der Evaluation von CORBA und der dazugehörigen Dienste muß man aber zuerst daran denken, wie gut ein solcher Standard und die es vertretenden Produkte die eigenen konkrete Probleme löst. Dies kann nicht auf Ebene eines objektorientierten RPCs geschehen, da diese Abstraktionsebene die eigentlichen Probleme ignoriert. Die zentrale technische Fragestellung

Erschienen in *OBJEKTspektrum*. 6/97 (November/Dezember 1997). Seite 36-39.

muß unseres Erachtens auf einer Architekturebene ansetzen: Wie gut unterstützt CORBA die Art von Systemen, die wir bauen müssen? Zum Beispiel: In welchem Ausmaß hilft uns CORBA bei der Entwicklung von 2- oder mehrschichtigen Client/Server Architekturen? Wie hilft uns CORBA bei der Entwicklung einer Datenbus-Architektur (publish and subscribe) für kontinuierlich einlaufende Börsendaten?

Aus dieser Architekturperspektive heraus läßt sich dann evaluieren, wie gut eine bestimmte Middleware, zum Beispiel CORBA, für die eigenen Zwecke eingesetzt werden kann und wie groß die konzeptuelle und implementatorische Lücke zwischen der Middleware und der geplanten verteilten Objektarchitektur ist.

In diesem Artikel betrachten wir die Tauglichkeit von CORBA hinsichtlich der Entwicklung von Anwendungssystemen. Wir betrachten die OMA (Objekt-Management-Architektur) und was ihr Einsatz für Konsequenzen für darauf aufbauende Softwarearchitekturen hat. Daneben diskutieren wir eine Menge von pragmatischen Aspekten, wie die jedem Standardisierungsprozess inhärenten Probleme und Probleme mit aktuellen CORBA Implementierungen.

2 Objektmetamodell

Das OMA Objektmetamodell beschreibt was ein Objekt ist, was Operationsaufrufe sind, etc. Wir beziehen uns hierbei auf die CORBA-Spezifikation und OMA-Diskussion in [1-4].

Aus den Spezifikationen ist ersichtlich, daß die allen Objekten gemeinsame Standardfunktionalität minimal ist. Es werden lediglich die einfachsten Aspekte von Objekten definiert und keine weiteren Garantien über grundsätzliche Funktionalität gemacht. Denkbare Grundfunktionalität wird als Eigenschaft (capability) im Rahmen eines Dienstes definiert, so daß eine Anwendungsklasse zuerst von der die Eigenschaft repräsentierenden Schnittstelle erben muß. Als Konsequenz ergibt sich, daß Anwendungsentwickler Kombinationen von Eigenschaften und die daraus folgende Verwendbarkeit im Rahmen eines Dienstes für ihre Anwendungsklassen selbst definieren und implementieren müssen.

Die Situation gleicht den Standardisierungsbemühungen des ANSI-C++-Kommittees. Breiter Konsens ist nur auf kleinstem gemeinsamen Nenner zu erreichen, der zwar die Breite des Feldes abdeckt, aber die Lücke zur eigentlichen Anwendungsentwicklung größer werden läßt.

Neben fehlender Standardfunktionalität gibt es zudem keine Schnittstelle mit generischer Funktionalität, welche ein Objekt an unvorhergesehene Kontexte und Anwendungsfälle anpassen hilft. So gibt es zum Beispiel kein allgemeines Zugriffsprotokoll für den Objektzustand, mit dessen Hilfe sich Debugger, Serialisierung oder Persistenz generisch implementieren liessen. Auch gibt es kein generisches Protokoll zur Konfiguration und Erweiterbarkeit des Kontrollflusses bei Operationsaufrufen, wie man es zum Beispiel beim dynamischen Ein- und Ausschalten von Logging oder Security gern hätte. Produkte wie IONA’s Orbix bieten zwar Lösungen wie Smart-Proxies oder Filterobjekte an, diese sind aber leider proprietär und zumeist nicht sonderlich ausgereift.

Das Fehlen generischer Funktionalität macht die Anpassung und Erweiterung existierender Implementierungen für den Anwender unnötig schwer bzw. unmöglich, weil jede Einführung neuer Funktionalität notwendig eine Neubearbeitung der existierenden Implementierung verlangt.

Schließlich scheint uns die Evolution von Objektsystemen nicht berücksichtigt worden zu sein. Die Kopplung von Objekten ist starr: die Schnittstellen, die Operationsnamen und Parameterfolgen, und schlußendlich die Aufrufe müssen exakt zueinander passen. Zwar kann man dies grundsätzlich mit Hilfe der dynamischen Aufrufschnittstelle von Objekten (DII,

Dynamic Invocation Interface) abfangen; dies transparent abzufangen, wäre aber Aufgabe des ORBs, so daß man einen relevanten Teil der für Evolution wichtigen ORB-Funktionalität selbst implementieren muß. Da es zudem, wie bereits angemerkt, keine generischen Zugriffsmöglichkeiten gibt, muß man bei jedem Evolutionsschritt einer Schnittstelle die Implementierungen direkt anfassen.

All diese Aspekte zusammen führen dazu, daß CORBA für die Evolution großer verteilter Systeme nicht vorbereit ist ist, sofern man nicht die entsprechende Infrastruktur rundherum baut. Dies ist insbesondere dann kritisch, wenn man über Architekturen für Systeme nachdenkt, die 7 Tage in der Woche während 24 Stunden laufen müssen.

3 Softwarearchitektur

Die Architekturkritik setzt die Kritik des Objektmetamodells auf Architekturebene fort.

Will man für die firmeninterne Anwendungsentwicklung so etwas wie ein 3-schichtiges Client/Server Framework definieren, so hilft einem CORBA hierbei kaum. Man bekommt die wichtigsten Grundelemente in die Hand gedrückt, aber von einer Menge Legobausteine bis zu einem Haus ist es ein weiter Schritt. Es ist schwer abzuschätzen, wie groß die Lücke tatsächlich ist, da dies sehr umgebungsspezifisch ist. Es ist aber klar, daß es von einer Menge an Diensten und Facilities zu einem Framework für Client/Server-Architekturen ein nicht trivialer Schritt ist. Somit bringt CORBA einen zwar auf den Weg, aber die Strecke muß man selbst zurücklegen. Dies bezieht sich leider nicht nur auf die Anwendungssemantik sondern auch auf die Infrastruktur, das heißt die spezifische Kombination der bereits existierenden Dienste.

Hierbei können auch die Facilities nicht helfen. Sofern überhaupt spezifiziert und implementiert, decken sie jeweils nur einen bestimmten Systemaspekt ab. Die Kombination und Integration zum Abdecken einer bestimmten Anwendungskategorie bleibt weiterhin dem Anwender überlassen.

Die Objekt-Management-Architektur ist eine *Objekt*-Architektur, die Konzepte wie Objekt, Schnittstelle und Vererbung als Modellierungsmittel verwendet. Neben Objektarchitekturen haben sich seit einiger Zeit Komponentenarchitekturen etabliert, welche zwar Objektarchitekturen ähnlich, aber für die Erstellung von Softwarearchitekturen besser geeignet sind. Objekte waren ursprünglich für die feingranulare Modellierung und Programmierung gedacht, während Komponenten für die Architekturebene gedacht sind. Ein wohldefiniertes Komponentenmetamodell macht es einfacher, Architekturen mit Hilfe von Komponenten zusammenzustecken, als dies ein Objektmetamodell kann. Das Problem ist allerdings erkannt, und so haben sich IBM, Sun, Oracle and Netscape zusammengefunden, um ein Java Beans kompatibles Komponentemodell für CORBA zu entwerfen [5].

4 Pragmatische Überlegungen

Jeder großtechnische konsensorientierte Standardisierungsprozeß in den viele Parteien involviert sind führt dazu, daß man es vielen Seiten recht machen muß. Die daraus resultierenden Standards sind dann normalerweise relativ komplex, lassen aber einiges offen, da man sich über zu konkrete Lösungen schnell nicht mehr einigen kann. Daneben hat man das Problem, daß Standards sich weiterentwickeln aber ein sehr starker Druck herrscht, die älteren Teile des Standards nicht zu verändern. Dadurch entstehen immer komplexere, schwerer verständliche Standardgebäude, die aus einer Menge voneinander abhängigen Standars bestehen.

Diese Entwicklung, die schon bei CORBA's gemeinsamen Objektdiensten zu beobachten ist, führt dazu, daß es für den Anbieter einer Implementierung immer schwieriger und langwieriger wird neu standardisierte Funktionalität zu implementieren. Es entstehen damit zwangsweise Unterschiede zwischen dem, was man dem Anwender verspricht, aber noch nicht standardisiert ist, dem was bereits standardisiert ist und dem was man schlußendlich kaufen kann. Für den Anwender wird es damit immer schwieriger den Standard zu verstehen und festzustellen welches Produkt denn am besten für die eigenen Zwecke geeignet ist.

Der Vorteil einer Standardisierung ist, daß man nicht mehr nur von einem Hersteller abhängig ist. Damit handelt man sich aber auch das Problem ein, daß man sich mit einer Menge von Anbietern herumschlagen und viele implementierungsspezifische Details analysieren muß, um eine sinnvolle Auswahl treffen zu können.

Der Vorteil und gleichzeitig der Nachteil eines komplexen, verbreiteten Standards ist, daß man sich als Kunden einer Vielzahl von Implementierungen und Anbietern gegenüber sieht. Neben dem Aufwand der Evaluation der Lösungen und dem Risiko, daß man auf einen Anbieter setzt, der den Konkurrenzkampf nicht überlebt, hat man immer das Problem, daß man trotz Standard vom Hersteller abhängig wird. Dies liegt zum einen daran, daß kein Standard alle Details abdecken kann und zu seiner Implementierung gewisse Enscheidungen nötig sind, die dazu führen, daß man die Implementierungen nicht mehr einfach so autauschen kann. Beispiele bei CORBA-Implementierungen sind die Aktivierung von Implementierungen von Server-Objekten, die Herstellung der Verbindung zu einem Server-Objekt sowie die Struktur des Implementierungs-Repositories.

Daneben sind im Standard viele Aspekte nicht behandelt, die für die Anwender wichtig sind um den oben angesprochenen Graben zwischen den Implementierungen des Standards und ihren architekturellen Zielen zu schließen. Anbieter sind motiviert den Kunden dabei zu helfen indem sie proprietäre Erweiterungen vornehmen, wie z.B. die oben angesprochenen Smart-Proxies oder Filterobjekte von Orbix. Verwendet ein Kunde diese, dann ist er nachher stark an den Hersteller gebunden, solange er nicht bereit ist alle Erweiterungen selbst nochmals zu implementieren, was vielfach nicht möglich ist, da es sich um Erweiterungen der Implementierung des ORBs handlet. Wir haben folglich bei CORBA kompatiblen Lösungen dasselbe Problem, das man schon lange bei relationalen Datenbanken kennt.

Standardisierungen führen häufig zu ausgereiften technischen Vorschlägen, die so viele Aspekte behandeln, daß sie kaum mehr in ihrem ganzen Umfang zu beherrschen und zu implementieren sind. Für Softwareentwicklung im großen Stil macht dies häufig Sinn. Man sieht es aber in der Praxis immer wieder, daß sich wesentlich einfachere Lösungen als defakto Standard durchsetzen, da sie verfügbar und beherrschbar sind oder, daß leichtgewichtigere Ansätze, die nur den Teil der Probleme lösen, den man wirklich braucht, den Markt für auf den Standard basierende Lösungen massiv einschränken. TCP/IP, das sich ganz klar gegen die ISO/OSI Protokolle durchgesetzt hat, ist ein gutes Beispiel für eine leichtgewichtige Lösung, die zu einem defakto Standard geworden ist. COM und DCOM sind ein Beispiel für eine leichtgewichtige, wenngleich nicht unproblematische Lösung mit einem potentiell großen Marktanteil. Auf COM und DCOM basierende Lösungen sind konzeptionell CORBA basierten Lösungen unterlegen. Trotzdem ist es, gerade wenn man nur für Windows entwickelt, häufig sinnvoll, die Lösung von Microsoft zu verwenden. Dies werden wir in Zukunft noch in größerem Umfang erleben, falls Microsoft's Unterstützung für die Entwicklung verteilter Transaktionssysteme das hält, was sie verspricht.

5 Zusammenfassung

Nach dieser Diskussion können wir die im Titel gestellte Frage wieder aufgreifen: Löst CORBA wirklich alle unsere Probleme? Offenkundig nein. CORBA löst einen Teil unserer Probleme, aber eben nur einen kleinen Teil der technischen Probleme (von den organisatorischen Problemen ganz zu schweigen). Insbesondere fehlen Frameworks, wie sie sich in der Evolution objektorientierter Systeme im Laufe des letzten Jahrzehnts ergeben haben.

Der heutige Stand der OMA ist der Verwendung von C++ mit Standardbibliotheken vergleichbar: Es gibt lauter vereinzelte Lösungen, aber keine Architekturvision. Statt dessen wurde der kleinste gemeinsame Nenner an möglichen Anforderungen abgedeckt. CORBA befreit einen somit von vielen Low-level-details der verteilten objektorientierten Programmierung, es hilft einem aber kaum Architekturelle Probleme zu lösen, wie das z.B. objektorientierten Frameworks tun.

Dies heißt nicht, daß wir von CORBA abraten: zum heutigen Zeitpunkt gibt es nichts besseres, das zugleich auch herstellerunabhängig ist. CORBA ist zumindest ein solider Remote Method Invocation Mechanismus. Es ist aber unsinnig, zu erwarten, daß CORBA die grundsätzlichen architekturellen Probleme bei der Entwicklung großer verteilter Objektsysteme löst. Dies zu tun, hieße auf den üblichen Marketinghype hereinzufallen, der leider auch CORBA umgibt. Unserer Einschätzung nach ist der einzig vernünftige Umgang mit CORBA, dieses als Implementierungsvehikel zu benutzen und die architekturellen Hausaufgaben zu machen. Das heißt, sich über Anforderungen an zu entwickelnde Anwendungssysteme klar zu werden, Architekturvisionen in Form von Architekturstilen und -mustern zu entwickeln und die resultierende Architektur in wiederverwendbare Frameworks zu gießen. Für Großanwender bedeutet das große und langwierige Investitionen. Wenn man dazu nicht bereit ist, dann sollte man sich gründlich überlegen ob es Sinn macht in die verteilte Objekttechnologie einzusteigen.

Abschließend möchten wir unserem Kollegen Kai-Uwe Mätzel für die gemeinsamen Diskussionen danken.

Literaturverweise

[1] OMG. *The Common Object Request Broker: Architecture and Specification.* Framingham, MA: Object Management Group, 1997.

[2] OMG. *CORBAservices. Common Object Services Specification.* Framingham, MA: Object Management Group, 1997.

[3] OMG. *Common Facilities Architecture.* Framingham, MA: Object Management Group, 1997.

[4] OMG. *A Discussion of the Object Management Architecture.* Framingham, MA: Object Management Group, 1997.

[5] Netscape Communications. *http://home.netscape.com/newsref/pr/newsrelease422.html.* Netscape Communications, 1997.

Zu den Autoren

Dr. Walter Bischofberger ist “Chief Scientist” der Firma TakeFive (http://www.takefive.com). In dieser Funktion beschäftigt er sich hauptsächlich mit Methoden und Werkzeugen für die kooperative Entwicklung großer objektorientierter Systeme.

Dipl.-Inform. Dirk Riehle arbeitet am Ubilab (http://www.takefive.com), dem Informatik-Forschungslabor der UBS. Er ist Autor zahlreicher Fachartikel zu den Themen Objektorientierung und Entwurfsmuster. Außerdem ist er Übersetzer des Buchs *Design Patterns* (siehe [2]) sowie Autor des frisch bei Addison-Wesley Deutschland erschienenen Buchs *Entwurfsmuster für Softwarewerkzeuge*.

E-Mail: Dirk.Riehle@ubs.com or riehle@acm.org