CORBA

- **C**ommon **O**bject **R**equest **B**roker **A**rchitecture
    - erste Version 1991: CORBA 1.1
    - 1994: CORBA 2.0 (Interoperabilität: IIOP)
    - derzeit aktuell: CORBA 3.0 (aber: konforme Produkte "dauern")

eine Architektur, kein Produkt!

- OMG (Object Management Group)
    - herstellerübergreifendes Konsortium
    - gegründet 1989 von 11 Firmen
    - jetzt mehr als 700 Mitglieder (wer eigentlich nicht? Selbst Microsoft ist dabei); vgl. http://www.omg.org/cgi-bin/membersearch.pl
    - Ziel: Bereitstellung von Konzepten für die Entwicklung verteilter Anwendungen mit objektorientierten Modellen
    - genauer: Definition und Entwicklung einer Architektur für kooperierende objektorientierte Softwarebausteine in verteilten heterogenen Systemen (--> "Middleware")

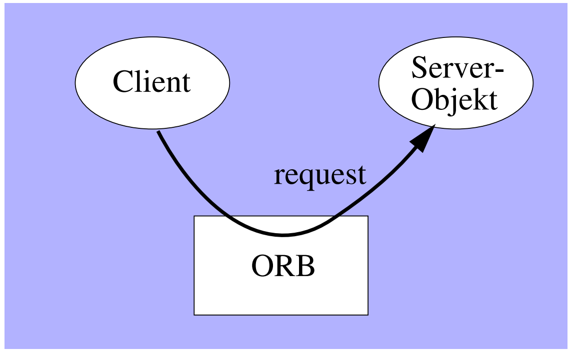
- CORBA beruht i.w. auf der Idee der Realisierung von Softwaresystemen aus interagierenden Komponenten
    + Services
    + Objektorientierung

Beachte: Objektorientierung selbst ist eigentlich ein "altes" Konzept

- Mitte der 60er-Jahre (Programmiersprache "Simula")
- damals bereits fast alle Aspekte der Objektorientierung (Klassenhierarchien, virtuelle Klassen, Polymorphismus...)

CORBA - Übersicht

- Objektmodell (mit Aufrufsemantik etc.)
- IDL mit entspr. Generatoren und Compilern
- Object Request Broker (ORB) als Vermittlungsinfrastruktur

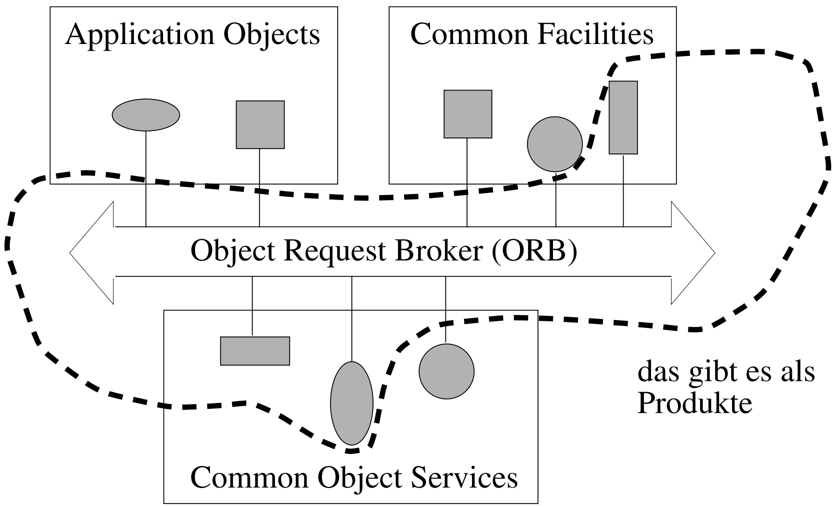


- Systemfunktionen in Form von Object Services
- Unterstützung von Anwendungen durch Common Facilities / Application Domains
- Konventionen bezüglich Schnittstellen, Protokollen etc.

- CORBA ist also eine Infrastruktur und unterstützt die Ausführung verteilter objektorientierter Systeme
- Die Entwurfs- und Spezifikationsphase solcher Systeme werden mit anderen Konzepten unterstützt, z.B. UML ("Unified Modeling Language"), mit der u.a. Diagrammnotationen standardisiert werden

Object Management Architecture

- "OMA" ist die Referenzarchitektur der OMG, die die wesentlichen Bestandteile einer Plattform für verteilte objektorientierte Applikationen definiert



- Application Objects: Objekte der eigentl. Anwendung
  - gehören damit nicht zur CORBA-Infrastruktur
- ORB: Vermittlung zwischen verschiedenen Objekten; Weiterleitung von Methodenaufrufen etc.
  --> Ortstransparenz, Kommunikation...
- Object Services: objektorienteirte Schnittstelle zu standardisierten wichtigen Diensten
- Common Facilities: allgemein nützliche Dienste
  - nicht notw. Teil aller CORBA-Implementierungen

Object Services (1)

- Basisdienste als systemweite Infrastruktur
  - Dienstleistungskomponenten besitzen eine objektorientierte Schnittstelle
- COSS (Common Object Services Specification)
  - Realisierung ist für CORBA-konforme Produkte verpflichtend
  - noch nicht alle Services sind derzeit vollständig spezifiziert

---

1) Ereignismeldung
   - Weiterleitung asynchroner Ereignisse an Ereigniskonsumenten
   - Einrichten von "event channels" mit Operationen wie push, pull...

2) Persistenz
   - Dauerhaftes Speichern von Objekten auf externen Medien

3) Naming
   - Erzeugung von Namensräumen
   - Abbildung von Namen auf Objektreferenzen
   - Lokalisierung von Objekten

4) Lifecycle
   - erzeugen, löschen, kopieren, verlagern... von Objekten

5) Concurrency
   - Semaphore, Locks,...

6) Externalization
   - Export von Objekten in "normale" Dateien

Object Services (2)

7) Transactions

- 2-Phasen-Commit etc.

8) Time

- Uhrensynchronisation etc.

9) Security

10) Licensing

- Management von Lizenzdiensten für Komponenten

11) Trading

- Matching von Services zu einer Service-Beschreibung eines Klienten

12) Replikation

- Sicherstellung der Konsistenz replizierter Objekte in einer verteilten Umgebung

- es gibt noch einige weitere Services...

Common Facilities

- Höherwertige Dienste für ein breites Spektrum von Anwendungsbereichen

- Bereitstellung allgemein interessanter Funktionalität
  - analog zu grossen Klassenbibliotheken

- Horizontal ”Common Facilities”
  - Basisfunktionalität, die für verschiedene Anwendungsbereiche von Nutzen ist
    - secure time
    - printing
    - mobile agents
    - internationalization (“...will enable developers to use an application in their own language using their own cultural conventions...will allow the developer to use a culture’s numeric and currency conventions...)
    - user interface
    - information management (z.B. Speicherung komplexer Objektstrukturen, Formatkonvertierungen,...)
    - systems management (z.B. Installation, Konfiguration... von Objekten)
    - task management (Workflow, lange Transaktionen...)

- Vertikale Common Facilities (“Domain Facilities”)
  - Basisfunktionalität für diverse Marktsegmente, z.B. Banken, Finanzdienste, Gesundheitswesen...
  - vgl. “application frameworks”, “business objects” etc.

OMA Component Definitions

Object Request Broker - commercially known as CORBA, the ORB is the communications heart of the standard. It provides an infrastructure allowing objects to converse, independent of the specific platforms and techniques used to implement the objects. Compliance with the Object Request Broker standard guarantees portability and interoperability of objects over a network of heterogeneous systems.

Object Services - these components standardize the life-cycle management of objects. Interfaces are provided to create objects, to control access to objects, to keep track of relocated objects, and to control the relationship between styles of objects (class management). Also provided are the generic environments in which single objects can perform their tasks. Object Services provide for application consistency and help to increase programmer productivity.

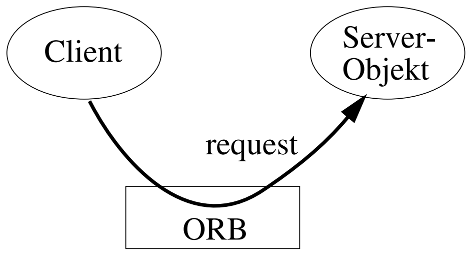
Common Facilities - Common Facilities provide a set of generic application functions that can be configured to the specific requirements of a particular configuration. These are facilities that sit closer to the user, such as printing, document management, database, and electronic mail facilities. Standardization leads to uniformity in generic operations and to better options for end users for configuring their working environments.

Domain Interfaces - Domain Interfaces represent vertical areas that provide functionality of direct interest to end-users in particular application domains. Domain interfaces may combine some common facilities and object services, but are designed to perform particular tasks for users within a certain vertical market or industry.

Quelle: http://www.omg.org/omaov.htm

Kommunikation zwischen Objekten

- Menge interagierender Objekte typw. in zwei Rollen
  - Client-Objekt (Aufrufer)
  - Server-Objekt



- Methodenaufruf durch requests unterschiedl. Semantik
  - synchron (insbes. bei Rückgabewerten; analog zu RPC)
  - “verzögert synchron” (Aufrufer wartet nicht auf das Ergebnis sondern holt es sich später ab)
  - “one way” (asynchron: Aufrufer wartet nicht)

- Beim Methodenaufruf muss angegeben werden
  - das Zielobjekt
  - die Parameter
  - ggf. Angaben über Exceptions und Rückgabewerte

Interface Description Language (IDL)

- Sprache zur Definition von Schnittstellen (Parameter, Attribute, Superklasse bzgl. Vererbung, Exceptions...)

- lexikalisch an C++ angelehnt

- Bsp: oneway void move (in long x, in long y)

Eingabe-, Rückgabe- oder Eingabe- und Rückgabeparameter

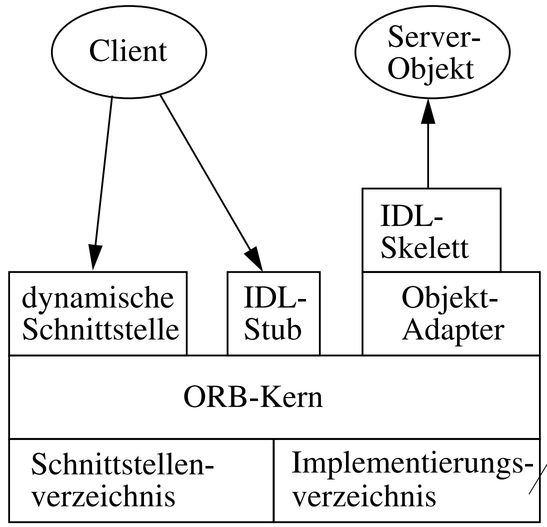
- grundsätzlicher Aufbau:

  [oneway] <op_type_spec> <identifier> (param$_1$,....,param$_L$)
  [raises(except$_1$,...,except$_N$)] [context(name$_1$,...,name$_M$)]

erlaubt Angabe möglicher Exceptions, die bei Durchführung der Operation auftreten können

operationsspezifische Informationen, die dem Operationsaufruf mitgegeben werden können

Kommunikation (2)



enthält “Umgebungsdaten” eines Server-Objektes (z.B. Name der Datei mit dem Programmcode, Speicherbedarf...)

- ORB bietet einem Client zwei Arten von Schnittstellen für den Methodenaufruf an
  - statische Schnittstelle (Erzeugung von Stubs aus der IDL-Beschreibung analog zu RPCs)
  - dynamsiche Schnittstelle (Client kann zur Laufzeit das Schnittstellenverzeichnis abfragen und einen geeigneten Methodenaufruf zusammenstellen)

- Objektadapter: Steuert anwendungsunabhängige Funktionen des Server-Objekts
  - z.B. Aktivierung des Server-Objektes bei Eintreffen eines requests, Authentifizierung von requests, Zuordung von Objektreferenzen zu Objektinstanzen etc.
  - zuständig ausserdem für Registrierung von Services
  - es gibt einen standardisierten Basic Object Adapter (BOA), der für viele Anwendungen ausreichende Grundfunktionalität bereitstellt

Server-Objekte

- Bereitsstellung von Services teilweise analog zu Prozeduren, die im Rahmen von RPCs verwendet werden

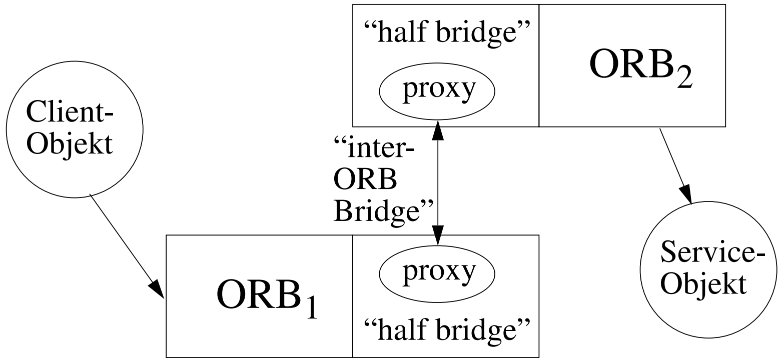
- Objekte können ein aus der IDL-Spezifikation generiertes Objekt-Skelett nutzen

- Objekt muss sich beim lokalen Objekt-Adapter anmelden und dabei eine “server policy” angeben
  - *Shared Server:* kann mit mehreren anderen aktiven Server-Objekten von einem einzigen Prozess verwaltet werden
  - *Unshared Server*
  - *Server per Method:* Start eines eigenen Prozesses bei Methodenaufruf
  - *Persistent Server:* ein Shared Server, der von CORBA initial bereits gestartet wurde

- Objekt muss sich ferner beim Implementierungsverzeichnis anmelden
  - damit es bei einem Methodenaufruf gefunden wird

CORBA 2.0

- Insbesondere Interoperabilität von ORB’s verschiedener Herstellerimplementierungen gefordert



- ORB Bridge: Formatkonvertierung und Weiterleitung eines requests etc. an einen anderen ORB
  - Schnittstellen und Konventionen für solche Bridges sind im CORBA-Standard festgelegt
  - “Request Level Bridge” besteht aus zwei Teilen mit einer CORBA-Schnittstelle, welche bei Bedarf Proxy-Objekte erzeugen, die die Aufrufkonvertierung vornehmen

- Inter-ORB-Kommunikation mittels bestimmter Protokolle
  - GIOP (General Inter-ORB Protocol); es muss mindestens eine auf TCP/IP-aufbauende Realisierung geben (diese ist im “Internet Inter-ORB Protocol” (IIOP) festgelegt)
  - ESIOP (Environment-Specific Inter-ORB Protocol) ist optional; es existieren Realisierungen aufbauend auf DCE

CORBA - weitere Entwicklungen (insbesondere CORBA 3.0)

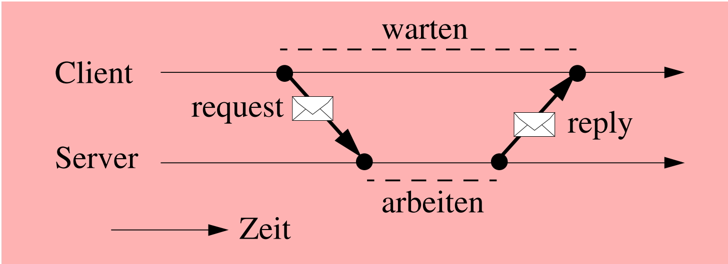
- Messaging Service
  - Objekte können sich asynchrone Nachrichten schicken (store & forward)
- Objects by Value
  - Kopie eines Objektes wird übergeben, nicht nur Referenz
- Persistente Objekte
  - “Abspeichern” von Objekten
- Komponenten-Modell
- Java-Unterstützung
  - Generieren von IDL aus Java bzw. Java-RMI (“reverse mapping”)
- Firewall-Unterstützung
  - klassische Firewalltechnik (z.B. Services identifiziert mit Portnummern) versagt teilweise; Callbacks erscheinen als Aufruf von aussen...
- Minimum CORBA
  - Unterstützung von embedded systems (i.w. Weglassen von Dynamik)
- Realzeit-Unterstützung
- Fault Tolerant CORBA
  - durch redundante Einheiten

hierzu gibt es z.Z. wenig mehr als gutgemeinte Absichten (bestenfalls Experimentelles)

- CORBA-URLs des Interoperable Naming Service
  - z.B.: iioploc://meinefirma.com:683/NamingService
  - löst damit das Bootstrapping-Problem des Namensdienstes

Neu: Messaging Service

- Asynchrones Kommunikationsparadigma
- Motivation:
  - mobile Geräte (PDA, Laptop,...) sind oft nicht online
  - bei sehr grossen verteilten Systemen sind unausweichlich stets einige Geräte bzw. Services nicht erreichbar (Netzprobleme etc.)
- CORBA basierte bisher auf einer engen (“synchronen”) Kopplung von Client und Server



- Asynchron:
  - Entkopplung von Sender / Empfänger
  - Sender blockiert nicht solange bis Nachricht angekommen ist
  - Nachricht kann von Hilfsinstanzen (“Router”) zwischengespeichert werden, bis Empfänger (bzw. bei Antwort: Sender) erreichbar ist
  - Antwort kann ggf. von anderem Klient als ursprünglichem Sender entgegengenommen werde

Asynchronous Method Invocation

- Bisherige Möglichkeiten eines Methodenaufrufs in CORBA:
  - "synchron" (insbes. bei Rückgabewerten; analog zu RPC)
  - "verzögert synchron" (Aufrufer wartet nicht auf das Ergebnis, sondern holt es sich später ab)
  - "one way" (Aufrufer wartet nicht) mit "best effort"-Semantik ("fire and forget")

nur bei Dynamic Invocation Interface (DII)

gedacht war an UDP-Implementierung; Semantik (z.B. Fehlermeldung bei Misslingen?) implementierungsabhängig

---

- Neu: Asynchronous Method Invocation (AMI)
  - geht auch bei statisch generierten Stubs
  - bisher umständlich mit mehreren Threads simuliert (bzw. DII)

- Zwei Aufruftechniken bei AMI: (1) Callback
  - Client gibt dem Aufruf eine Objektreferenz für die Antwort mit
  - Callback-Objekt kann sich im Prinzip irgendwo befinden
  - Kommunikations-Exceptions werden im Callback-Objekt ausgelöst

- (2) Polling
  - Client erhält sofort ein Objekt zurück, das er für Polling oder zum Warten auf Antwort nutzen kann

Time-independent Invocation (TII)

- Teil des Messaging Services: Aufruf von Objekten, die nicht aktiv sind oder zeitweise nicht erreichbar sind

- Aufruf-Nachrichten werden von zwischengeschalteten "Router Agents" verwaltet
  - Store and Forward-Prinzip
  - Router Agent beim Client ermöglicht disconnected operations
  - Router Agent beim Server kann dessen Eingangsqueue verwalten

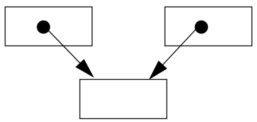
- "Interoperable Routing Protocol" sorgt dafür, dass Router Agents verschiedener Hersteller interagieren

- Quality of Service (QoS) steuerbar (als "Policy")
  - z.B. max. Round Trip-Zeit: dadurch müssen Router Agents Nachrichten nicht beliebig lange aufbewahren
  - oder z.B. Aufrufreihenfolge: Soll Router seine gespeicherten Aufträge zeitlich geordnet oder nach Prioritäten oder... ausliefern?

- Beachte: QoS ist ein gewichtiger Name für ein einfaches Prinzip ohne feste Garantien

Objects by Value

- Bisher war nur *Referenzübergabe* möglich
  - um es Objekten zu gestatten, Methoden anderer Objekte aufzurufen, konnten bisher Objekt<u>referenzen</u> als Parameter übergeben werden
  - Objekt selbst bleibt aber “am Platz”, Aufruf wird also immer als Fernaufruf über das Netz geschickt
  - ferner kommt es zum gelegentlich unerwünschten Aliasing-Effekt: unbedachte Rückwirkungen auf das “Originalobjekt”
- Bei *Wertübergabe* wird das Objekt serialisiert und im Adressraum des Empfängers eine Kopie angelegt
  - Marshalling des Objektzustandes (d.h. der *Daten*)
  - auf Empfängerseite existiert eine Factory, die das Objekt als Kopie (mit eigener Identität) erzeugt (wie macht die Factory das?)
  - was geschieht bei Alias-Zeigern bzw. Zyklen bei der Serialisierung komplexer Strukturen?



- Bei heterogenen Umgebungen: Wie transportiert man das *Verhalten* des Objektes zum Empfänger?
  - es handelt sich um ausführbaren Code (für welche Maschine?)
  - kann von verschiedenen Sprachen (C, Java,...) erzeugt worden sein
  - einfach bei Java auf beiden Seiten: Bytecode ist unabhängig vom Maschinentyp durch die JVM in gleicher Weise interpretierbar
  - ansonsten muss die Factory beim Empfänger (aber wo kommt die her?) sich den Code besorgen (aus lokaler Bibliothek, übers Netz...)
- Leider können jedoch keine normalen CORBA-Objekte per Value übergeben werden, nur sogen. “valuetypes”
  - neues Konstrukt der IDL (--> für Anwender dadurch kompliziert)
  - Wertübergabe echter Objekte wäre zu schwierig

Real-Time CORBA

- Ziel: Vorhersagbares Ende-zu-Ende-Verhalten (insbesondere beim entfernten Methodenaufruf)
  - sowohl für “Hard Real-Time”
  - als auch für “Soft Real-Time” (mit nur statistischen Aussagen)
- Basiert auf einem zugrundeliegendem Realzeit-BS
- Einige Mechanismen:
  - Prioritäten
  - Timeouts für Aufrufe
  - Multithreading (mit geeignetem Scheduling), Threadpools
  - private (statt gemultiplexte) Verbindungen
  - austauschbare Kommunikationsprotokolle
- Anwendungsgebiete z.B.:
  - verteilter Telefonswitch
  - Prozessautomatisierung
  - Avionics (?)
  - ...

CORBA Components: Komponenten

- Was sind Komponenten?
  - eigenständige, wiederverwendbare “Softwarebausteine”
  - wohldefinierte Schnittstelle
  - Implementierung vollständig gekapselt (unsichtbar von aussen)

- Warum Komponenten?
  - schnelle Anwendungsentwicklung
  - einfache und damit hohe Wiederverwendbarkeit
  - vorgefertigte Standard-Komponenten
  - grafische Tools zum “Zusammenstecken” der Komponenten

- Elemente eines Komponenten-Modells
  - Definition (Aussehen einer Komponente)
  - Interaktion (Austausch von Nachrichten zwischen Komponenten)
  - Komposition (Zusammenbau mehrerer Komponenten)
  - Konfiguration (Anpassen an spezielle Bedürfnisse)
  - Introspektion (Herausfinden der Schnittstellen zur Laufzeit)
  - Persistenz (Speichern/Wiederherstellen von Komponenten)
  - Verpackung (Auslieferung von Komponenten)

CORBA Components: Allgemeines

- Serverseitiges Komponenten-Modell, Ähnlichkeiten zu EJB

Enterprise Java Beans, Java-basiertes Komponentenmodell von Sun

- Anbindung von EJB an CORBA Components spezifiziert

- Tiefgreifende Änderungen und Erweiterungen am CORBA-Objektmodell, dem ORB und der IDL notwendig

- Verschiedene Kategorien von Komponenten (Service, Session, Process, Entity), die sich in Lebensdauer, Vorhandensein von Zustand und Identität unterscheiden

CORBA Components (1)

- Definition

- spezielles CORBA-Objekt
- funktionale Schnittstelle (mehrere Interfaces, Navigation)
- Attribute für die Konfiguration
- Abhängigkeiten von anderen Komponenten
- Komponenten “leben” in Containern, die bestimmte Basisdienste bereitstellen

Sicherheit, Persistenz, Transaktionen, ...

- Interaktion

- Event-Modell (*Publish-Subscribe*), vereinfachte Form des Notification-Service
- Events sind strukturiert und müssen in IDL deklariert werden
- Events können Transaktions-Charakter haben

- Komposition

- Aggregation (Vererbung würde Zugriff auf die Implementation der Komponente erfordern)
- Aggregation bedeutet hier Verbinden der Komponenten durch Event-Kanäle und Skripte
- Container enthält die “aggregierten” Komponenten und ist selbst wieder eine Komponente mit mehreren Interfaces

- Konfiguration

- Attribute der Komponente mit Tools auf bestimmte Werte setzen
- vordefinierte Konfigurationen (Attributwerte), die beim Erzeugen einer Komponente durch eine Factory automatisch gesetzt werden

CORBA Components (2)

- Introspektion

- Herausfinden der Schnittstellen einer Komponente zur Laufzeit (beispielsweise der Attribute)
- Erweiterung des Interface Repository

- Persistenz

- Sichern / Wiederherstellen einer Komponente bzw. deren Zustand
- Überarbeitete Version des POS, Spezifikation liegt noch nicht vor

Persistent Object Service

- Verpackung

- Auslieferung einer Komponente in Form eines speziellen Archivs (CAR Dateien)

CORBA ARchive

- Enthält Implementation der Komponente, der Factory und des Attribut-Editors für mehrere Plattformen, zusätzliche Ressourcen (Bilder, Fehlermeldungen), Dokumentationen und Installations-Skripte
- Beschreibung des Archivinhalts (bzw. der Komponente) durch XML-basierte Datei (CSD Datei)

CORBA Software Descriptor

- Aggregation mehrerer Komponenten ebenfalls durch XML-basierte Datei beschrieben (CAD Datei)

CORBA Assembly Descriptor

CORBA und DCOM

- DCOM: verteilte Version von COM (Microsoft)

Component Object Model

- Weitgehend gleiche Zielsetzung
  - Integration, Interoperabilität, Komponenten-Software, Objektmodell
  - transparenter Zugriff auf entfernte Objekte
  - statische und dynamische Aufrufe
  - IDL heisst hier MIDL (‘M’ für Microsoft) und Repository “Registry”
  - Aufgaben von ORB und Object Adapter werden von einem “Service Control Manager” (SCM) wahrgenommen
- DCOM ist aber proprietär statt offen
  - entstanden aus einem “Dokumentenkomponentenmodell”
- Einige Unterschiede
  - Schnittstellen-Aggregation statt -Vererbung
  - mehrere Schnittstellen pro Objekt (aber: Komponentenmodell von CORBA 3.0)
  - DCOM hat Garbage Collection mit Referenzzähler
  - DCOM definiert ein “Binärformat”
  - ...
- Integration von CORBA und DCOM?
  - möglich bis zu einem gewissen Grad: Brücke zwischen den beiden Welten (z.B. OLE-Broker von ORBIX)

CORBA und Java

- Java ist weit verbreitet (als “Internetprogrammier-sprache”) und in gewissem Sinn eine “Konkurrenz”
  - allerdings hegemonistisch im Sinne von “überall Java / JVM”
- Ziel: Interoperabilität durch Zusammenführung von Java RMI und CORBA IIOP

> - Remote Method Invocation: Entfernter Methodenaufruf mit Transport (und dabei Serialisierung) auch komplexer Objekte
> - auch Code (Bytecode) wird übertragen und remote ausgeführt
> - damit kann z.B. ein Proxy eines Servers (z.B. ein Druckertreiber) jeweils aktuell zur Laufzeit vom Server zum Client geschickt werden
> - der Proxy kann dann (unsichtbar für den Client) z.B. mit dem Server ein privates Protokoll realisieren oder sonstige Dinge lokal tun

- IDL automatisch aus Java-Programmen generieren
  - “reverse mapping” (weiterhin existiert natürlich IDL --> Java mapping)
  - Java-Programmierer brauchen kein IDL zu nutzen und zu lernen
  - aus Java heraus sind so Objekte anderer Sprachen ansprechbar (bzw. z.B. Java-Server, der von Clients anderer Sprachen genutzt werden kann)
- Aufbrechen des “single language Paradigmas” von Java
  - Java-Objekte werden von CORBA aus zugreifbar
- Kleinere Einschränkungen jedoch notwendig
  - Java-RMI und CORBA-IDL sind nicht deckungsgleich

CORBA und Open Source

- Open-Source-Bewegung
  - Quellcode steht allen zur Verfügung
  - Weitergabe von Änderungen, Erweiterungen,... an die Community

- Effekt von Open-Source
  - Synergien von sehr vielen Entwicklern und Nutzern
  - Erwartung: Schnellere Entwicklung und höhere Qualität
  - Konkurrenzmodell zu kommerzieller Softwareerstellung

- Beispiele für Open-Source
  - Internet: DNS / BIND, TCP/IP, sendmail, usenet news,...
  - GNU-Tools (emacs, gcc,...)
  - Tex, Linux, Apache-Web-Server, Perl
  - Mutanten: StarOffice, Netscape, Jini,...

- MICO

Mico Is COrba

- modularer Aufbau
- Mikrokern-basiert

  - Open-Source CORBA-Implementierung
  - Motivation: Grundlage für Projekte in Lehre und Forschung
  - Beginn Dez. 96 (Kay Römer, Arno Puder)
  - Vers. 0.1 (4/97: 26k LOC); 2.0.0 (1/98, 130k), 2.2.7 (6/99, >300k)
  - Mailing-Liste mit derzeit > 1500 Teilnehmer
  - CORBA 2.1-Zertifizierung durch OpenGroup
  - ständige Weiterentwicklung bzgl. neuester CORBA-Versionen
  - Nutzergruppen: Lehre, Forschung, Industrie

CORBA-Produkte und Implementierungsstand

- Es gibt eine Reihe von Implementierungen von CORBA
  - d.h. ORB mit den wichtigsten Services, tools, Bibliotheken (Herbst '99):

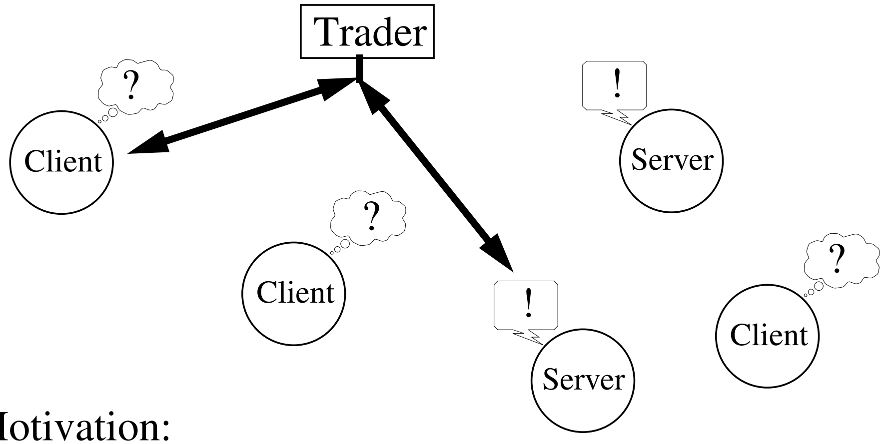
OpenSource:	*Kommerzielle Produkte:*
- MICO	- IONA: Orbix
- JacORB	- Inprise: Visibroker
- TAO	- BEA: ObjectBroker
- omniORB	- OOC: ORBacus
- ILU	- Expersoft: PowerBroker

- Implementierungsstand ist unterschiedlich:
  - POA (CORBA 2.2): in Mico, nur Beta-Versionen in kommerziellen ORBs
  - ObV (CORBA 2.3): in Mico, noch nicht in kommerziellen ORBs
  - Firewall: erst Prototypimplementierungen
  - Messaging Service: noch nicht implementiert
  - Real Time, Fault Tolerance: nur experimentelle Systeme
  - Komponentenmodell: noch nicht endgültig spezifiziert
  - Minimum CORBA: nur experimentelle Systeme

- Wie geht es danach weiter mit CORBA? Immer mehr Ergänzungen, neue Services...?
  - CORBA kann nicht alles selbst machen (z.B. Fehlertoleranz, Realzeit, Mobilität, embedded systems, electronic commerce...)
  - Brücken (und Kompromisse) zu anderen Systemen und Techniken (z.B. XML, DNA, Jini, DCOM, Internet-Protokolle...)

Trading

- Vermittlungsdienst für Client und Server (“matching”)
  - aktuelles Forschungsthema
  - noch keine umfassenden Lösungen!



- Motivation:
  - steigende Zahl von Diensten führt zu einem unübersichtlichen Angebot
  - ideal: offener Dienstleistungsmarkt, in dem sich Dienstanbieter und Dienstnutzer nicht a priori zu kennen brauchen
- Vorbilder: Makler, Headhunter, gelbe Seiten...
- Trading ist typischerweise selbst ein verteilter Dienst (Kooperation mehrerer Trader-Instanzen etc.)
  - noch keine Standards etc. für “Trader interworking”
- Schwierig, sobald Semantik wesentlich betroffen ist
  - wie beschreibt man Eigenschaften?
  - Attributmengen? Ausführbare Typbeschreibungen? Vor-/Nachbedingungen?
  - KI-Methoden, um (semantische) Eigenschaften abzuleiten, zu kategorisieren, Ähnlichkeiten zu ermitteln...? (Wissenbasierte Dienstvermittlung)
  - Wissensrepräsentationstechnik? (z.B. Konzeptgraphen etc.)