

## Softwaretechnik

### Middleware

Stephan Arlt

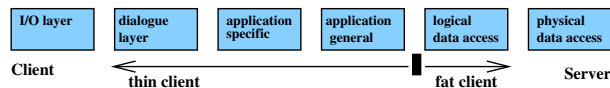
University of Freiburg

SS 2011

### Basic choices

- Architecture
  - ▶ Client/Server architecture
  - ▶ Web-Architecture
- Middleware
  - ▶ communication between program components
  - ▶ requirements
    - ★ language independence
    - ★ platform independence
    - ★ location independence
- Security

## Client/Server Architecture



- Application divided in client-part and server-part
- → five possible divisions of standard (six) layer architecture (thin client → fat client)
- characteristics fixed in the requirements (# of users, operating systems, database systems, ...)

**advantages** traceability of user session, special protocols, design influenced by # users

**disadvantages** scalability, distribution of client software, portability

## Web Architecture

- Client: only I/O layer; Server: everything else
- Client requirements: Web browser (user interface)
- Server requirements:
  - ▶ Web server (distribution of documents, communication with application)
  - ▶ Application server (application-specific and application-general objects)
  - ▶ Database server (persistent data)

**advantages** scalability (very high number of users, in particular with replicated servers), maintainability (standard components), no software distribution required

**disadvantages** restriction to HTTP, stateless and connectionless protocol requires implementation of session management, different Web browsers need to be supported (Internet Programming)

Recent technology addresses some of the disadvantages: Servlets, ASP, ...

## Refinement: N-tier Architecture

- Physical deployment follows the logical division into layers (tiers)
- Why?
  - ▶ separation of concerns (avoids e.g. mixing of presentation logic and business logic)
  - ▶ scalability
  - ▶ standardized frameworks (e.g., Java 2 Enterprise Edition, J2EE) handle issues like security and multithreading automatically
- Example (J2EE):
  - ▶ Presentation: Web browser
  - ▶ Presentation logic: Web server (JSP/servlets or XML/XSLT)
  - ▶ Business logic: Session EJBs (Enterprise Java Beans)
  - ▶ Data access: Entity EJBs
  - ▶ Backend integration (legacy systems, dbms, distributed objects)

## Enterprise JavaBeans (EJB): Goals

- A SPECIFICATION! but implementations are available
- standard component architecture for business applications in Java <sup>1</sup>
- defines interaction of components with their container <sup>2</sup>
- development, deployment, and use of web services
- abstraction from low-level APIs
- deployment on multiple platforms without recompilation
- interoperability
- components developed by different vendors
- part of Java 2 Platform, Enterprise Edition (J2EE)
- compatible with other Java APIs
- compatible with CORBA protocols

<sup>1</sup>→ main target: business logic, between UI and DBMS

<sup>2</sup>directory services, transaction management, security, resource pooling, fault tolerance

## Middleware / Components / Communication infrastructure Sockets

### Connection of resources in Client/Server architecture

- 1 Sockets (TCP/IP, ...)
- 2 RPC
- 3 RMI
- 4 SOAP (Simple Object Access Protocol)/Web Services
- 5 .NET
- 6 COM, COM+ (Distributed Component Object Model)
- 7 CORBA (Common Object Request Broker Architecture)

Items 6 and 7 are software component models

- software terminal of a network connection (a data structure)
- two modes of communication to host
  - ▶ reliable, bidirectional communication stream or
  - ▶ unreliable, unidirectional one-shot message
- local variant: inter-process communication (IPC)
- low level:
  - ▶ manipulation of octet-streams required
  - ▶ custom protocols

```

ServerSocket serverSocket = new ServerSocket(1234);
while ( true ) {
    Socket client = serverSocket.accept();
    InputStream input = client.getInputStream();
    OutputStream output = client.getOutputStream();
    int value1 = input.read();
    int value2 = input.read();
    output.write(value1 + value2);
    input.close();
    output.close();
}

```

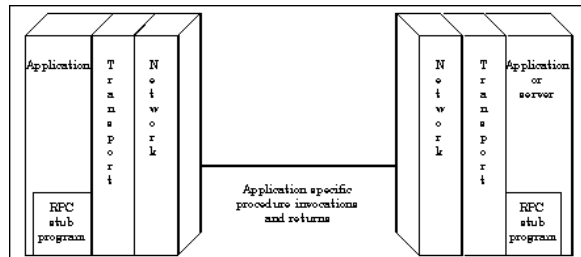
```

Socket server = new Socket("localhost", 1234);
InputStream input = server.getInputStream();
OutputStream output = server.getOutputStream();
output.write(1);
output.write(2);
int result = input.read();
input.close();
output.close();

```

## Remote Procedure Call (RPC)

- procedure call across process and system boundaries (heterogeneous)
- transparent to client code, but some specialities
  - ▶ Error handling: failures of the remote server or network
  - ▶ No global variables or side-effects
  - ▶ Performance: RPC usually one or more orders of magnitude slower
  - ▶ Authentication: may be necessary for RPC



## Anatomy of RPC

- define the interface in terms of XDR
  - XDR is a language for describing eXternal Data Representation
  - XDR is independent of a particular host language (network format)
  - host language data has to be marshalled<sup>3</sup> to and from XDR
- stub functions for each remotely callable procedure
  - client code is written in terms of calls to client stubs
  - server code is called from server stubs
- stub functions generated by RPC compiler from interface definition

<sup>3</sup>data marshalling = transferring data to a network buffer and conversion to external representation; synonyms: serialization, pickling

## Timeline of an RPC

time	client stub		server stub
↓	marshall parameters to XDR		
	connect to server	→	invoked by incoming connection
	transmit parameters	→	receive parameters
	<b>wait for server response</b>		unmarshall parameters
			call actual implementation
			marshall results
	receive results	←	transmit results
	unmarshall results from XDR		exit

- object-oriented RPC
- specific to Java
- implements method calls
  - ▶ dynamic dispatch
  - ▶ access to object identity (`self`, `this`)
- object serialization (marshalling)
- access via interfaces
- easy to use
- latest variant: asynchronous method invocation
- *"Experience has shown that the use of RMI can require significant programmer effort and the writing of extra source code"*

Douglas Lyon: "Asynchronous RMI for CentiJ", in Journal of Object Technology, vol. 3, no. 3, March-April 2004, pp. 49-64. [http://www.jot.fm/issues/issue\\_2004\\_03/column5](http://www.jot.fm/issues/issue_2004_03/column5)

## Simple Object Access Protocol (SOAP)

- protocol specification for invoking methods
- based on HTTP plus extensions <sup>4</sup>
- encodes information using XML / XML Schema <sup>5</sup>

```
POST /StockQuote HTTP/1.1
Host: www.stockquoteserver.com
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn
SOAPAction: "Some-URI"
```

```
<SOAP-ENV:Envelope ...>
  <SOAP-ENV:Body>
    <m:GetLastTradePrice xmlns:m="Some-URI">
      <symbol>DIS</symbol>
    </m:GetLastTradePrice>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

## SOAP example: travel agent ⇒ tour operator

```
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
  <env:Header>
    <m:reservation xmlns:m="http://travelcompany.example.org/reservation"
      env:role="http://www.w3.org/2003/05/soap-envelope/role/next"
      env:mustUnderstand="true">
      <m:reference>uuid:093a2da1-q345-739r-ba5d-pqff98fe8j7d</m:reference>
      <m:dateAndTime>2003-11-29T13:20:00.000-05:00</m:dateAndTime>
    </m:reservation>
    <n:passenger xmlns:n="http://mycompany.example.com/employees"
      env:role="http://www.w3.org/2003/05/soap-envelope/role/next"
      env:mustUnderstand="true">
      <n:name>Marilyn Manson</n:name>
    </n:passenger>
  </env:Header>
```

<sup>4</sup>reason: internet security, firewalls

<sup>5</sup>reason: standard, extensibility

```

<env:Body>
  <p:itinerary xmlns:p="http://travelcompany.example.org/reservation/travel">
    <p:departure>
      <p:departing>New York</p:departing>
      <p:arriving>Los Angeles</p:arriving>
      <p:departureDate>2003-12-14</p:departureDate>
      <p:departureTime>late afternoon</p:departureTime>
      <p:seatPreference>aisle</p:seatPreference>
    </p:departure>
    <p:return>
      <p:departing>Los Angeles</p:departing>
      <p:arriving>New York</p:arriving>
      <p:departureDate>2003-12-20</p:departureDate>
      <p:departureTime>mid-morning</p:departureTime>
      <p:seatPreference/>
    </p:return>
  </p:itinerary>
  <q:lodging xmlns:q="http://travelcompany.example.org/reservation/hotels">
    <q:preference>none</q:preference>
  </q:lodging>
</env:Body>
</env:Envelope>

```

- Web Service Description Language
- XML based
- describes location and protocol of the service
- main elements (XML):

**portType** operations of service (cf. RPC program)  
**message** spezifikation of parameters  
**types** data types (XML Schema)  
**binding** message format and protocol

## WSDL Example

```

<message name="getTermRequest">
  <part name="term" type="xs:string"/>
</message>

<message name="getTermResponse">
  <part name="value" type="xs:string"/>
</message>

<portType name="glossaryTerms">
  <operation name="getTerm">
    <input message="getTermRequest"/>
    <output message="getTermResponse"/>
  </operation>
</portType>

```

- xs is the namespace for XML Schema definitions  
 xmlns:xs="http://www.w3.org/2001/XMLSchema"

## WSDL example: one-way-operation

```

<message name="newTermValues">
  <part name="term" type="xs:string"/>
  <part name="value" type="xs:string"/>
</message>

<portType name="glossaryTerms">
  <operation name="setTerm">
    <input name="newTerm" message="newTermValues"/>
  </operation>
</portType>

```

- no return value ⇒ no answer message

## Further Kinds of Operation

- output-only (no <input>), Ex:

```
<message name="whatTimeValue"/>
<message name="theTimeValue">
  <part name="time" type="xs:date"/>
</message>
<portType name="Date">
  <operation name="currentTime">
    <input name="whatTime" message="whatTimeValue"/>
    <output name="theTime" message="theTimeValue"/>
  </operation>
</portType>
```

- "Notification": output without request

## SOAP Binding

```
<portType name="glossaryTerms">
  <operation name="getTerm">
    ...
</portType>
<binding type="glossaryTerms" name="b0">
  <soap:binding style="document"
    transport="http://schemas.xmlsoap.org/soap/http" />
  <operation>
    <soap:operation soapAction="http://example.com/getTerm"/>
    <input>
      <soap:body use="literal"/>
    </input>
    <output>
      <soap:body use="literal"/>
    </output>
  </operation>
</binding>
```

- soap is SOAP's namespace
- style ∈ rpc, document
- transport defines base protocol (HTTP)

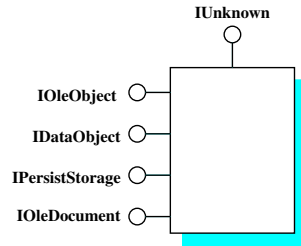
## Automatic generation of WSDL code

- translation from WSDL to a client API is tedious
- parsing XML
- verifying XML Schema
- choice of data types
- correct SOAP messages

⇒ tools: WSDL2Java

## Distributed Component Object Model (DCOM)

- proprietary format for communication between objects
- binary standard (not language specific) for "components"
- COM object implements interfaces (at least one)
  - ▶ described by IDL (interface definition language); stubs etc. directly generated by tools
  - ▶ immutable and persistent
  - ▶ may be queried dynamically
- COM services
  - ▶ uniform data transfer IDataObject (clipboards, drag-n-drop, files, streams, etc)
  - ▶ dispatch interfaces IDispatch combine all methods of a regular interface into one method
  - ▶ outgoing interfaces (required interfaces, female connector)

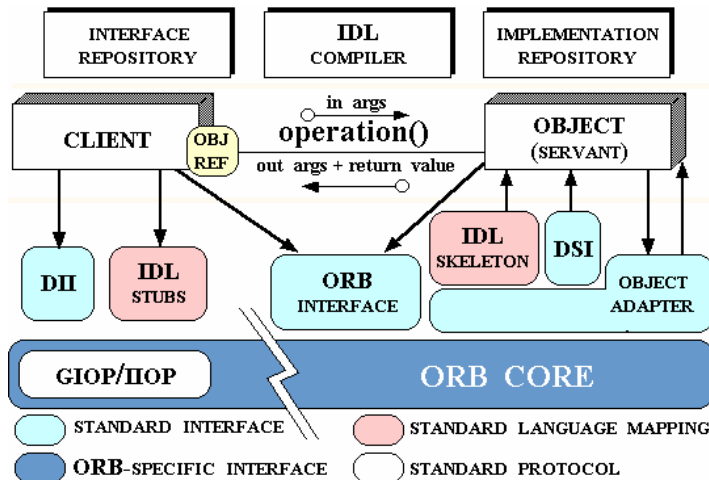


```
[ uuid(00000000-0000-0000-C000-000000000046) ]
interface IUnknown {
    HRESULT QueryInterface ([in] const IID requestedIID,
                           [out, iid_is(requestedIID)] void **ppvObject);

    ULONG AddRef ( );
    ULONG Release ( );
}
```

- emerging open distributed object computing infrastructure
- specified by OMG (Object Management Group)
- manages common network programming tasks
  - ▶ object registration, location, and activation;
  - ▶ request demultiplexing;
  - ▶ framing and error-handling;
  - ▶ parameter marshalling and demarshalling; and
  - ▶ operation dispatching
- extra services
  - component model reminiscent of EJB

CORBA ORB Architecture



Summary

- Distributed Systems Architecture
  - ▶ client/server
  - ▶ web
  - ▶ n-tier (J2EE)
- Middleware
  - ▶ communication infrastructure
  - ▶ component frameworks