Software Architectures for Enterprises SA4E

Peter Sturm
Universität Trier
Winter 2024


Monolith

(c) UKRI Science and Technology Facilities Council, available from https://www.chilton-computing.org.uk/


• Programmiersprache
• Imperativ, Prozedural, Objektorientiert, ...
• C, C++, C#, Java, Python, ...
• Bibliotheken
• Multi-Threading

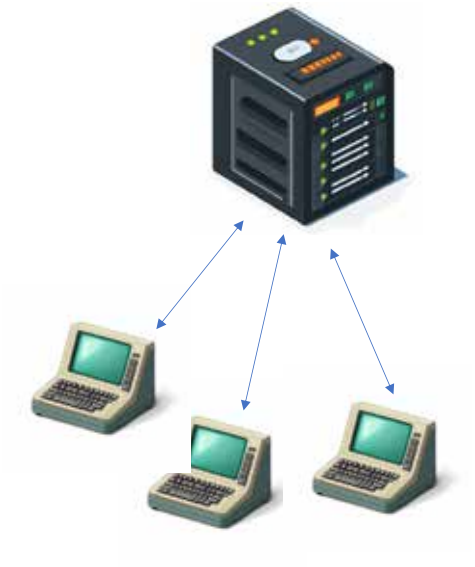

• Programmiersprache
• Imperativ, Prozedural, Objektorientiert, ...
• C, C++, C#, Java, Python, ...
• Bibliotheken
• Multi-Threading

Optionen


MAGNUS 12
bosdem

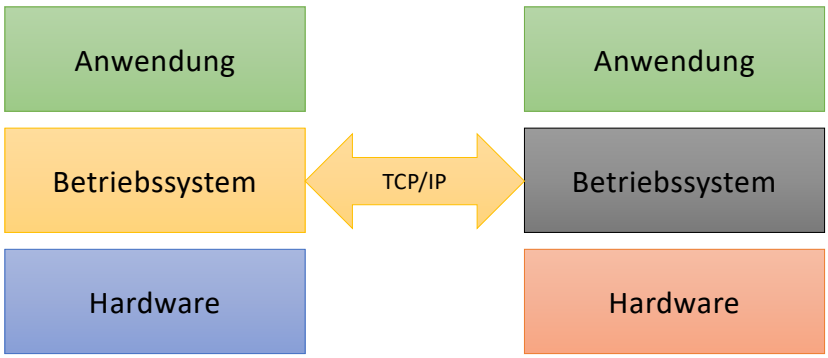
Optionen

- Dateien
  - Verschiedene Formate
- Terminal-Server
- Transaktionsmonitore
- Proprietäre Verbindungen
  - Seriell, Modems, Dial-up Lines
  - X.25 Packet-Switched Networks
  - IBM SNA
  - NetBIOS, IPX/SPX
  - ...



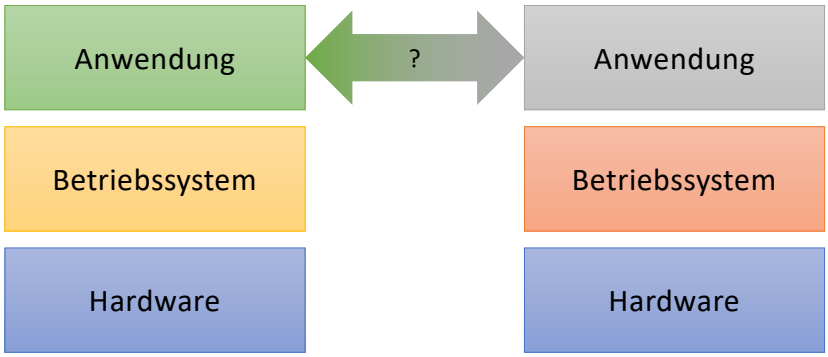
12

Oh? Da gibt es noch jemanden?

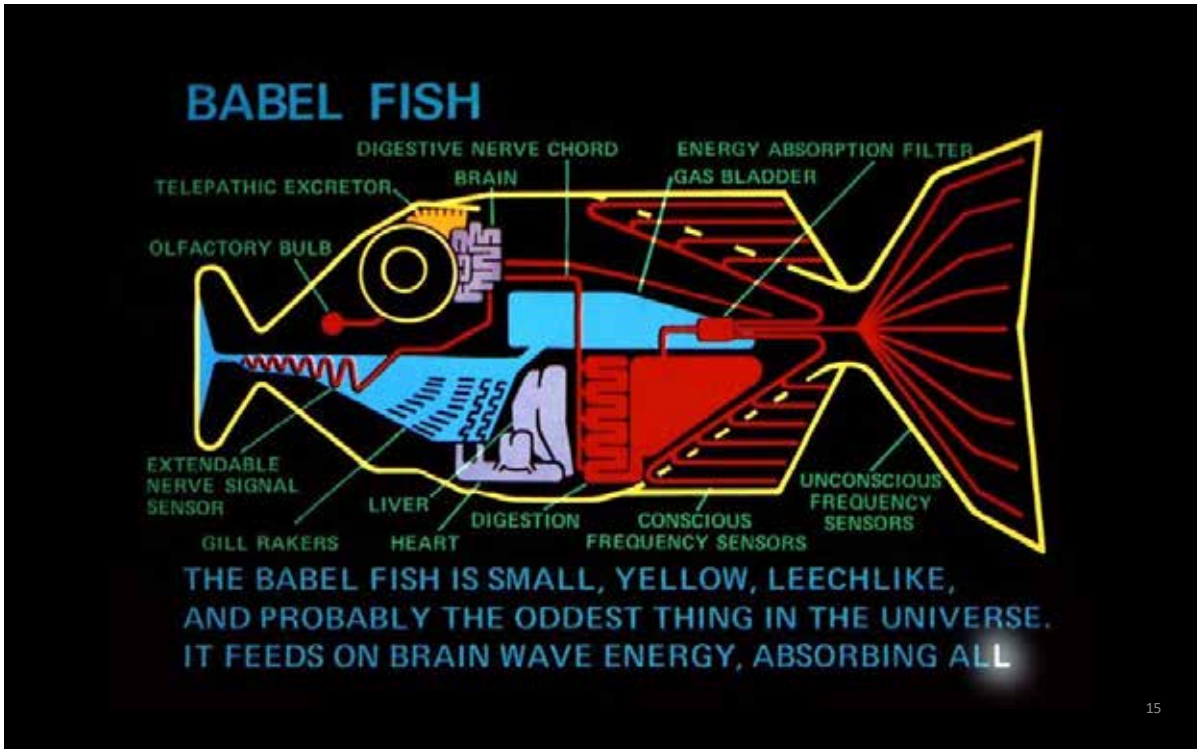


13

… und der ist anders als ich?



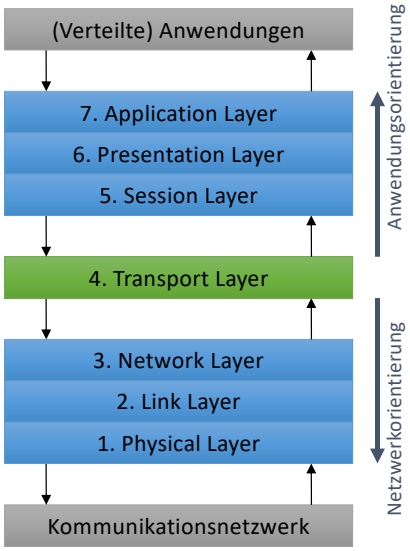
14



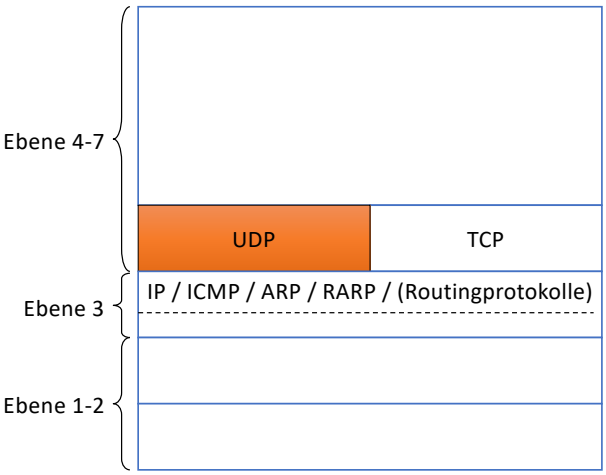
15

ISO-Referenzmodell

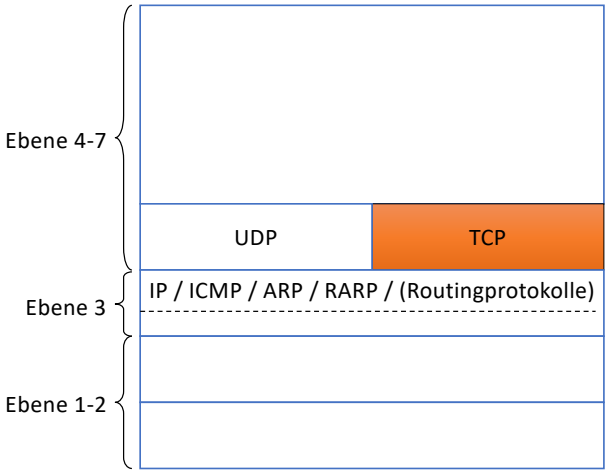
- OSI-Modell = Reference Model for Open Systems Interconnection
- Referenzmodell !
- Realisierung
  - Ebene 1-3: Hardware (Firmware)
    - Netzwerk-Controller
  - Ebene 4-7: Software



UDP



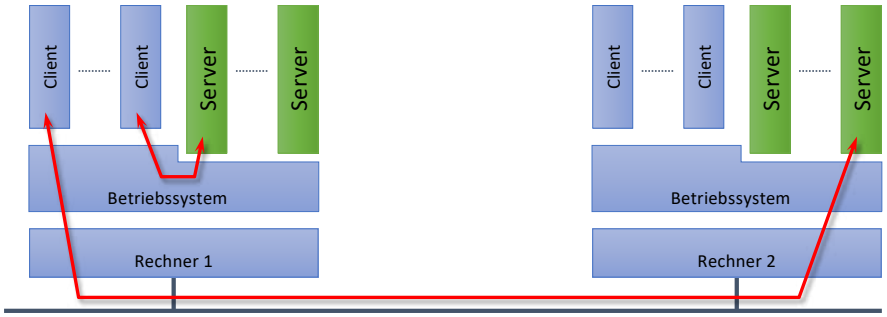
TCP


Ebene 4-7
UDP
TCP
Ebene 3
IP / ICMP / ARP / RARP / (Routingprotokolle)
Ebene 1-2

RPC

19

Client/Server-Architektur

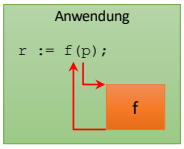


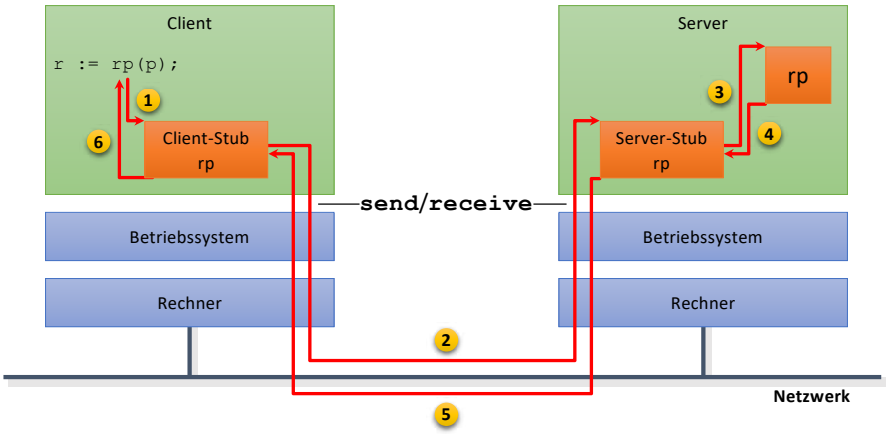
- Lokale Server
  - Dateisystem
  - E/A
  - Prozeßverwaltung
  - Paging, ...
- Entfernte Server
  - Netzwerkdateisystem
  - WWW
  - SSH, ...

20

RPC-Architektur

Lokaler Prozeduraufruf:





21

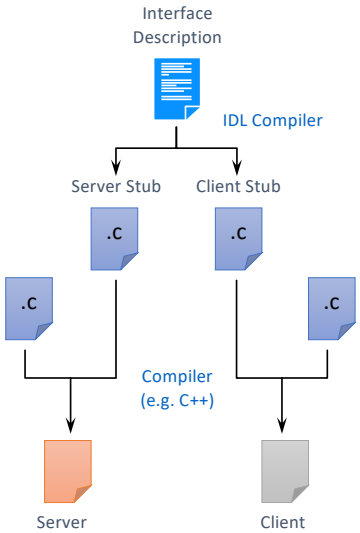
Marshalling

- Encoding of basic data types may differ
  - Little endian vs. big endian
  - …
- Define a generic network standard
  - Sender converts from local to network
  - Receiver converts from network to local
- Examples
  - External data representation (xdr)
  - ASN.1
  - XML (SOAP)
  - JSON

22

RPC Compiler

- Server
  - Defines a set of remote procedures
- Define signatures
  - Name of procedure
  - Type/Order of arguments
  - Return type
- Interface Definition Language
- IDL compiler generates
  - Client stubs
  - Server stubs
  - Additional server functionality



23

OO-RPC

24

Aus RPC wird RMI

- Zusätzlicher Objektbezug wird deutlicher
- Siegeszug virtueller Maschinen
  - Java-Ökosystem
  - .NET Framework

25

JAVA RMI

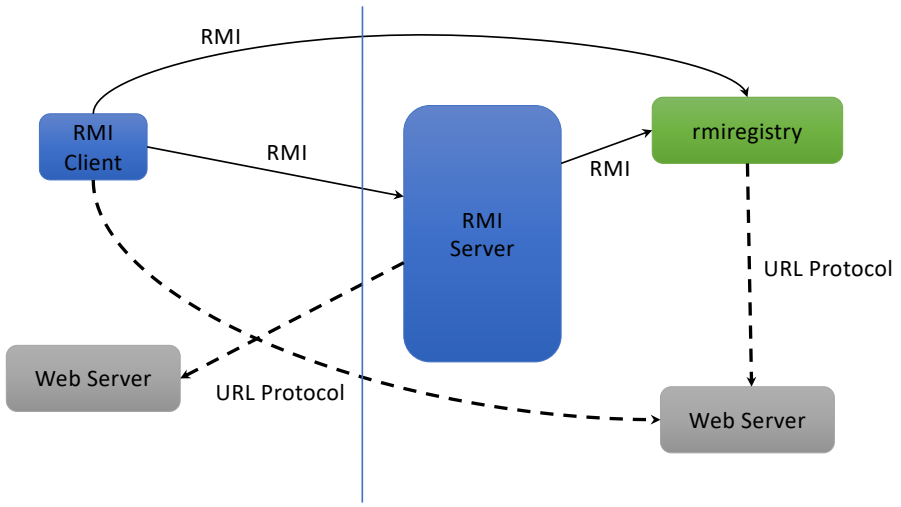
26

Java RMI

- Remote Method Invocation
- Java-basierte OO-Variante eines RPC
- Objekte inkl. Code können ausgetauscht werden
  - Vorteil einer homogenen Sprachumgebung
  - Sicherheitsproblematik „Function Shipping“
- Reflection
  - U.a. Generierung der Proxies

27

Architektur



28

http://java.sun.com/docs/books/tutorial/rmi/overview.html

Am Anfang war das Interface

```
package syssoft.rmi_example;

import java.rmi.Remote;
import java.rmi.RemoteException;

public interface Compute extends Remote {
    int Add ( int x, int y ) throws RemoteException;
    int Sub ( int x, int y ) throws RemoteException;
}
```

- Ableitung von Marker-Interface Remote (leeres Interface)
  - Erkennen RMI-zugänglicher Interfaces über Reflection
- Remote-Methoden „werfen“ RemoteException
  - Zugeständnis an mögliche Fehlerfälle in verteilten Systemen

29

Server

```
package syssoft.rmi_example.server;

import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.rmi.server.UnicastRemoteObject;

import syssoft.rmi_example.*;

public class Server implements Compute {

    public Server() { super(); }

    public int Add ( int x, int y ) { return x+y; }

    public static void main(String[] args) {
        if (System.getSecurityManager() == null) {
            System.setSecurityManager(new SecurityManager());
        }
        try {
            String name = "SyssoftCompute";
            Compute engine = new Server();
            Compute stub = (Compute) UnicastRemoteObject.exportObject(engine,0);
            Registry registry = LocateRegistry.getRegistry();
            registry.rebind(name, stub);
            System.out.println("ComputeEngine bound");
        } catch (Exception e) {
            System.err.println("ComputeEngine exception:");
            e.printStackTrace();
        }
    }
}
```

30

Client

```
package syssoft.rmi_example.client;

import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;

import syssoft.rmi_example.*;

public class Client {

      public static void main(String[] args) {
            if (System.getSecurityManager() == null) {
                  System.setSecurityManager(new SecurityManager());
            }
            try {
                  String name = "SyssoftCompute";
                  Registry registry = LocateRegistry.getRegistry(args[0]);
                  Compute comp = (Compute) registry.lookup(name);

                  int result = comp.Add(21, 21);
                  System.out.println(result);
            } catch (Exception e) {
                  System.err.println("SyssoftCompute exception:");
                  e.printStackTrace();
            }
      }
}
```

31

42

```
# Compiling, packaging, and distributing the interface
javac syssoft/rmi_example/Compute.java
jar cvf run/Compute.jar syssoft/rmi_example/Compute.class
cp run/Compute.jar ~peter/Sites

# Preparing the server
javac -cp run/Compute.jar syssoft/rmi_example/server/Server.java

# Preparing the client
javac -cp run/Compute.jar syssoft/rmi_example/client/Client.java

# Starting RMI registry
# rmiregistry &

# Running server
java -cp .:run/Compute.jar -
Djava.rmi.server.codebase=http://localhost/~peter/Compute.jar
      -Djava.rmi.server.hostname=localhost -Djava.security.policy=run/server.policy
      syssoft.rmi_example.server.Server &
sleep 10

# Running client
java -cp .:run/Compute.jar -Djava.rmi.server.codebase=http://localhost/~peter/
      -Djava.security.policy=run/client.policy  syssoft.rmi_example.client.Client
localhost
```

32

.NET Remoting

33

Interface

```
using System;
using System.Collections.Generic;
using System.Text;

namespace Calculator_Interfaces
{
    public interface ICalculator
    {
        int Sum(int x, int y);
        int Sub(int x, int y);
        int Mul(int x, int y);
        int Div(int x, int y);
    }
}
```

34

Implementierung

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Runtime.Remoting.Lifetime;

namespace Calculator_Server
{
    public class CalculatorImpl : System.MarshalByRefObject, Calculator_Interfaces.ICalculator
    {

        static int instanceCounter = 0;

        #region ICalculator Members

        public CalculatorImpl()...

        ~CalculatorImpl()...

        public int Sum(int x, int y)...

        public int Sub(int x, int y)...

        public int Mul(int x, int y)...

        public int Div(int x, int y)...

        public override object InitializeLifetimeService()...

        #endregion
    }
}
```

35

Server

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Runtime.Remoting;
using System.Runtime.Remoting.Channels;
using System.Runtime.Remoting.Channels.Tcp;

namespace Calculator_Server
{
    class Program
    {

        static void Main(string[] args)
        {
            // nutze Port 4242 für den Server
            TcpChannel tcp = new TcpChannel(4242);
            ChannelServices.RegisterChannel(tcp, true);

            // registriere Implementierung (über den Typ) (für Client-Side Activation)
            RemotingConfiguration.RegisterWellKnownServiceType(typeof(Calculator_Server.CalculatorImpl),
                "ClientActivated_SingleCall/Calculator", WellKnownObjectMode.SingleCall);
            RemotingConfiguration.RegisterWellKnownServiceType(typeof(Calculator_Server.CalculatorImpl),
                "ClientActivated_SingleTon/Calculator", WellKnownObjectMode.Singleton);

            // registriere Implementierung (über eine Instanz) (für Server-Side Activation)
            RemotingServices.Marshal(new CalculatorImpl(), "ServerActivated/Calculator");


            Console.WriteLine("Beliebige Taste drücken um zu beenden...");
            Console.ReadKey();
        }
    }
}
```

36

Client

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Runtime.Remoting;
using System.Runtime.Remoting.Channels;
using System.Runtime.Remoting.Channels.Tcp;

using Calculator_Interfaces;

namespace Calculator_Client
{
    class Program
    {
        static Calculator_Interfaces.ICalculator calc1, calc2, calc3 = null;

        static void Main(string[] args)
        {
            // nutze irgendeinen freien Port
            TcpChannel tcp = new TcpChannel();
            ChannelServices.RegisterChannel(tcp, true);

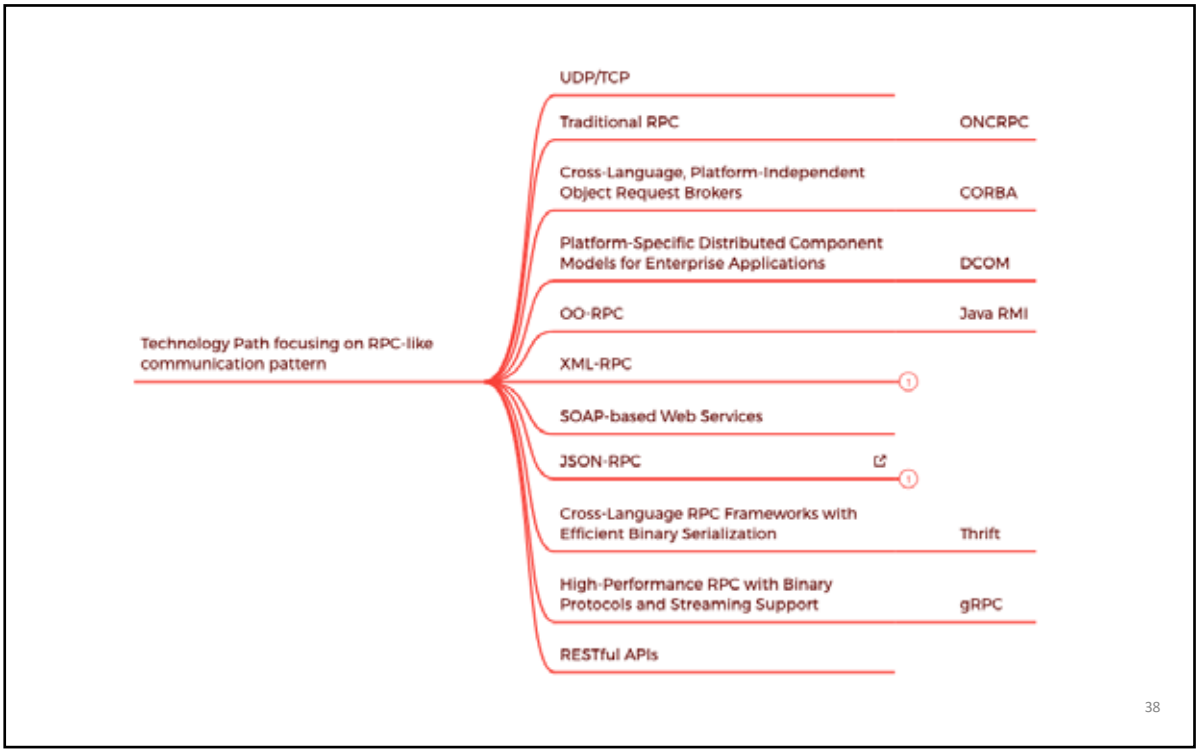
            // fordere Proxy-Objekt an (Client-Activated SingleCall)
            calc1 = (ICalculator)Activator.GetObject(typeof(ICalculator),
                "tcp://localhost:4242/ClientActivated_SingleCall/Calculator");

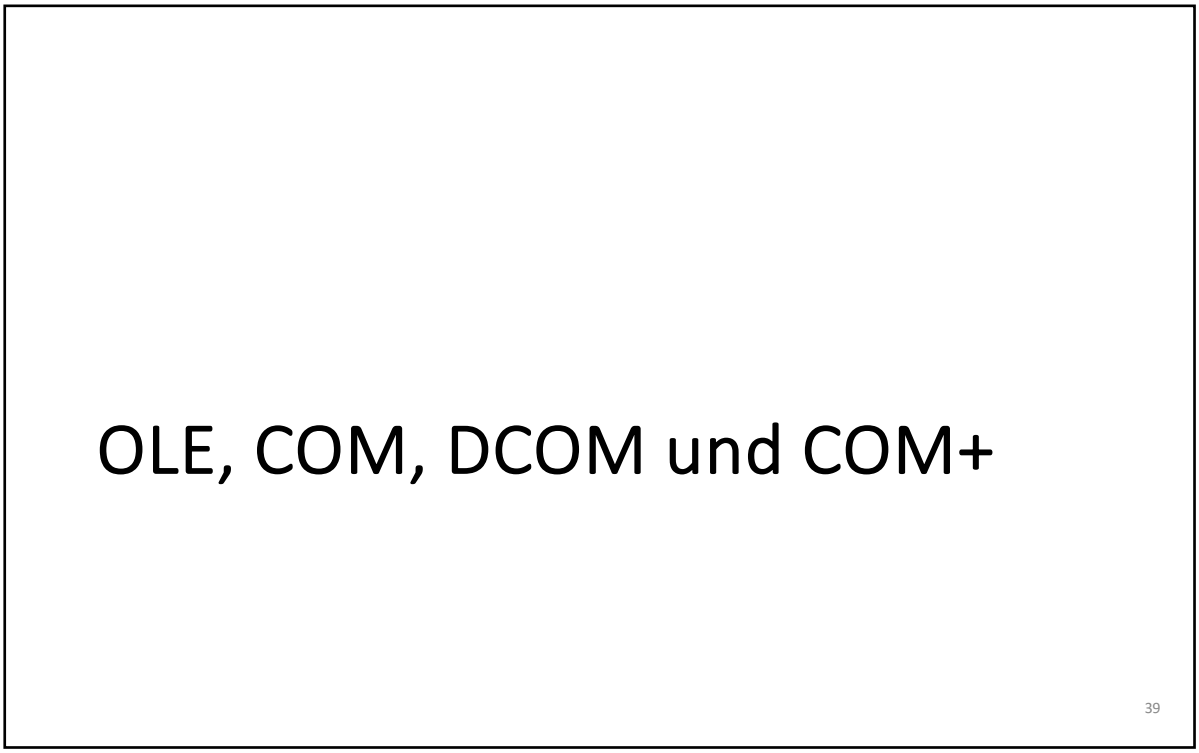
            // fordere Proxy-Objekt an (Client-Activated Singleton)
            calc2 = (ICalculator)Activator.GetObject(typeof(ICalculator),
                "tcp://localhost:4242/ClientActivated_SingleTon/Calculator");

            // fordere Proxy-Objekt an (Server-Activated)
            calc3 = (ICalculator)Activator.GetObject(typeof(ICalculator),
                "tcp://localhost:4242/ServerActivated/Calculator");

            if (calc3 != null)
            {
                int ergebnis = calc1.Sum(20, 21);
            }
        }
    }
}
```

37


Technology Path focusing on RPC-like communication pattern
UDP/TCP
Traditional RPC
ONCRPC
Cross-Language, Platform-Independent Object Request Brokers
CORBA
Platform-Specific Distributed Component Models for Enterprise Applications
DCOM
OO-RPC
Java RMI
XML-RPC
SOAP-based Web Services
JSON-RPC
Cross-Language RPC Frameworks with Efficient Binary Serialization
Thrift
High-Performance RPC with Binary Protocols and Streaming Support
gRPC
RESTful APIs
38


OLE, COM, DCOM und COM+
39

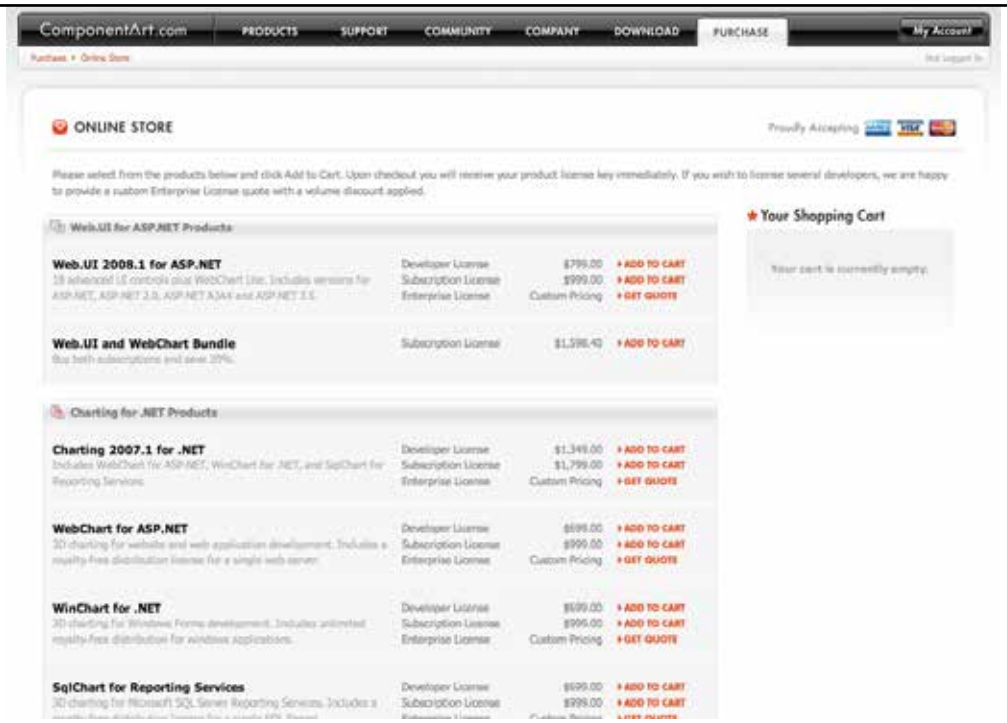
Microsofts erster Komponentenansatz

- Verwickelter Enstehungsprozeß
- “Noch immer” kein allgemein definierter Standard
  - Draft, As is
  - Nur noch wegen Rückwärtskompatibilität bedeutungsvoll
- Hohe Verbreitung
  - GUI-Bereich, Charts, …
  - Markt

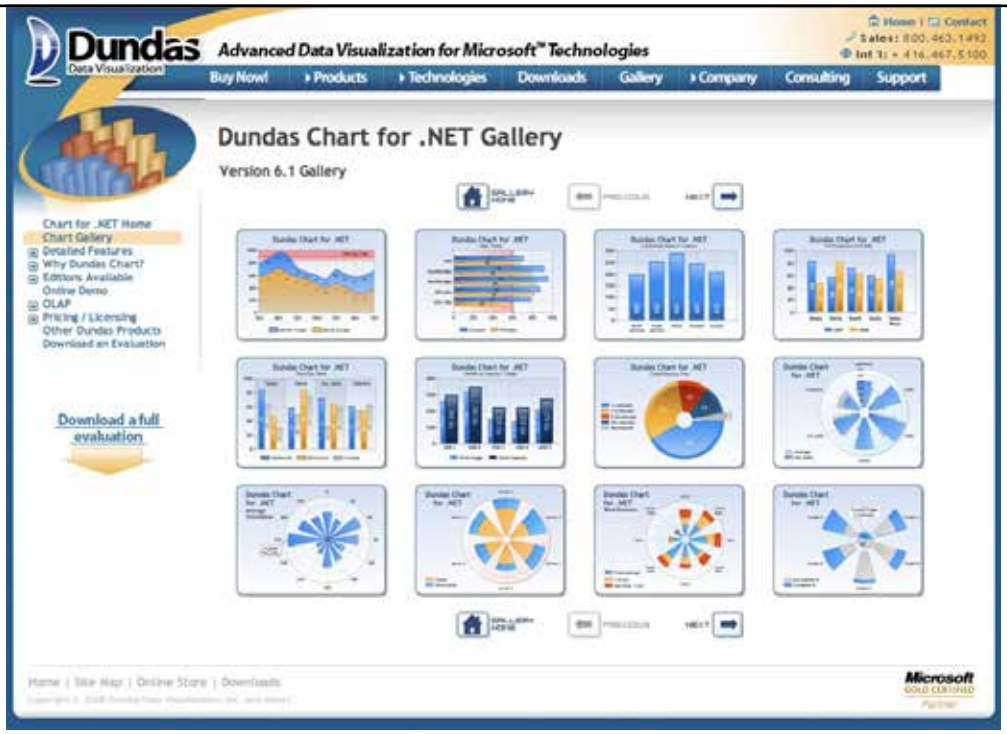
40

Der Komponentenmarkt

41

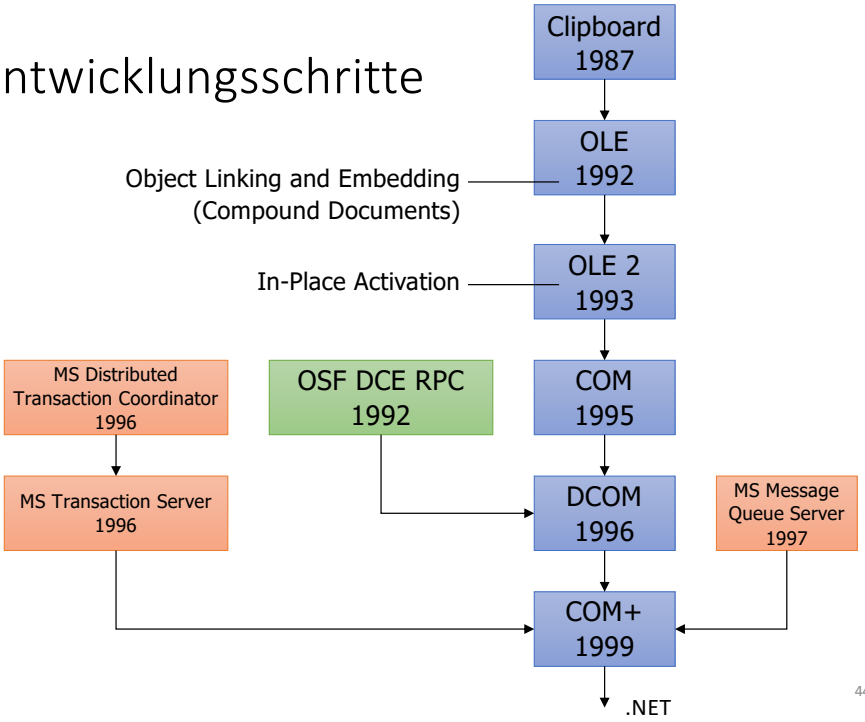

ComponentArt.com
PRODUCTS
SUPPORT
COMMUNITY
COMPANY
DOWNLOAD
PURCHASE
My Account
ONLINE STORE
Your Shopping Cart
Web.UI for ASP.NET Products
Web.UI 2008.1 for ASP.NET
Web.UI and WebChart Bundle
Charting for .NET Products
Charting 2007.1 for .NET
WebChart for ASP.NET
WinChart for .NET
SqlChart for Reporting Services

42


Dundas
Data Visualization
Advanced Data Visualization for Microsoft™ Technologies
Buy Now!
Products
Technologies
Downloads
Gallery
Company
Consulting
Support
Dundas Chart for .NET Gallery
Version 6.1 Gallery
Chart for .NET Home
Chart Gallery
Detailed Features
Why Dundas Chart?
Editions Available
Online Demo
OLAP
Pricing / Licensing
Other Dundas Products
Download an Evaluation
Download a full evaluation
Microsoft

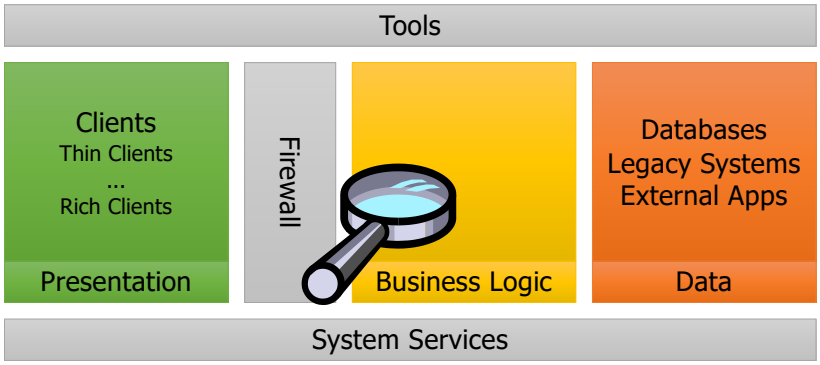
43

COM+ Entwicklungsschritte



44

Windows DNA



- DNA = Distributed interNet Application Architecture
- 3-Tier Ansatz

45

Component Services

- COM
  - Interface-based Programming
  - Basic Component Facilities
- DCOM
  - Remoting Architecture
  - Distributed Component Services
- COM+
  - Load Balancing
  - In-memory Database
  - Object Pooling
  - Queued Components
  - Event Model
- MTS
  - Transaction Services
  - Resource Pooling
  - Role-based Security
  - Administration
  - Just-in-time Activation

46

Basics

- Binärer Komponentenstandard
  - Implementierungssprache unbestimmt
- Komponente realisieren Interfaces
- Interface = Schnittstellenbeschreibung
  - Abstrakte C++-Klasse mit virtuellen Funktionen
  - vtbl definiert Schnittstelle
- Minimal
  - **`IUnknown`**: Elementarmethoden jeder Komponente
  - **`IClassFactory`**: Komponentenerzeugung

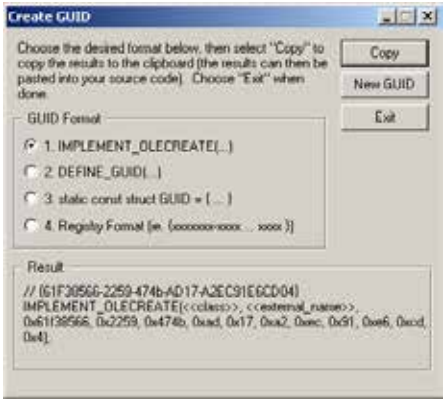
47

Identifikation von Komponenten

- Interfaces besitzen eine GUID
- 128 Bit „Zufallszahl”
  - z.B. IUnknown {00000000-0000-0000-C000-000000000046}
  - Erzeugung über API: **`CoCreateGuid()`**
  - Erzeugungstool: guidgen.exe
- Registry = Namensdienst

48

guidgen.exe



- Erzeugung eigener GUIDs
  - Hinreichend zufällig
- Bestandteile
  - Aktuelles Datum
  - Aktuelle Uhrzeit
  - Fortlaufende „Clock Sequence“ (persistent)
  - Inkrementeller Zähler (hochfrequente Abfragen)
  - MAC-Adresse der Netzkarte (geht aber auch ohne)

49

IUnknown

```
interface IUnknown
{
    typedef [unique] IUnknown *LPUNKOWN;

    HRESULT QueryInterface (
        [in] REFIID riid,
        [out, iid_is(riid)] void **ppvObject );

    ULONG AddRef ();

    ULONG Release ();
}
```

50

Beispiel

- Aufgabe
  - Übertragung von n ganzen Zahlen
  - „Komponente“ bildet die Summe
- Schritte
  - Interface definieren
  - Header-Dateien generieren
  - Komponente implementieren
  - Client programmieren
  - Registry-Eintrag

51

1. Interface

- Utility-Projekt in Visual Studio
- Interface **`Isum.idl`**:

```
import "unknwn.idl";

[ object, uuid(1E10C200-E306-11D3-A557-B6EEE489CA00) ]

interface ISum : IUnknown {

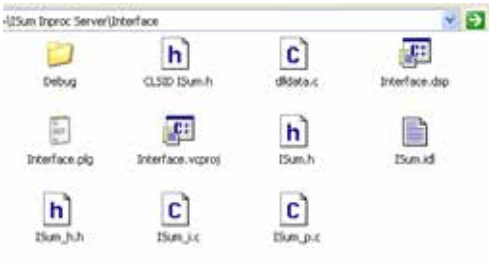
     HRESULT AddElements (
           [in] int nElem,
           [in, size_is(nElem)] int *elements,
           [out] int *sum
     );

};
```

52

2. MIDL ausführen

- Mehrere Dateien werden generiert
- Bedeutung:
  - C++-Version des Interfaces
  - GUID-Definitionen
  - Proxy-Stubs für entfernte Server



53

3. Komponente implementieren

- Jedes Interface muß implementiert werden
  - Ableitungen von den abstrakten Basisklassen
  - „Pure virtual functions“ implementieren
- Virtuelle Funktionen in jeder „Ableitung“ erneut implementiert
  - Binärstandard

54

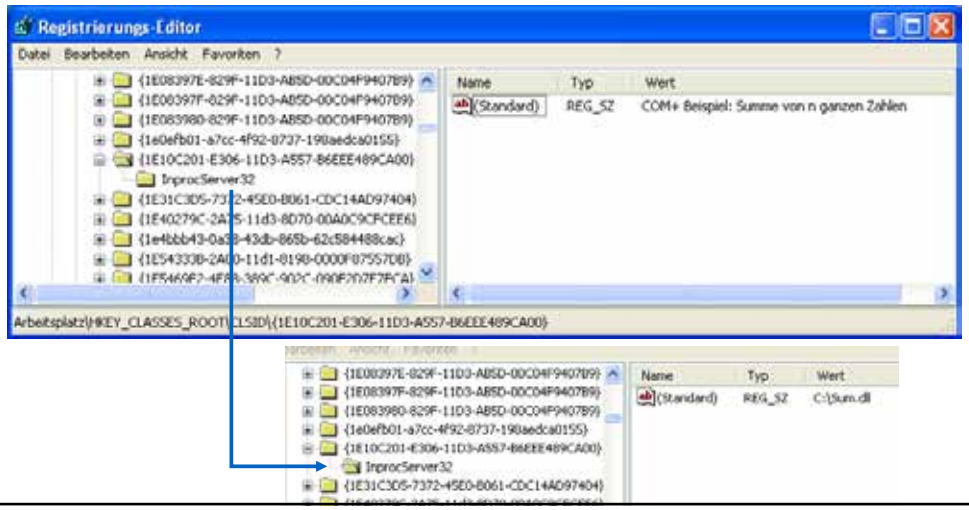
4. Client programmieren

- Initialisierung des Komponentenzugriffs
- **`Iunknown`**-Interface über GUID erfragen
- Downcast zum gewünschten Interface
- Komponente benutzen
- Explizites Reference-Counting beachten

55

5. Registry-Eintrag

- HKCR/CLSID/{GUID}



56

Remoting

57

Architekturen

- Inproc-Server
  - Komponente als DLL im Client
  - Funktionsaufrufe
  - Hohe Performance
- Server
  - Komponente als EXE auf Client-Rechner
  - Zugriff über Proxy-Objekte
  - Surrogate: Spezifische DLL-Container
  - COM-Laufzeitumgebung: MTS
- Remote Server
  - Komponente auf entferntem Rechner (meist in lokalem MTS)
  - Zugriff über Proxy-Objekte
  - Marshalling, RPC

58

ISum: Local Server

- Expliziter Aufruf eines lokalen Servers:
  - `hr = CoCreateInstance(CLSID_ISum, NULL, CLSCTX_LOCAL_SERVER, IID_IUnknown, (void **) &pUnknown);`

  statt
  - `hr = CoCreateInstance(CLSID_ISum, NULL, CLSCTX_INPROC_SERVER, IID_IUnknown, (void **) &pUnknown);`
- Es geht aber auch transparent
  - Zuerst nach Inproc-Server schauen
  - Dann nach lokalem Server
  - Dann ggf. entfernter Server

59

Wir verlassen die Prozeßgrenzen

- RPC = Remote Procedure Call
  - Argumente in Nachricht verpacken
  - Nachrichtenübertragung an den Server
  - Auf Antwortnachricht warten
  - Ergebnis an den Aufrufer zurückgeben
- Marshalling und Unmarshalling
  - MIDL generiert die notwendigen Routinen automatisch
  - Proxy = Client-Seite
  - Stub = Server-Seite
- Verankerung der Proxy- und Stubroutinen in Registry

60

Automation

61

Interfacevarianten

- Statischer Zugriff
  - Interface der gewünschten Komponente bekannt
  - Zugriff über generierte Schnittstellenbeschreibung
  - Überprüfungen zur Übersetzungszeit
- Dynamischer Zugriff
  - Interfacebeschreibung liegt in Form einer Type Library vor
  - Erfragen der Schnittstelle
  - Dynamischer Methodenaufruf
  - Keine Überprüfungen zur Übersetzungszeit
  - Vgl. Reflection(Java), Dynamic Interfaces (Corba)
  - Beim Zugriff über VB, VBA oder Java notwendig

62

Dynamische Interfaces

- Automation
- Schnittstelle **`IDispatch`**
- Kombination aus statischem und dynamischem Interface möglich

63

IDispatch

```
[
    object,
    uuid(00020400-0000-0000-C000-000000000046),
    pointer_default(unique)
]

interface IDispatch : IUnknown
{
    typedef [unique] IDispatch * LPDISPATCH;

    HRESULT GetTypeInfoCount(
        [out] UINT * pctinfo );

    HRESULT GetTypeInfo(
        [in] UINT iTInfo,
        [in] LCID lcid,
        [out] ITypeInfo ** ppTInfo);

    ...
```

64

IDispatch(2)

```
    HRESULT GetIDsOfNames(
        [in] REFIID riid,
        [in, size_is(cNames)] LPOLESTR * rgszNames,
        [in] UINT cNames,
        [in] LCID lcid,
        [out, size_is(cNames)] DISPID * rgDispId );

    [local]
    HRESULT Invoke(
        [in] DISPID dispIdMember,
        [in] REFIID riid,
        [in] LCID lcid,
        [in] WORD wFlags,
        [in, out] DISPPARAMS * pDispParams,
        [out] VARIANT * pVarResult,
        [out] EXCEPINFO * pExcepInfo,
        [out] UINT * puArgErr );
```

65

IDispatch(3)

```
[call_as(Invoke)]
    HRESULT RemoteInvoke(
        [in] DISPID dispIdMember,
        [in] REFIID riid,
        [in] LCID lcid,
        [in] DWORD dwFlags,
        [in] DISPPARAMS * pDispParams,
        [out] VARIANT * pVarResult,
        [out] EXCEPINFO * pExcepInfo,
        [out] UINT * pArgErr,
        [in] UINT cVarRef,
        [in, size_is(cVarRef)] UINT * rgVarRefIdx,
        [in, out, size_is(cVarRef)]
            VARIANTARG * rgVarRef
    );
}
```

66

Active Template Library

67

ATL

- Active Template Library
  - Teil der MFC
  - Basiert auf STL
- Grundfunktionen
  - Implementierungen
    für Iunknown, IClassFactory und IDispatch
  - Komponentenregistrierung
- DLL oder EXE möglich

68

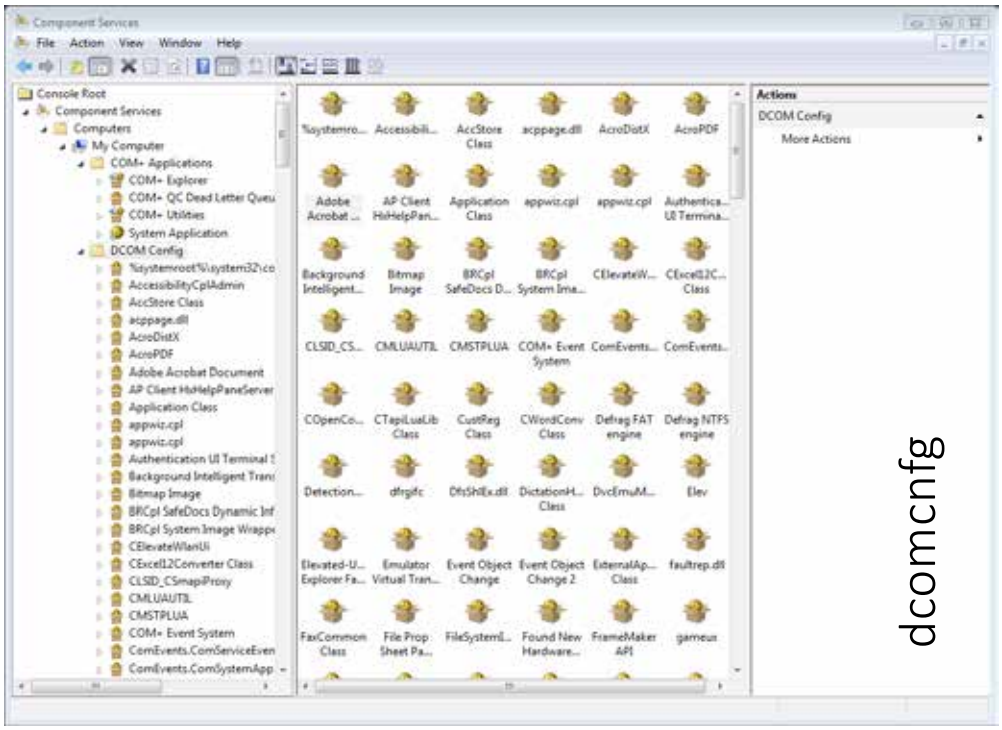
Konfiguration

69

Spielarten

- Manuell
  - Eingriffe in die Registry (Brrrr)
- Selbstregistrierende Komponenten
  - Vorgehensweise ATL
  - Problematik Deregistrierung
- Dcomcnfg: Weitergehende Konfiguration
  - Veränderungen an der Remoting-Struktur

70



71

Literatur

- Guy Eddon, Henry Eddon
  Inside COM+ Base Services
  Microsoft Press 1999

  geplanter 2. Band „Inside COM+ Component Services“ ist immer noch nicht erschienen ☺

72

CORBA

73

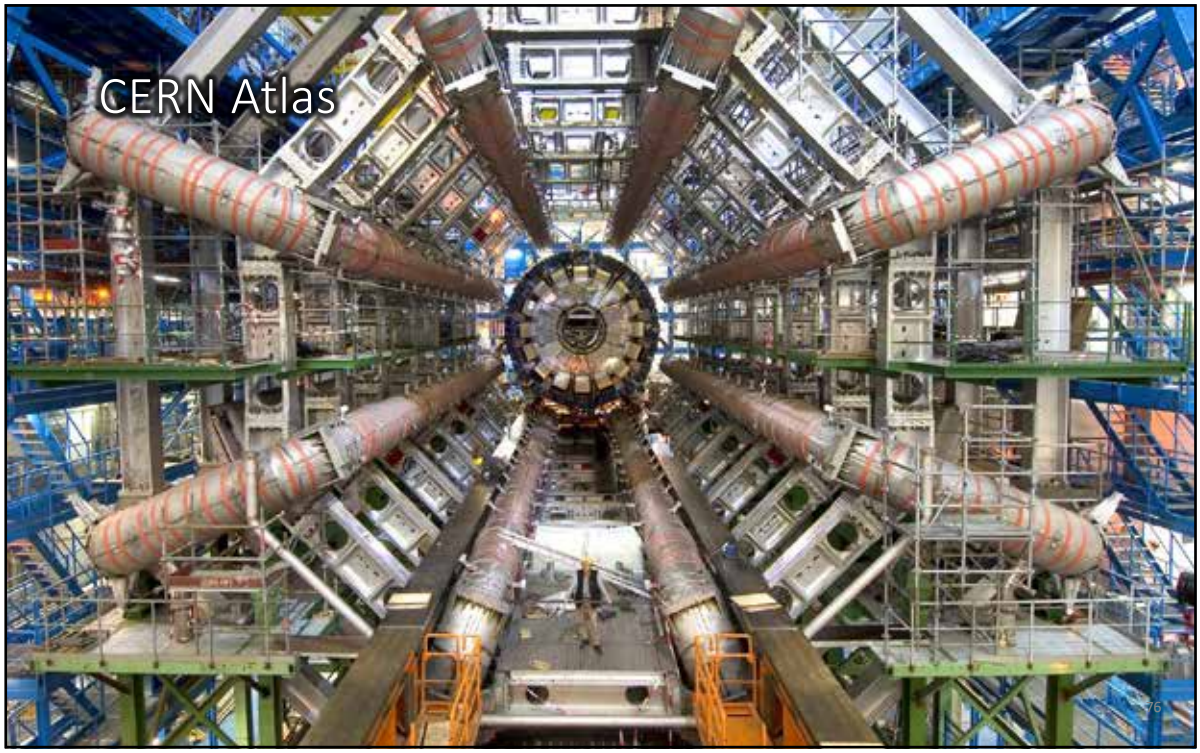
CORBA (1991-heute)

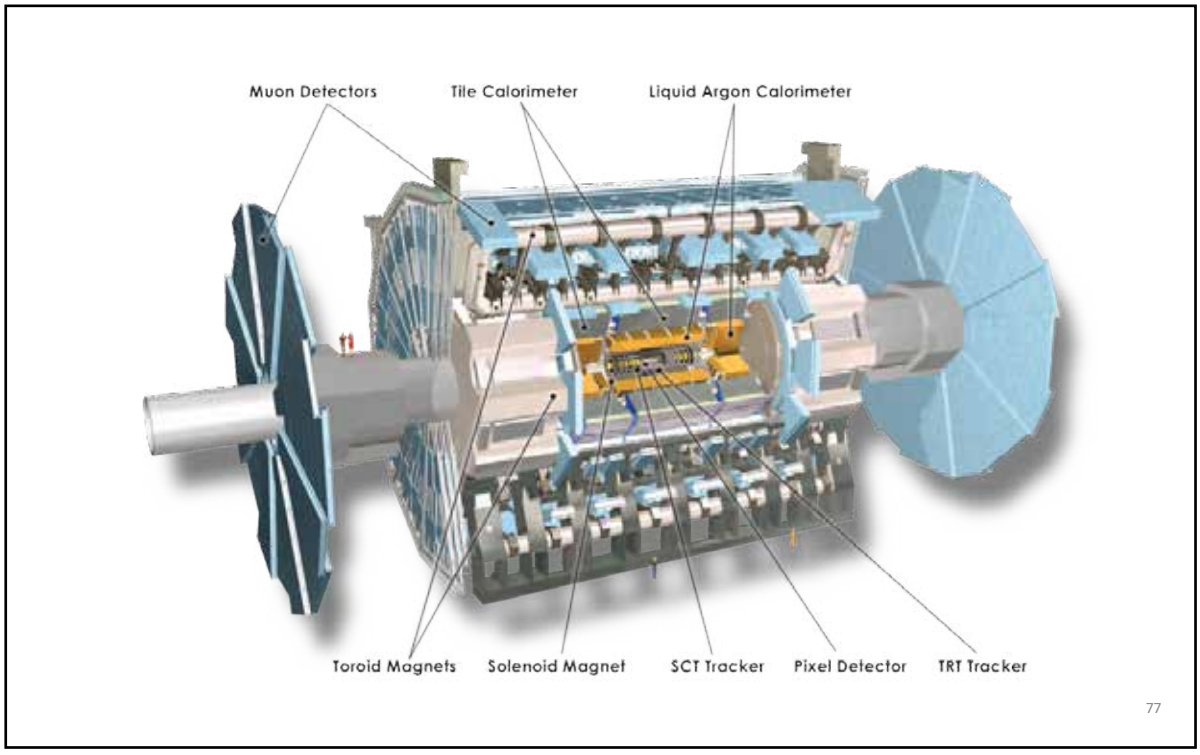
- Common Object Request Broker Architecture
- Ziele
  - Verteilte Anwendungen mit objektorientierten Methoden
  - Objektorientierte Softwarebausteine (Komponenten)
  - Schnittstellenstandards für verteilte Büroanwendungen
- Zurückgehende Bedeutung
  - Telekommunikationsbereich
  - Wissenschaftliche Systeme (z.B. CERN, Genf)

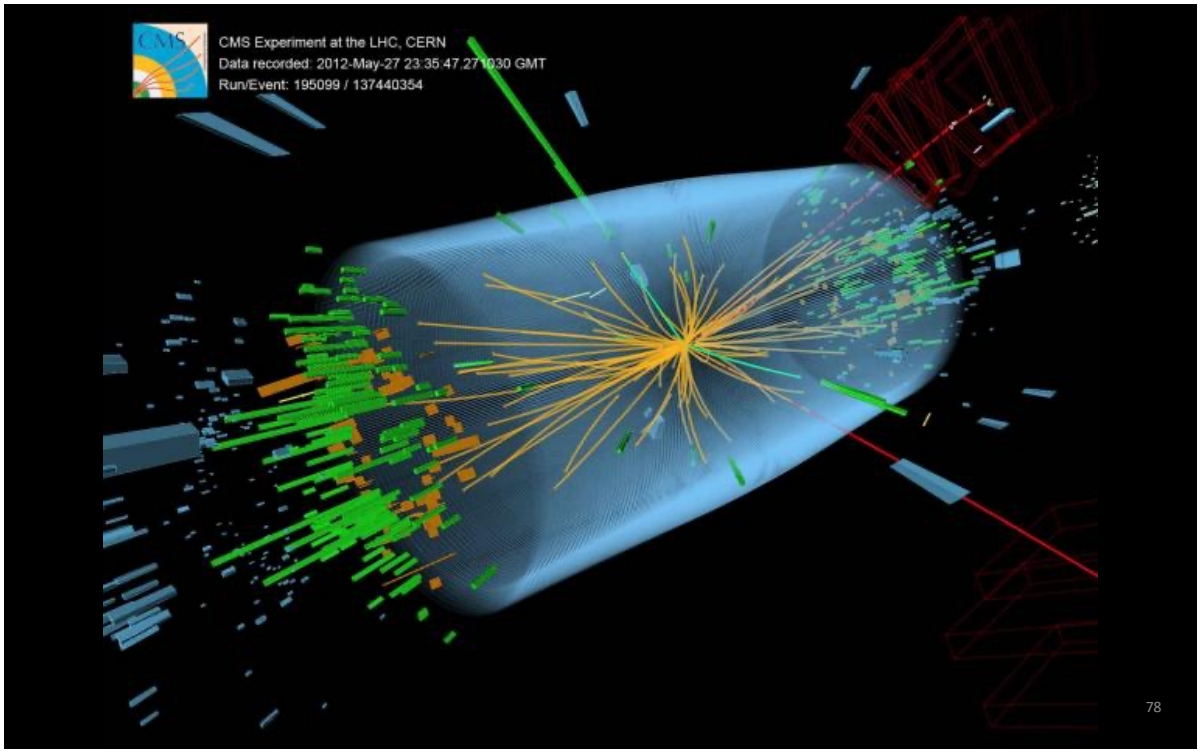
74

CERN?

75


CERN Atlas


Muon Detectors
Tile Calorimeter
Liquid Argon Calorimeter
Toroid Magnets
Solenoid Magnet
SCT Tracker
Pixel Detector
TRT Tracker
77



Problem

- Eine Datenquelle
- Viele Ereignisse
  - Wenige interessant
- Sehr große Datenmengen in sehr kurzer Zeit
- 15 PByte Daten entstehen pro Jahr
- Komplexes verteiltes Softwaresystem

79

LHC

- Large Hadron Collider (LHC)
- 40 Millionen Ereignisse pro Sekunde
- Auslese mittels Trigger
  - Level 1 wählt 75000 pro Sekunde aus
  - Level 2 reduziert auf 1000 pro Sekunde
  - Level 3 wählt ca. 200/s aus und speichert Daten

80

Historische Einordnung

- 1977 erste Pläne
- 1984 wurde es ernst
- Parallel
  - Bau des LHC
  - Konzeption und Realisierung der Software
- 1999 Plattform-Entscheidungen
- 2009 beginnen die Experimente

81

Zurück zu CORBA

82

Historisches

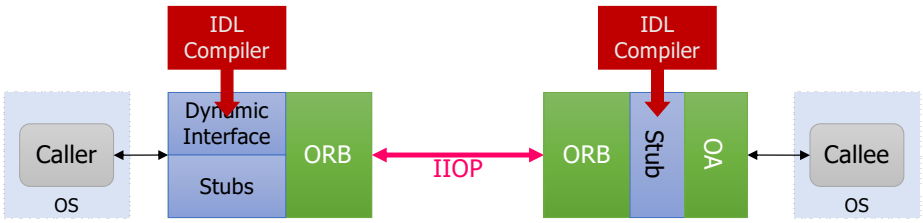
- Standard der Object Management Group (OMG)
  - 1991 CORBA 1.1
  - 1994 CORBA 2.0
  - seit ca. 2001 CORBA 3.0
- OMG
  - Herstellerübergreifendes Konsortium
  - Gegründet 1989 von 11 Mitgliedern
  - Aktuell mehr als 600 Mitglieder

1996 Java JDK 1.0
1997 Java RMI

83

Aufbau

- Eigene CORBA-IDL
- ORB = Object Request Broker
  - Kommerziell und Open Source



84

IDL

- CORBA-IDL = Mächtige Schnittstellenbeschreibung
- Alles drin
  - Basistypen: Zahlen, Strings, …
  - Benutzerdefinierte Typen: Strukturen, Arrays, …
  - Dynamische Strukturen: Bäume, Graphen, ...
  - Interfaces und Operationen
  - Module
  - …

85

Interface

- Bestandteile
  - Konstanten und Typdefinitionen
  - Exceptions
  - Attribute (Data Members)
  - Operationen (Function Members)
- Directional Attribute
  - in
    - Parameter wird vom Client zum Server gesendet
  - out
    - Parameter wird vom Server zum Client gesendet
  - inout
    - Paramter wird vom Client zum Server gesendet. Der resultierende Wert wird nach Beendigung der Operation an den aufrufenden Client zurückgesendet

86

Operationen

- Operationsdefinition
  - Ergebnistyp
  - Operationsname
  - 0 oder mehr Parameterdeklarationen
- Overloading nicht erlaubt
  (keine gleichen Operationsnamen)
- Einwegoperationen
  - ```
    typedef sequence<octet> bytestream;
    oneway void send ( in bytestream data );
    ```

87
    ```

IDL ⇒ C++ Mapping

- Vergleichsweise direkt, da C++ mächtiger ist
- Interface = abstrakte Klassen

```
interface IF {
    long f ();
}

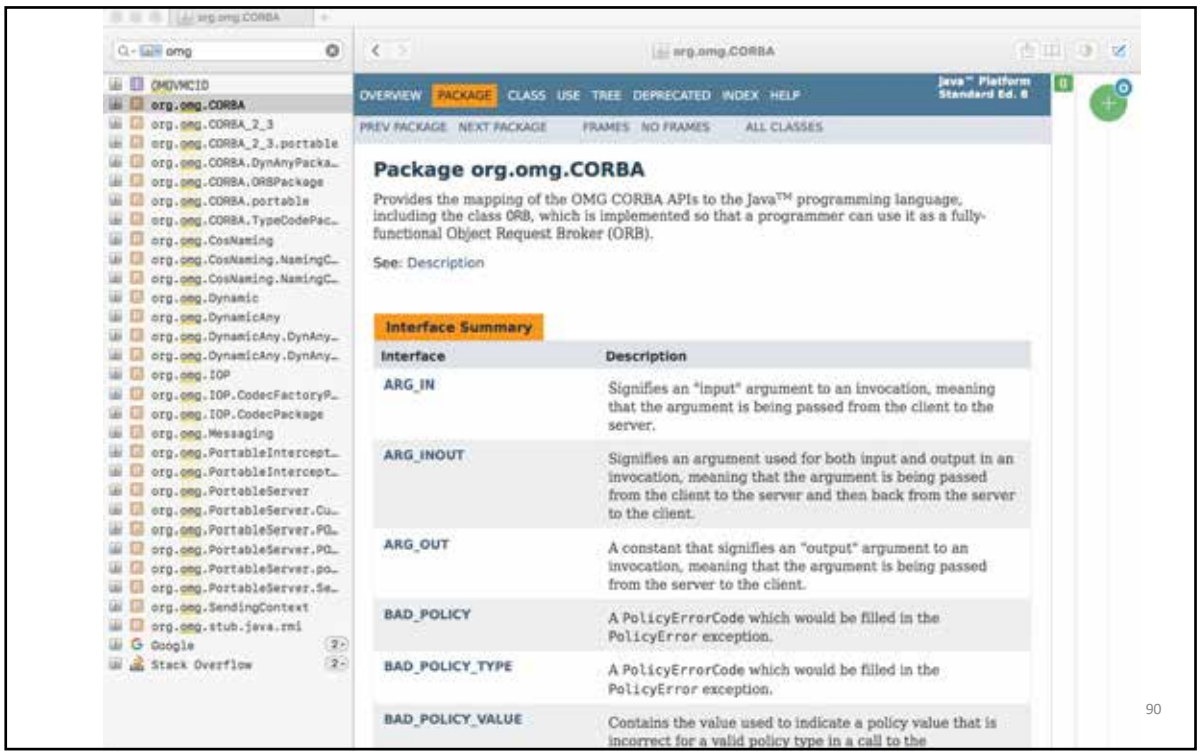
class IF : public virtual CORBA::Object {
public:
    virtual CORBA::Long f () = 0;
    …
}
```

88

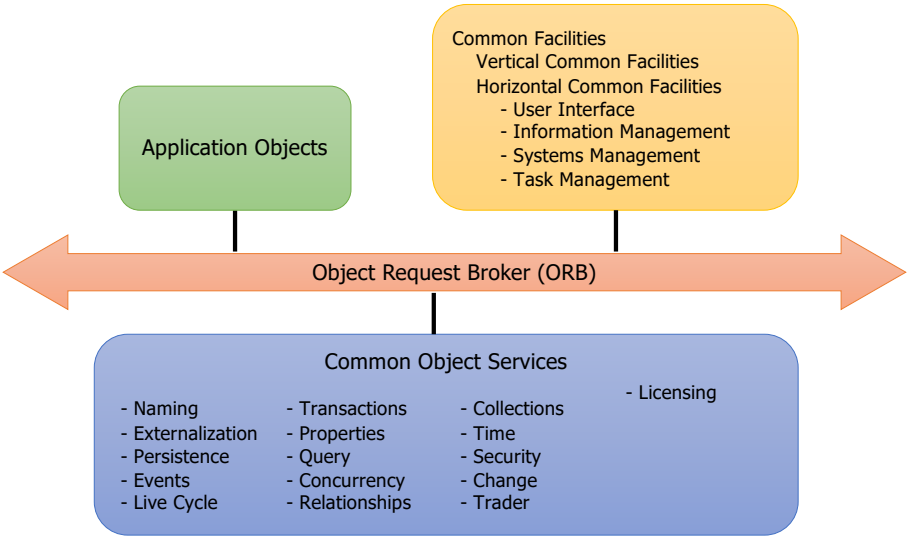
IDL ⇒ Java

- Grundfunktionalität über Package org.omg.*
- Ansonsten Generierung der Java-Mappings vergleichbar C++
- Aus einem IDL-Interface X wird
  - Java-Interface X für Client (Signature Interface)
  - Java-Interface X_Operations für Server (Operations Interface)

89



Object Management Architecture (OMA)



91

Common Object Service Specification (COSS)

- Gültig für alle CORBA-konformen Plattformen
- Naming
- Externalisation
  - Export von Zuständen in „flache“ Dateien
- Persistenz
  - Dauerhafte Objektzustände
- Events
  - Weiterleitung asynchroner Ereignisse
- Lifecycle
  - Unterstützung des Objekt-Lebenszyklus

92

COSS contd.

- Transactions
  - 2 Phase-Commit-Protokoll
  - Flache und geschachtelte Transaktionen
- Properties
  - Attribut/Wert-Paare für Objekte speichern und abfragen
- Query
  - SQL-Abfragen
- Concurrency
  - Verwaltung und Realisierung von Locks
- Relationships
  - Gruppierung von Objekten

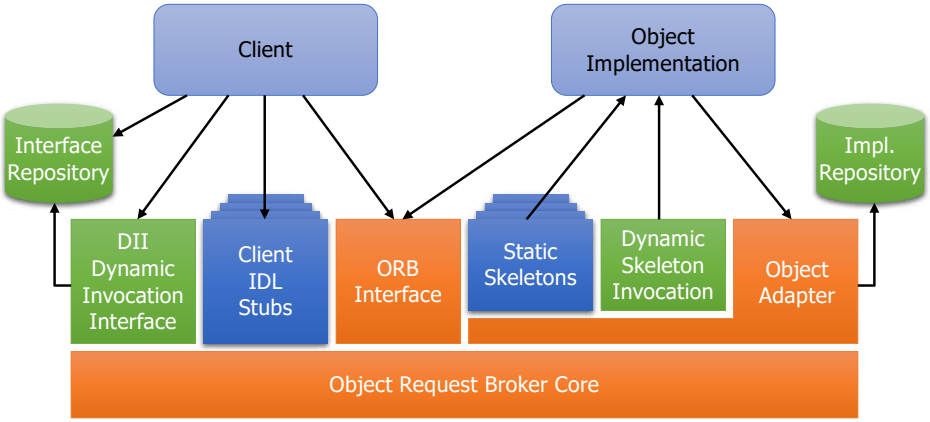
93

COSS contd. (2)

- Collections
- Time
- Security
- Change Management
- Trader
- Licensing

94

CORBA ORB



95

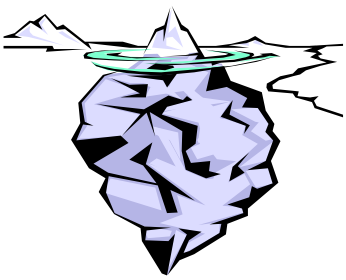
Methodenaufrufe

- Schnittstellenspezifikation über IDL
  - Sprachbindungen für C, C++, Smalltalk, Java, Cobol, ...
- Aufruf
  - Zielobjekt
  - Aufrufparameter
  - Eventuell Rückgabewerte, Exceptions
- Aufrufsemantik
  - Synchroner Auftrag (vgl. RPC)
  - Asynchroner Auftrag
  - Asynchrone Meldung
- Klassisch statische Schnittstellen (Stubs)
- Dynamische Schnittstellen

96

Object Adapter

- Steuert Funktionen des Server-Objekts
  - Aktivierung des Objekts bei eingehendem Request
  - Authentifizierung des Aufrufers
  - Zuordnung Objektreferenzen zu Instanzen
  - Registrierung des Server-Objekts
  - Lebensdauer des Server-Objekts
- Basic Object Adapter (BOA)
  - Standardadapter
- Extremfall
  - ORB und BOA Laufzeitbibliothek in der Anwendung
  - Leistung mit Unterprogrammaufruf vergleichbar
  - Neue Basisarchitektur für Betriebssysteme?



97

Object Adapter

- Schnittstelle zwischen Skeleton und ORB
- Verwaltet Objekte
  - Instanziierung
  - Lebenszyklus
  - Binden an Implementierung, etc.

98

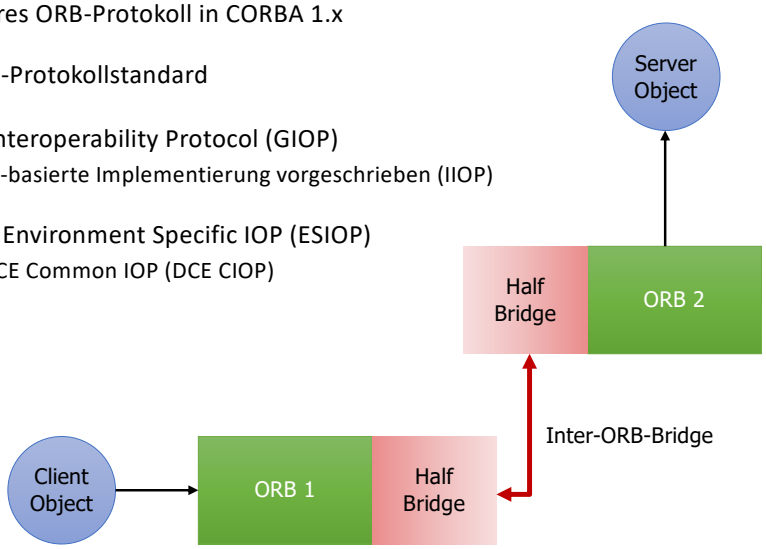
Object Adapter?

- Warum nicht im ORB?
  - Austauschbare Komponente
  - Ziel: ORB möglichst klein halten (µORB)
- BOA
  - Basic Object Adapter
  - Single-Threaded (Multi-process)
- POA
  - Portable Object Adapter
  - Single- oder Multithreaded
  - Definition von Objektgruppen etc.
- Ab CORBA 2.3 ersetzt der POA den BOA

99

ORB-2-ORB-Kommunikation

- Proprietäres ORB-Protokoll in CORBA 1.x
- Inter-ORB-Protokollstandard
- General Interoperability Protocol (GIOP)
  - TCP/IP-basierte Implementierung vorgeschrieben (IIOP)
- Optional: Environment Specific IOP (ESIOP)
  - Z.B. DCE Common IOP (DCE CIOP)



100

Objektreferenzen

- Schlüsselkonzept der Transparenz
- Zugriff auf Methoden
- Objektreferenzen über Adreßräume hinweg?
  - Referenz auf Proxy-Objekt
- Referenzen …
  - … sind global eindeutig!
  - … können als Argumente übergeben werden!
  - … sind interoperabel

101

Interoperable Object Reference

- IOR
  - Enthält IP und Port des “zuständigen” ORB
  - ID des Object Adapters
  - Eindeutige ID des Objekts
  - Kann als String repräsentiert werden
- Woher erhält Client eine IOR?
  - Einfachste Methode = Gemeinsame Datei
  - Namensdienste

```
IOR:010000001600000049444c3a6970632f706172746974696f6e3a312e3000000001000000000000006c
0000000101020000d0000003133372e3133382e352e36370000ad8316000000ff6970632f70617274697469
6f6e00696e697469616c0000020000000000000000800000001000000005454410100000001c000000010000
0001000100010000000100010509010100010000009010100
```

102

CCM

- CORBA Component Model
- Bestandteil der IDL (CIDL)
- Komponenten haben ein Home
- CCM in der Praxis nicht weit verbreitet

103

Bemerkungen

- Standards, Standards, Standards, …
  - CORBA-Architektur inkl. IIOP: 1150 Seiten
  - C++ Language Binding: 184 Seiten
  - Java Language Binding: 158 + 84 Seiten
  - CORBA CCM: 434 Seiten
  - …
- War einmal technisch anspruchsvoll, aber andere Ansätze haben überholt
  - Sinnvoll einsetzbar bei Verknüpfung von Legacy Software
  - Verbreitet im Telekomumfeld
- Rückzieher
  - Aus CORBA in KDE wurde DCOP
- CORBA paßt sich an (… und wird noch komplexer)
  - Unterstützung und Anbindung an COM+
  - Unterstützung für WSDL und SOAP (.NET)
  - …

104

Literatur

- Advanced CORBA Programming with C++
  - *Michi Henning, Steve Vinoski*
  - *Addision Wesley, 9. Auflage, 2004*
- Middleware für verteilte Systeme
  - *Arno Puder, Kay Römer*
  - *Dpunkt Verlag, 1. Auflage 2001*

105