

F Microsoft & Verteilte Objekte

F.1 Überblick

- Terminologie
- COM Architektur
- Vergleich mit CORBA
- .NET Architektur

F.2 Literatur

- EdEd98.** G. Eddon, H. Eddon: *Inside Distributed COM*. Microsoft Programming Series, Microsoft Press, Redmond, Wash., 1998.
- OHE96.** R. Orfali, D. Harkey, J. Edwards: *The essential distributed objects survival guide*. John Wiley & Sons, 1996.
- Micr96.** Microsoft Corporation: *DCOM technical overview*. White paper. Redmond, Wash., 1996.
- Micr98.** Microsoft Corporation: *DCOM architecture*. White paper. Redmond, Wash., 1998.
- .
- Wan+97.** P. Chung, Y. Huang, S. Yajnik, D.-R. Liang, J. Shih, C.-Y. Wang, Y.-M. Wang: "DCOM and CORBA Side by Side, Step By Step, and Layer by Layer." In *C++ Report*, Jan. 1998.
<http://akpublic.research.att.com/~ymwang/papers/HTML/DCOMnCORBA/S.html>
- Kirt97.** M. Kirtland: "The COM+ Programming Model Makes it Easy to Write Components in Any Language." In *Microsoft Systems Journal*, Dec. 1997.
<http://www.microsoft.com/msj/1297/complus2/complus2.htm>

Weitere online-Artikel über .NET:

- Yale University
<http://www.yale.edu/tp/framework.htm>
- DISCOURSE: The Berlin Distributed Computing Laboratory
<http://www.discourse.de>

F.3 Terminology

1 OLE – Object Linking and Embedding

- Microsoft's standard for collaboration of software components
 - ◆ E.g., spreadsheet table cells in a text document
 - ◆ E.g., graphics in a spreadsheet table cell

- Defines object/component interfaces and protocols for
 - ◆ Linkage and notification for embedded components
 - ◆ "Drag and drop" of graphical objects
 - ◆ Clipboard
 - ◆ Structured storage (Compound files)
 - ◆ Scripting

- Microsoft Foundation Classes (MFC)
 - ◆ GUI programming and handling

2 COM – Component Object Model

- OLE's components belong to different processes/programs
 - ◆ Communication substrate needed

- COM as an object request broker and service provider
 - ◆ OLE components are COM objects
 - ◆ Single-machine environment

- Intra-address-space communication
 - ◆ Forwarding requests to other COM objects
 - ◆ Integration into the MFC event model

- Inter-address-space communication
 - ◆ Stubs
 - ◆ Light-weight RPC (LRPC)

3 DCOM – Distributed COM

- Extends COM to a distributed environment
 - ◆ DCE/RPC with at-most-once/exactly-once semantics

4 ActiveX

- COM enabled for the Internet (whatever that means)
 - ◆ *Just a marketing buzzword!*

5 COM+

- Improved programming environment for COM
 - ◆ Maps COM+ objects to COM objects
 - ◆ Handles reference counting and other standard procedures

F.4 COM Architecture

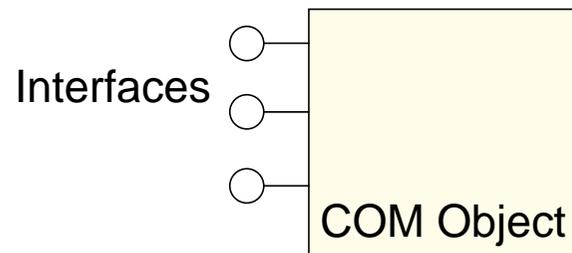
1 IDL – Interface Definition Language

- ▲ Not the same as CORBA IDL!
- Language for describing object interfaces
 - ◆ Independent from the target programming language
 - ◆ No mapping to language constructs
 - ◆ Definition of a binary object invocation interface (*vtables*)
- MIDL compiler = stub generator
 - ◆ Client stubs (proxies)
 - ◆ Server stubs

2 Object Model

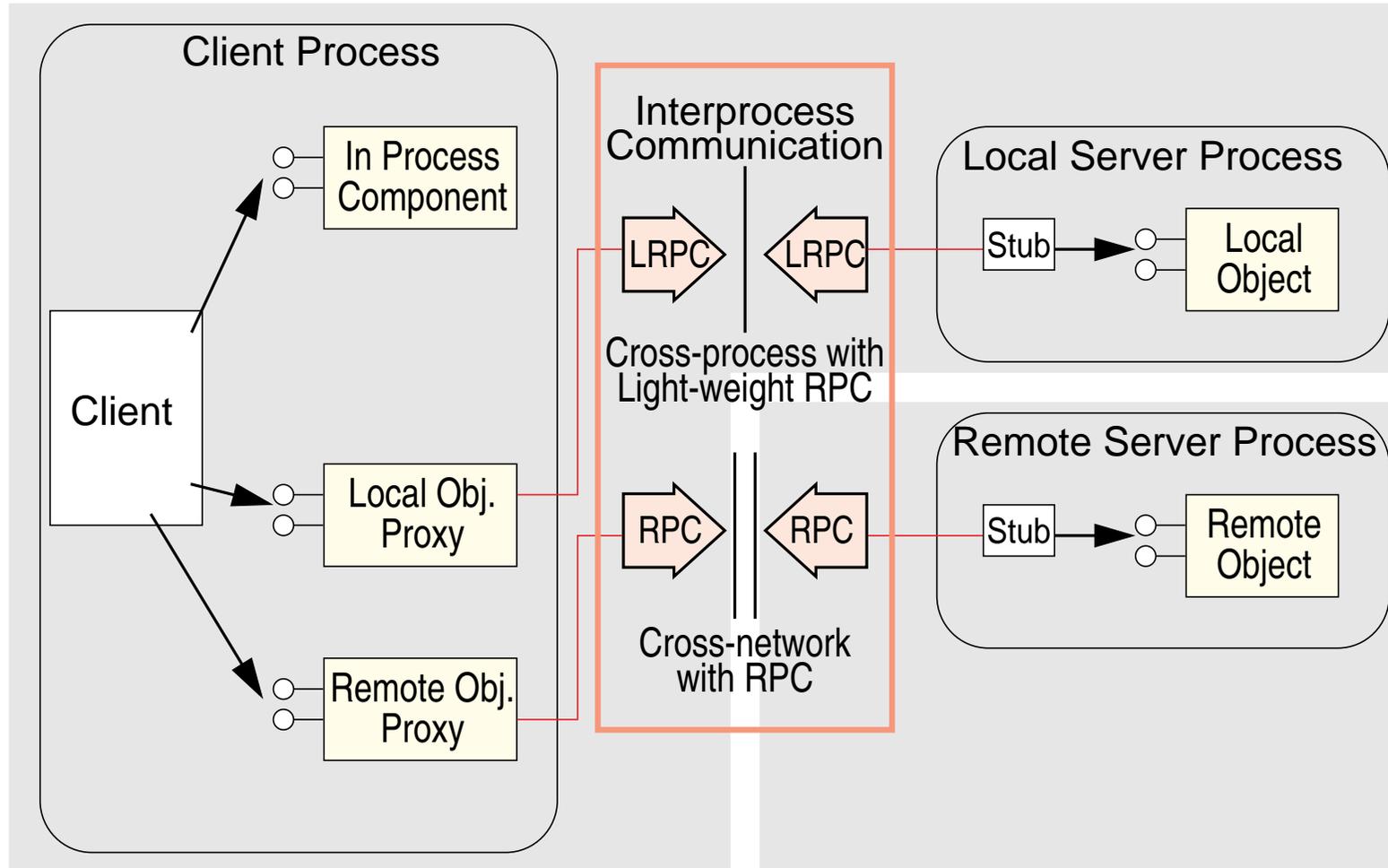
- Objects can have multiple interfaces
 - ◆ Multiple versions of one interfaces
 - ◆ Different interfaces for different purposes
 - ◆ Means to investigate the other interfaces

- Single inheritance on interfaces
 - ◆ Every interface inherits from **IUnknown**, which implements methods for finding other interfaces
 - ◆ Multiple inheritance must be emulated by multiple interfaces



- Centralized object approach

2 Object Model (2)



3 Process of Creation and Binding

- Creation of a server object
 - ◆ Description of the object interfaces in IDL
 - ◆ Programming server class and class factory in a target language
 - ◆ Registration of the class factory in the registry
 - ◆ On client demand an object is created
 - ◆ A transient object reference is marshalled and handed out to the client

- Binding to the server object at the client site
 - ◆ Retrieve class ID of factory object from the registry
 - ◆ Invoke `CoCreateInstance ()` method, which returns a reference to the object
 - ◆ Proxy (client stub) is automatically installed (code needs to be registered in the registry)
 - ◆ Method invocations using the proxy

3 Process of Creation and Binding (2)

- Proxies are COM objects
 - ◆ Class of the proxy object must be known at the client site (registered at the registry)

- *Custom Marshalling*
 - ◆ User may create his own proxy objects
 - Intelligent proxies
 - Non-RPC communication
 - ◆ Custom marshalling is similar to the fragmented object approach

4 Monikers

- COM does not know persistent object references
 - ◆ If a server object is deactivated the object reference will be invalid.

- Monikers
 - ◆ COM object
 - ◆ Knows a name for a "persistent" object
 - ◆ Can (re-)create the object and
 - ◆ feed it with its former state

- "Names"
 - ◆ URLs
 - ◆ Filenames
 - ◆ e.g., `c:\windows\test.xls!a1-d4` for spreadsheet cells in a particular file

F.5 Comparison to CORBA

- IDL and language mapping
 - ◆ **CORBA:** IDL is mapped to language constructs
 - Mapping is easier
 - ◆ **DCOM:** IDL defines binary data layout, language constructs are mapped to this layout
 - Heterogeneous binary component can be hosted in one address space

- Persistent object references
 - ◆ **CORBA:** POA and implementation repository
 - Arbitrary and user-defined implementations
 - ◆ **DCOM:** Monikers as mediators

F.5 Comparison to CORBA (2)

- Communication
 - ◆ **CORBA**: RPC-based invocation (at-most-once/exactly-once)
 - ◆ **DCOM**: RPC-based invocation (at-most-once/exactly-once) plus Custom Marshalling
 - Arbitrary communication mechanisms can be used

- Binding
 - ◆ **CORBA**: Interface-dependent stub must be known at client site
 - ◆ **DCOM**: Class ID and code of proxy must be registered at the registry

- Dynamic invocation
 - ◆ **CORBA**: DII, interface repository
 - ◆ **DCOM**: `IDispatch` interface, type library

F.5 Comparison to CORBA (3)

- Availability
 - ◆ **CORBA**: Virtually all platforms
 - ◆ **DCOM**: Windows 95/98/NT, MacOS, recently Solaris

- Bodies
 - ◆ **CORBA**: OMG and its several hundred members
 - ◆ **DCOM**: Microsoft and some supporters

- ★ CORBA defines gateways to the DCOM world
 - ◆ "Fully" interoperable

F.6 .NET-Architektur

1 Motivation & Ziele (1)

- 1. Ziel: Effiziente Ausführung sehr vieler Anwendungen
 - ◆ Ausführung in Prozessen ist zu teuer
 - Ressourcen pro Prozess
 - Umschaltung zwischen Prozessen
 - Interprozesskommunikation

- Lösung (1)
 - ◆ Ausführung in einem Adressraum
 - CICS (IBM, 1969)
 - ◆ Probleme
 - kein Schutz zwischen Anwendungen
 - Fehler in einer Anwendung können Gesamtsystem gefährden

1 Motivation & Ziele (1)

■ Lösung (2)

◆ Ausführung in einem Adressraum + Software-Schutzkonzept

- Java (Sun, 1995)

◆ Probleme

- Microsoft & Sun
- Sprachabhängigkeit
- Konzept für mehrere unabhängige Anwendungen in einer JVM

◆ gute Ideen

- Objektorientierung + Typisierung
- Bytecode mit voller Typinformation
- kontrollierte Code-Ausführung + Garbage Collection

1 Motivation & Ziele (2)

- 2. Ziel: Web-Anwendungen
 - ◆ sichere, effiziente Ausführung von Web-Services
 - ◆ flexible Ausführung von Software in heterogenen, verteilten Systemen

- Lösung
 - ◆ Ausführung in einem Adressraum + Software-Schutzkonzept
 - ◆ Hardware-Plattform-unabhängiger Code
 - ◆ Middleware-Unterstützung für Ausführung und Kommunikation
 - Java und EJB
 - ◆ Probleme
 - Microsoft & Sun
 - nur Java
 - nur bedingter Schutz (durch Typisierung) innerhalb einer JVM, aber keine Ressourcenkontrolle

2 .NET-Framework — Überblick

- Sicht des Anwendungsentwicklers
 - ◆ Softwareentwicklungsumgebung (Visual Studio .NET) mit umfangreicher Unterstützung durch Klassenbibliotheken
 - vergleichbar zu Java

- Systemarchitektur- und Betriebssystemansicht
 - ◆ Architektur einer Programmiersprachen-unabhängigen Ausführungsplattform
 - Common Language Infrastructure (CLI)
 - ◆ konkrete Ausführungsumgebung für Windows-Anwendungen: Betriebssystem innerhalb des Betriebssystems
 - Common Language Runtime (CLR)
 - Ausführung mehrerer Programme für verschiedene Benutzer
 - Ressourcenverwaltung
 - Gewährleistung von Sicherheit

3 Sprachunterstützung

- Programme werden in Zwischencode übersetzt
 - Microsoft Intermediate Language (MSIL)
 - ähnlich zu Java-Bytecode
- MSIL wurde sprachunabhängig entworfen
 - nicht auf Java beschränkt
 - nicht nur objektorientierte Sprachen
- Microsoft und Drittanbieter bieten Unterstützung für über 20 Sprachen an
 - C++
 - C# (C++-Derivat mit Ideen aus Java)
 - Visual Basic
 - JavaScript
 - J# (= Java 1.1.4)
 - Perl
 - COBOL, Smalltalk, APL, Pascal, ...

4 Common Language Infrastructure — CLI

- definiert die Regeln für die sprach-unabhängige Zusammenarbeit zwischen Anwendungen bzw. Software-Komponenten
 - Common Type System (CTS)
 - welche Datentypen gibt es und wie werden sie behandelt
 - Instruktionssatz (MSIL)
 - 220 Befehle, ausgerichtet auf JIT - nicht auf Interpretation
 - Thread-Aufbau
 - Aufbau von ladbaren Software-Modulen (Assemblies)
 - vergleichbar mit jar-Dateien
 - Laufzeitunterstützung durch die CLR
 - Laufzeitprüfung für statisch nicht prüfbare Dinge
 - standardisiert (mit C#) durch ECMA und ISO

5 Common Language Runtime

- Plattform zur kontrollierten Ausführung verschiedener .NET-Anwendungen innerhalb eines Windows-Prozesses
 - ◆ weitgehend vergleichbar mit Funktion der JVM
 - jeweils nur typ-konforme Operationen erlaubt
 - kein Zugriff auf "fremde" Daten (anderer Funktionen/Objekte)
 - Arraygrenzen-Überprüfung
 - keine Freigabe von noch-referenzierten Objekten
 - eingeschränkter Zugriff auf Systemschnittstelle

- MSIL-Code wird vor der ersten Ausführung überprüft (Verifier) und in Maschinencode übersetzt (JIT-Compiler)

- CLR wird für beim Laden von .NET-Anwendungen (spezielle .exe/.dll-Dateien) automatisch gestartet
 - CLR läuft in einem Windows-Prozess ab
 - CLR bearbeitet *Managed Code*

6 Assemblies

- Anwendungen (.exe) bzw. Anwendungskomponenten (.dll) für .NET
 - Portable Execution (PE) Files

- Metadaten (Manifest) zur Selbstbeschreibung
 - Name, Version, Authentisierungsinformation
 - enthaltene Dateien, Typinformationen
 - Liste referenzierter Assemblies

- statische Assemblies

- dynamische Assemblies
 - ◆ zur Laufzeit von anderen .NET-Anwendungen erzeugte Assemblies
 - z. B. zur Template-basierten Codegenerierung in Entwicklungsumgebungen

7 Virtual Execution System — VES

- Teil der CLR
- Class Loader
- Verifier
- Domain-Konzept
 - Kapselung der Ausführung von Assemblies
- Thread-Unterstützung
- Sicherheits-Management
- Garbage Collection
- Metaprogrammierung / Reflexion

8 Unmanaged / Unsafe Code

- Aufruf von Komponenten / Bibliotheksfunktionen / COM-Objekten, die nicht in MSIL vorliegen
 - ◆ Werden nicht von der CLR verwaltet: *Unmanaged Code*
 - ◆ Freigabe der Ressourcen bleibt dem Aufrufer überlassen (kein Garbage Collector)

- Verwendung von Zeigeroperationen: *Unsafe Code*
 - ◆ Direkter Zugriff auf den Speicher, keine Garbage Collection
 - ◆ Code muss als "unsafe" gekennzeichnet werden

9 Application Domains

- eine .NET-CLR wird in einem virtuellen Adressraum ausgeführt
 - keine Hardware-Kapselung innerhalb der CLR
- Software-Kapselung der Ausführung von Assemblies
 - kein direkter Zugriff auf Assemblies anderer Domains möglich
 - CLR verhindert Übergabe von Objekt-Referenzen zwischen Domains
 - Interaktion nur über speziell gesicherten Inter-Domain-Kommunikationsmechanismus möglich
- eigene Domains werden erzeugt wenn
 - Code mit anderen Sicherheits-Einstellungen/Zugriffsrechten geladen wird
 - Isolation zwischen Code-Teilen explizit gewünscht wird
 - Code unabhängig voneinander terminieren können soll
- Terminierung von Domains
 - kontrollierte Freigabe aller belegten Ressourcen

9 Application Domains (2)

- CLR startet mit einer *Default Application Domain*
- Default Application Domain lädt *Hosting Code*
 - baut Umgebung für eine Anwendung auf
 - erzeugt weitere Application Domains
- Weitere Application Domains führen *User Code* aus
 - *User Code* enthält die eigentliche Anwendungslogik
 - Ist in Form von Assemblies geladen
 - In der Regel mehrere Assemblies pro Application Domain
- Code Access Security
 - Legt Rechte basierend auf Code-Charakteristiken (Evidence) fest
 - Evidence = Quelle des Codes (lokal, remote) oder Signaturen
- Role-based Security
 - Rechte basierend auf aktuellem Benutzer (z. B. login-Information)

10 Programmierung verteilter Systeme

- Verteilungskonfiguration
 - ◆ Fernaufruf: Fernverweis auf Objekt wird übergeben (call by reference)
 - ◆ Serialisierung: Objektkopie wird übergeben (call by value)

- Objekterzeugung:
 - ◆ Bekanntmachung eines öffentlichen Objekts (lokale Erzeugung)
 - ◆ Fernerzeugung eines öffentlichen Objekts (Erzeugung wird lokal vorbereitet, aber erst durchgeführt, wenn ein Client darauf zugreift)
 - ◆ Fernerzeugung eines privaten Objekts

- Proxy und Skeleton werden implizit zur Laufzeit aus Metadaten generiert

- XML-Konfigurationsdatei legt Verteilung fest

10 Programmierung verteilter Systeme

- Unterstützung verschiedener Transportprotokolle
 - ◆ z.B. TCP oder HTTP, Binärformat oder XML
- Eingriffsmöglichkeiten in das Nachrichtenformat (*Formatters*) und den Transport von Nachrichten (*Sinks, Channels*)
 - ◆ Verschlüsselung, Kompression, Mitführen von Zusatzinformationen
- "Leasing Distributed Garbage Collector"
 - ◆ Objektreferenz ist nur für eine bestimmte Zeit gültig
 - ◆ Lease wird durch Fernaufruf verlängert
 - ◆ kein Referenzzähler oder periodisches Pingen notwendig (DCOM)