Softwaretechnik

Middleware

Manuel Geffken

University of Freiburg, Germany

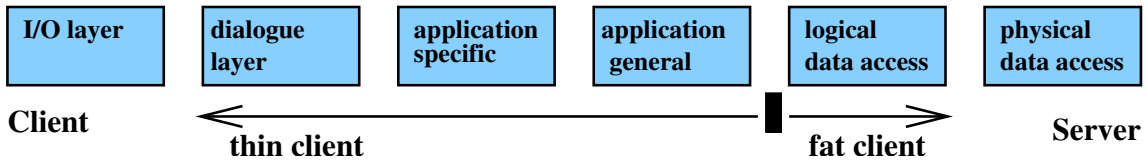
SS 2012

Distributed Applications

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
- $\longrightarrow$ Five possible divisions of standard (six) layer architecture (thin client $\longrightarrow$ 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: Java Persistence API
  - Backend integration (legacy systems, DBMS, distributed objects)

Enterprise JavaBeans (EJB): Goals

- Part of Java Platform, Enterprise Edition (J2EE)
- A SPECIFICATION! but implementations are available
- Server-side component architecture for enterprise applications in Java [1]
- Defines interaction of components with their container [2]
- Development, deployment, and use of web services
- Abstraction from low-level APIs
- Deployment on multiple platforms without recompilation
- Interoperability
- Components developed by different vendors
- Compatible with other Java APIs
- Compatible with CORBA protocols

---

[1] $\longrightarrow$ main target: business logic, between UI and DBMS

[2] directory services, transaction management, security, resource pooling, fault tolerance

Middleware / Components / Communication infrastructure

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 **??** and **??** are software component models

Sockets

- 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

Sockets in Java

Server

```
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();
}
```

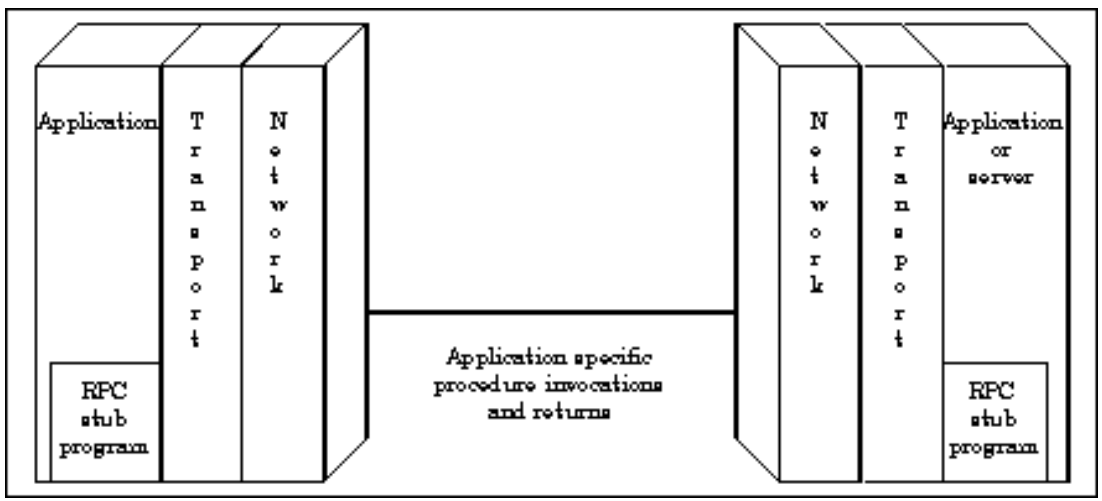
Sockets in Java

Client

```
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 interface in terms of XDR (e**X**ternal **D**ata **R**epresentation)
  - XDR is a data serialization format
  - XDR is independent of a particular host language (network format)
  - Host language data has to be marshalled[3] 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

[3] 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
	wait for server response		unmarshall parameters
			call actual implementation
			marshall results
	receive results	←	transmit results
	unmarshall results from XDR		exit

Remote Method Invocation (RMI)

- Object-oriented RPC
- Specific to Java
- Implements method calls
  - Dynamic dispatch
  - Access to object identity (`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`

Simple Object Access Protocol (SOAP)

- Protocol specification for invoking methods
- Based on HTTP plus extensions [4]
- Encodes information using XML / XML Schema [5]

```
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>
```

[4] reason: internet security, firewalls

[5] reason: standard, extensibility, can be validated

SOAP example: travel agent $\Rightarrow$ 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>
```

```
  <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>
```

WSDL

- Web Service Description Language
- XML-based
- Describes location and protocol of the service
- Main elements:

portType	Operations of service (cf. RPC program)
message	Spezification of parameters
types	Data types (XML Schema)
binding	Message format and protocol

WSDL 1.1 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 $\Rightarrow$ no answer message

Further Kinds of Operation

- output-only (no <input> params), Example:

```
<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

Binding to SOAP

```
<portType name="glossaryTerms">
  <operation name="getTerm">

...

<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 $\in$ {rpc, document}
- transport defines base protocol (HTTP)

Automatic generation of WSDL code

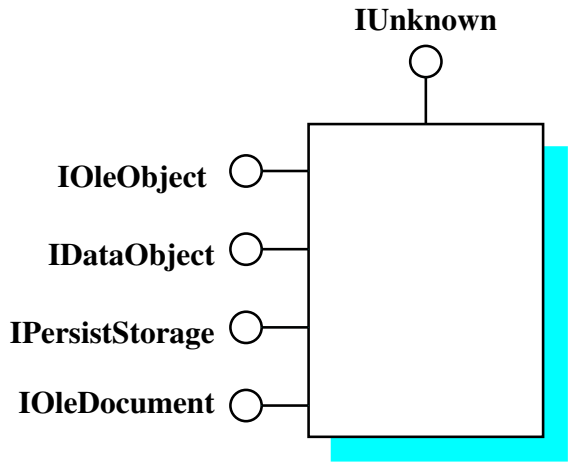
- Translation from WDSL to a client API is tedious
- Parsing XML
- Verifying XML Schema
- Choice of data types
- Correct SOAP messages

$\Rightarrow$ Tools: WSDL2Java

Distributed Component Object Model (DCOM)

- Proprietary format for communication between objects
- Binary standard (not language specific) for "components"
- COM object implements one or more interfaces
  - Described by IDL (**I**nterface **D**efinition **L**anguage); 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 (RTTI)
  - Outgoing interfaces (required interfaces, female connector)

Example: COM

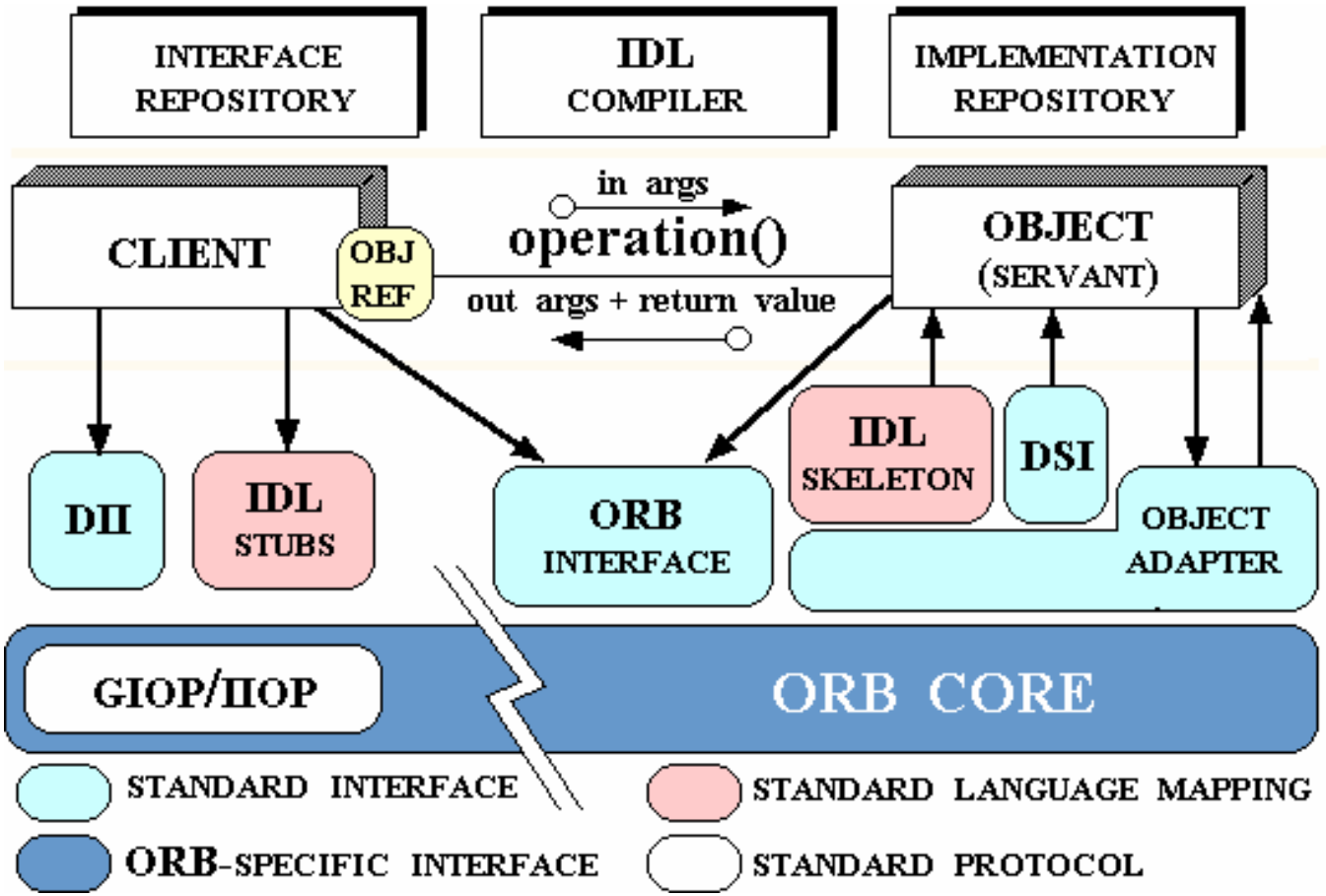



```
[ uuid(00000000-0000-0000-C000-000000000046) ]
interface IUnknown {
  HRESULT QueryInterface ([in] const IID requestedIID,
                          [out, iid_is(requestedIID)] void **ppvObject);
  ULONG AddRef ( );
  ULONG Release ( );
}
```

Common Object Request Broker Architecture (CORBA)

- Emerging open distributed object computing infrastructure
- Specified by OMG (Object Management Group)
- Manages common network programming tasks
  - Cross-Language: Normalizes the method-call semantics
  - Parameter marshalling and demarshalling
  - Object registration, location, and activation
  - Request demultiplexing
  - Framing and error-handling
- 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