Entwurf Verteilter Systeme

Prof. Dr.-Ing. Martin Gaedke
Technische Universität Chemnitz
Fakultät für Informatik
Professur Verteilte und selbstorganisierende Rechnersysteme

http://vsr.informatik.tu-chemnitz.de

RPC – Ex.: List over Net. transp.

Client: mylist.c

```
#include <stdio.h>
#include <rpc/rpc.h>
#include "mylist.h"


main(int argc, char *argv[])
{
  int dummy;
  int c = 5;
  node_p ptr;
  CLIENT *cl;

  if (argc != 2) {
    fprintf(stderr, "Benutzung : %s <host>\n", argv[0]);
    exit(-1);
  }

  if (!(cl = clnt_create(argv[1], MYLIST, BASVERS, "tcp"))) {
    clnt_pcreateerror(argv[1]);
    exit(-1);
  }

  create_nodes_1(&c, cl);
  ptr = *get_list_1(&dummy, cl);
  while(ptr) {
    printf("%d, ", ptr->val);
    ptr = ptr->next;
  }
  putchar('\n');
}
```

Server: mylist_svc_proc.c

```
#include <rpc/rpc.h>
#include "mylist.h“

static node_p anc = NULL;

void * create_nodes_1_svc(int *c, struct svc_req *r) {
  static int dummy;
  node_p ptr;
  int i;

  for (i = *c - 1; i >= 0; i--) {
    ptr = (node_p) malloc(sizeof(struct node));
    ptr->val = i;
    ptr->next = anc;
    anc = ptr;
  }
  return &dummy;
}


node_p * get_list_1_svc(void *dummy, struct svc_req *req) {
  return &anc;
}
```

List Transmission

Translation:

rpcgen mylist.x *erzeugt: mylist.h*
mylist_clnt.c (Client-Stub)
mylist_svc.c (Server-Stub)
mylist_xdr.c (eXternal Data Representation)

cc -o mylist mylist.c mylist_clnt.c mylist_xdr.c *erzeugt: mylist (Client)*

cc -o mylist_svc mylist_svc_proc.c mylist_xdr.c *erzeugt: mylist_svc (Server)*

Execution:

Server: sudo ./mylist_svc

Client: ./mylist localhost

Result:

0, 1, 2, 3, 4,

Demo RPC-Programmierung

- Use: IDL
- IDL-Generator (IDL "fkt")
  - → Client Stub
  - → Header
- Client
  - Code
- Compile
  - ClientStub
  - Header
  - Code
  - RPC-Runtime
  - → Client-Anwendung

- Create: IDL "fkt"
- IDL-Generator (IDL "fkt")
  - → ServerStub
  - → Header
- Server
  - Code
  - Manager (eigentlichen Funktionen)
- Compile
  - ServerStub
  - Header
  - Code & Manager
  - RPC-Runtime
  - → Server-Anwendung

RPC Final Remarks (1)

- RPC works well with small messages
  - What does it mean for multimedia data?
  - → Paradigm change / use of another mechanism (such as Sockets, HTTP)
- RPC mechanism is synchronous
  - Optimization: **Asynchronous RPC**
- Ways of realizing asynchronous RPCs
  - *Event Handle*: Function handle is passed to the calling function, such that an event can be triggered upon the arrival of server's response
  - *Future Object*: Function returns an object ("Future"), which can then be tested for usability

RPC Final Remarks (2)

- RPC - functional foundation for lots of other middleware approaches/functionality, such as
  - Object-based middleware (often in research context)
  - Remote Method Invocation (RMI) bei Java
  - Component-oriented middleware
  - Problems of those approaches: Firewalls, i.e. no cross-organizational use possible
- RPC programming problems with data alignment and types
  - **Marshaling** – requires constant encoding/decoding in transfer syntax
  - Debugging is very complex

XML-RPC

- XML-RPC requests are HTTP POST requests
  - with certain Header requirements
  - with an XML message in the body to invoke remote procedures

- Header requirements
  - User-Agent and Host must be specified
  - Content-Type text/xml
  - Content-Length specified and correct

XML-RPC

- Payload format
  - *methodCall* element for requests
    - Required *methodName* element
    - Optional *params* element
  - *methodResponse* element for responses
    - Contains either *params* or *fault* element
  - Data types: *int, boolean, string, double, dateTime.iso8601, base64*
  - Structs: *struct* element containing *member* elements containing a *name* and a *value* element
  - Arrays: *array* element containing a *data* element containing *value* elements
  - Faults: *fault* element

XML-RPC example

HTTP-Request

```
POST /XMLRPC HTTP/1.1
Host: vsr.informatik.tu-chemnitz.de
Content-Type: text/xml
Content-length: 160

<?xml version="1.0"?>
<methodCall>
 <methodName>getStudentId
 </methodName>
 <params>
  <param>
   <value><i4>42</i4></value>
  </param>
 </params>
</methodCall>
```

HTTP-Response

```
HTTP/1.1 200 OK
Connection: close
Content-Length: 180
Content-Type: text/xml

<?xml version="1.0"?>
<methodResponse>
 <params>
  <param>
   <value>
    <string>174-8F</string
   </value>
  </param>
 </params>
</methodResponse>
```

Fault HTTP-Response

```
HTTP/1.1 200 OK
Connection: close
Content-Length: 420
Content-Type: text/xml

<?xml version="1.0"?>
<methodResponse>
 <fault>
  <value>
   <struct>
    <member>
     <name>faultCode</name>
     <value><int>418</int></value>
    </member>
    <member>
     <name>faultString</name>
     <value>
      <string>I'm a teapot</string>
     </value>
    </member>
   </struct>
  </value>
 </fault>
</methodResponse>
```

Remote Method Invocation (RMI)

- Invocation of methods of remote Java objects (on a different JVM which may run on a different host)
- Object oriented approach
- Method parameters, returns and exceptions can be full Java objects
- Object serialization to marshal / unmarshal
- Dynamic code loading (fetching classes from foreign JVMs)
- Java Security features
- Distributed Garbage Collection
- Multithreading

Remote Method Invocation (RMI)

- Remote Interface
  - Declares methods of remote objects
  - Extends java.rmi.Remote
  - Each method declares java.rmi.RemoteException in its throw clause
- Remote object
  - Implements a remote interface
  - Represented to the requestor by a stub (remote reference)
- RMI registry
  - Servers registers remote objects with RMI registry with unique names
  - Clients lookup remote object at RMI registry by name

RMI-Server example

- Here is the remote interface that defines the methods the client can invoke on the server:

```
import java.rmi.*;
public interface SampleServer extends Remote {
   Policy getPolicy() throws RemoteException;
   void submitQuery(Query q)
   throws RemoteException,
          InvalidQueryException;
}
```

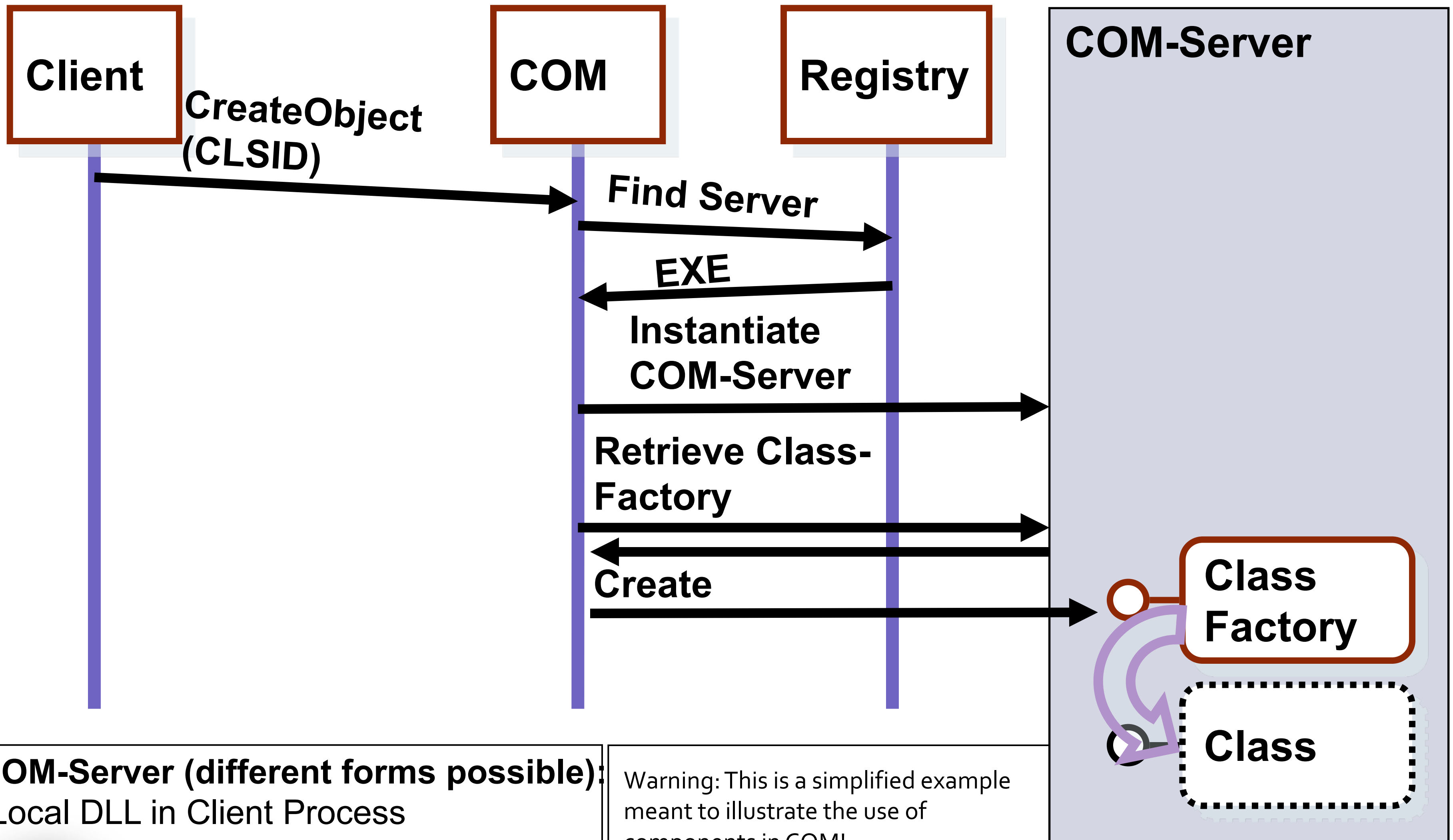
Chapter 3

COMPONENT-BASED MIDDLEWARE

Microsoft COM

- Component Object Model (COM)
  - Microsoft's former standard Component-Technology
  - Developed out of OLE experience
- Components are referred to as so-called COM Server
  - The offered functionality is referred to as COM Class
- Registry – Component database, saves components' location and meta data
  - Components must be registered in the registry
  - Components can be found through the registry
- COM Class
  - Code implements COM interface
  - Has a unique Id, so-called CLSID. Based on UUID
- COM Servers
  - Different types:
  - In-Process Server (DLL is loaded in the Client Process)
  - Out-of-Process Server (.Exe-File, is executed on the Client- or a remote machine (DCOM, COM+) with communication over RPC

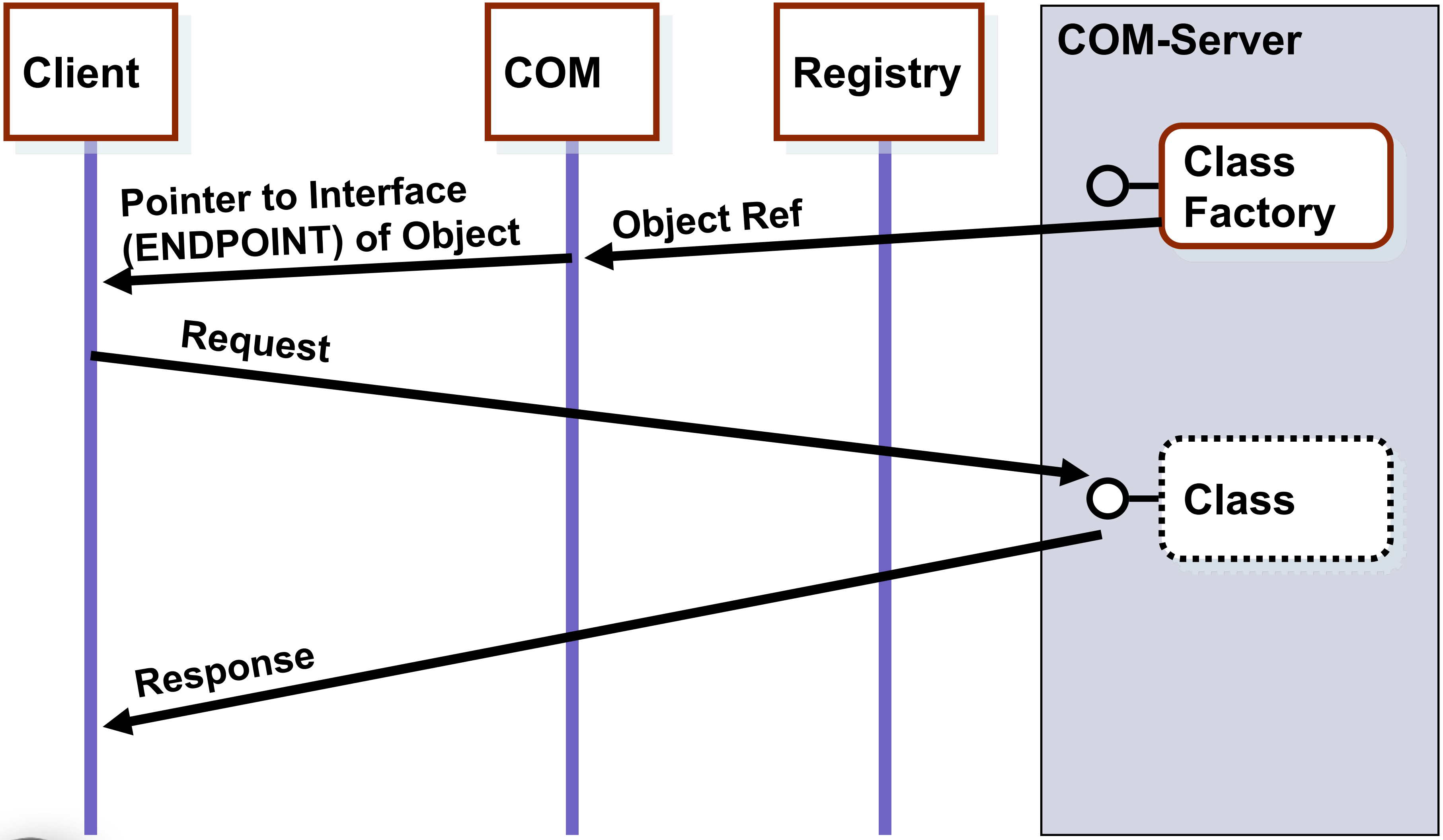
COM - Components, Example (1)



COM-Server (different forms possible):
- Local DLL in Client Process
- Remote DLL in Proxy Process

Warning: This is a simplified example meant to illustrate the use of components in COM!

COM - Components, Example (2)



COM Final Remarks

- Very important component technology
  - Runs on every Windows system (since NT, Win95)
  - Many improvements compared to standard RPC
  - Disadvantage: Proprietary, div. additions DCOM, COM+
- Endpoint in COM:
  - COM Interface defines the contract
  - Contract consists of a set of methods and properties
  - Single component can offer multiple interfaces, but at least the IUnknown interface
  - IUnknown interface for component use/management: QueryInterface, AddRef, Release
- DEMO: Windows Registry

Final Remarks

- Further approaches:
- CORBA – Common Object Request Broker Architecture
  - OMG specification for interoperability between distributed computer systems
  - ORB: Middleware, which realizes a Requestor-Provider relationship
  - CORBA – Problem: Inter-ORB incompatibilities require constraints
- Java Beans and Enterprise Java Beans
  - Based on the Virtual Machine principle
  - Use of principles similar to RMI
- Microsoft .Net Assemblies
  - Very interesting and powerful platform
  - Many modern concepts and very good class structure
- Other platforms partially adopt the approach (for example, Mono)

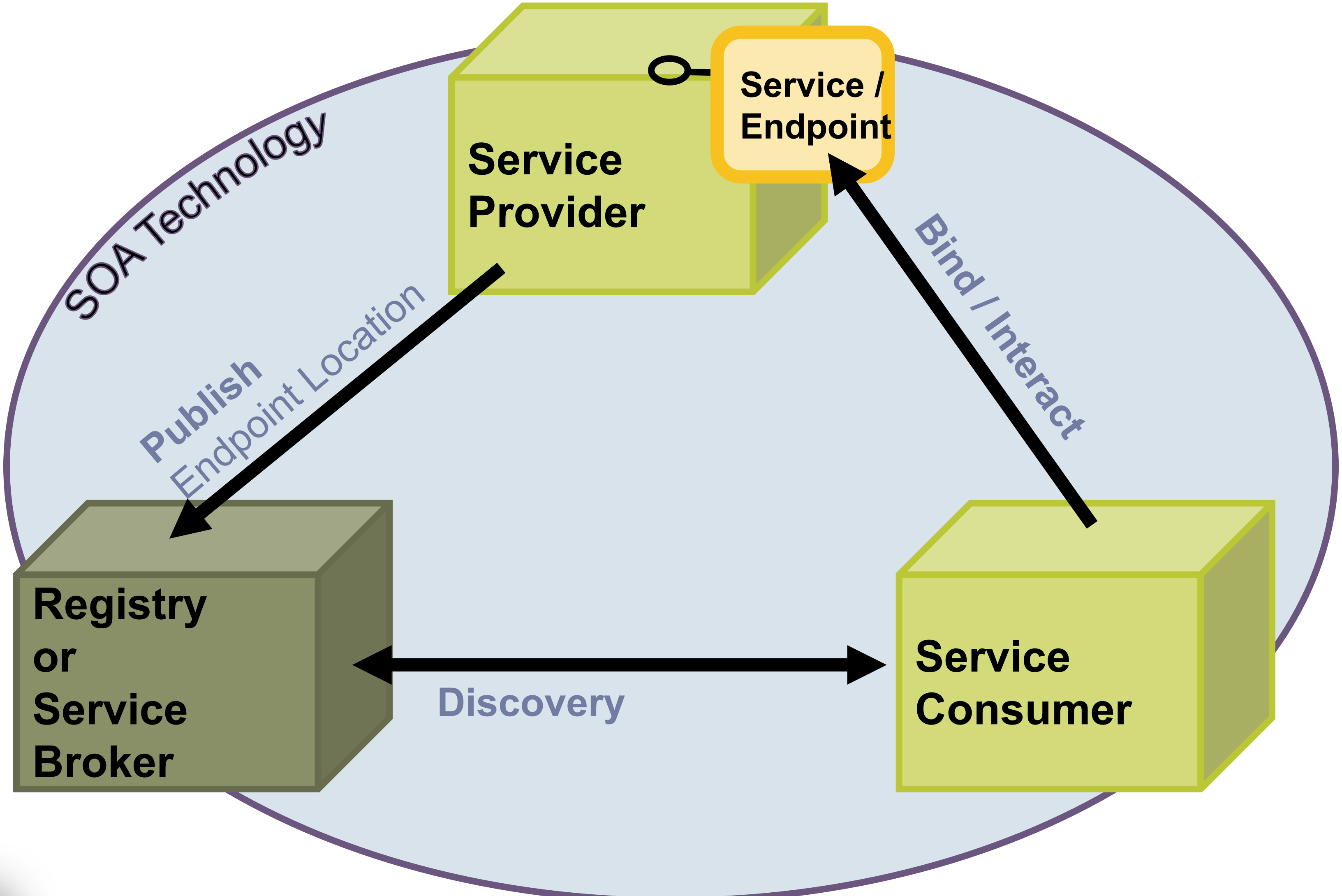
Final Remarks II

- OSGi Service Platform – formerly OSGi = Open Services Gateway initiative
  - OSGi Alliance Specification of a dynamic service platform and component model allowing dynamic deployment and management of services targeting various hardware platforms such as PCs, consumer electronics, cars, mobile phones etc.
  - Requires a Java Virtual Machine
  - *Bundles*: downloadable OSGi components
  - *Layers*: define handling of bundles
    - Security: Code Authentication (X.509), Permissions (File, Property, Metadata,…)
    - Module: Bundle configuration, dependencies, dynamic imports
    - Life Cycle: Start, Stop, Install, Update & Uninstall bundles
    - Service: Deployment & Communication of service bundles
      - Service interfaces to register with the Framework service registry

Chapter 4

SERVICE-ORIENTED PROGRAMMING (FIRST ASPECTS)

Programming with Components as Services



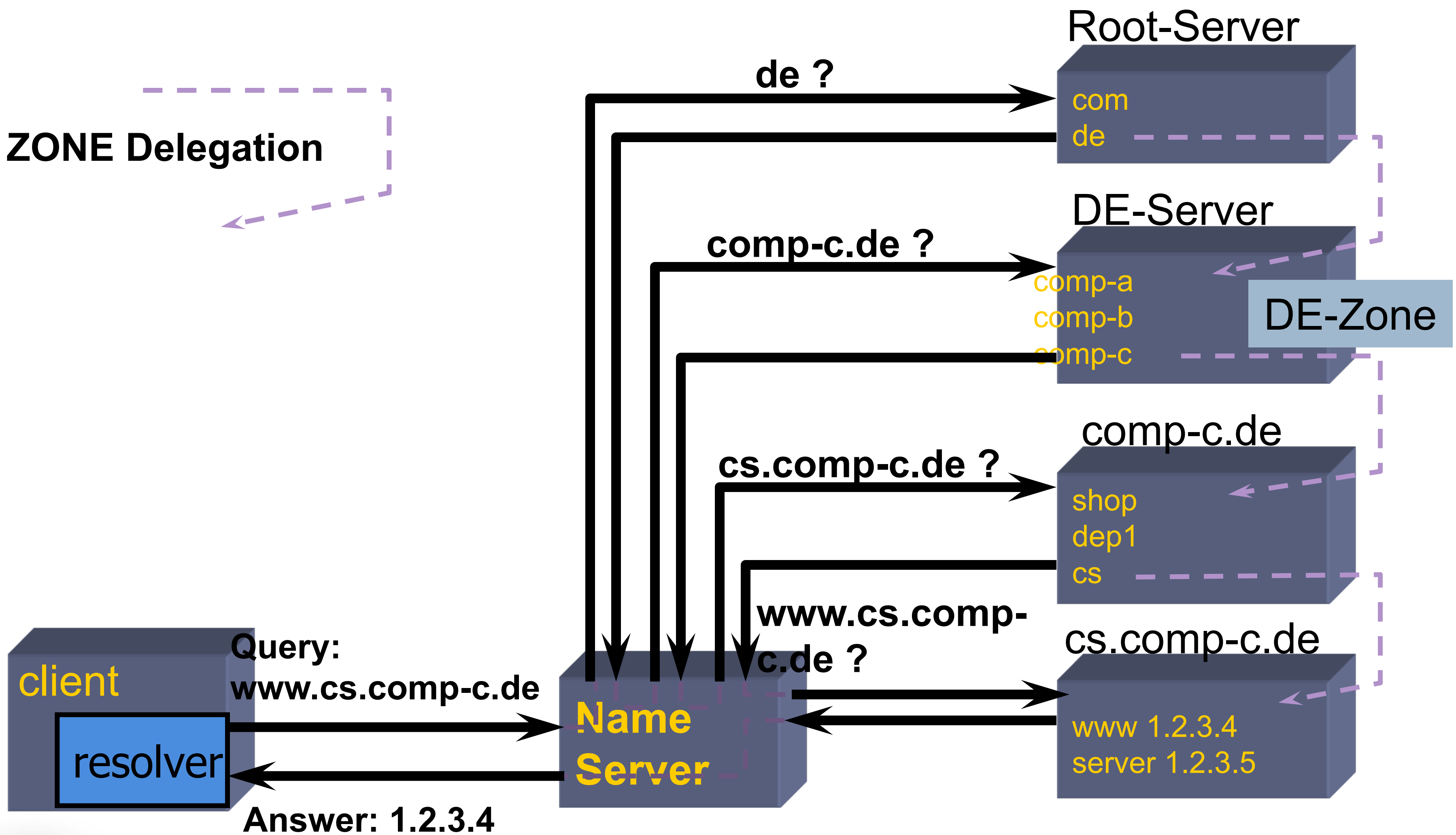
Domain Name System (DNS) as Registry

- A core registry that we use every day: DNS:
  - Where is the Web-Site, FTP-Server, Mail-Server etc.? DNS-Registry tells you where to go
  - Homework: Check how registering your service works in real-life!

Domain Name System (DNS) as Registry

- **DNS** – Idea: User-friendly domain names
  - RFCs 1034 and 1035
  - Map Names to IP Addresses
  - E.g. vsr.informatik.tu-chemnitz.de
- **Zone** - Part of the Domain Name Space provided by a Name-Server (DNS-Server)
  - DNS Database often called Zone File

DNS host name resolution



Example: Part of a Zone File

- Example:

  ; Name Server …
  ; Host addresses
  localhost. IN A 127.0.0.1
  www.mysite.edu. IN A 1.2.3.4

  ; Multi-homed hosts (Remember ROUND ROBIN!)
  ; different computers hosting the same service
  hightraffic-service.mysite.edu. IN A 1.2.3.6
  hightraffic-service.mysite.edu. IN A 1.2.3.7
  hightraffic-service.mysite.edu. IN A 1.2.3.8
  ; Aliases (Remember HTTP-Server)
  service.mysite.edu. IN CNAME service-v1.mysite.edu

Chapter 5

OTHER PROGRAMMING APPROACHES

Programming with P2P

- numerous networked computers instead of some powerful server computers
- Peers are both Service Providers and Service Consumers
- Ad-hoc connections between Peers
- Pure / Hybrid P2P networks
- Considerations
  - Node identification / addressing / discovery
  - Communication protocol + messages
  - Security
  - Distributed data + search

.NET Peer Channel

- Peer Node
- Peer Mesh
  - Peer node graph with unique ID
  - Publication of peer endpoint information for discovery
  - Optimized Connections
- Peer Channel Security
  - Message authenticity, integrity, encryption
  - X.509 certificates, TLS connections
- Peer Resolvers
  - Mesh ID > IP addresses of peer nodes
  - Peer Name Resolution Protocol (PNRP)

Programming with Grids

- Message Passing Interface (MPI)
- De facto standard for language independent communication between parallel processes
- Supports both networked (e.g. via TCP/IP) and dedicated parallel computers (e.g. via InfiniBand)
- Blocking and non-blocking communications
- Process groups connected by Communicators
- Point-to-point communication
  - explicit communication between two processes
- Collective communication
  - e.g. broadcast
- One-sided communication
  - introduced in MPI2
  - Put, Get, Accumulate