3. RPC System Examples, Messaging and Stream-based Communication
Outline

- RPC System Examples
  - HTTP/AJAX
  - Web Services

- Messaging
  - Principles
  - System Example

- Stream-based Communication
  - Principles
Synchronous HTTP

- **Click** = User action
- **Wait** = HTTP Request to web server -> Request processing -> HTTP Response with complete (X)HTML page to Browser
- **Refresh** = Browser refreshes the screen and displays the new (X)HTML page
Synchronous HTTP

- Synchronous with respect to user actions
  - to update the displayed data an HTTP Request is sent
  - user actions are *blocked* until an HTTP Response from the server is received

- High degree of dependence between HTTP Request and HTTP Response
  - for every data transfer from server to client a request has to be sent directly before this transfer

  > Coupling of user activity and server-side processing

- Shortcomings of the classic web application model
  - Slow performance due to “click, wait, and refresh”
  - Loss of operation context during page refresh
  - Excessive server load and bandwidth consumption due to redundant page refreshes

  > Result: slow, unreliable, low productivity, low interactivity and inefficient web applications
Intentions of AJAX:

- Create web applications that look and feel like desktop applications, e.g. GoogleMail or GoogleDocs
- Extend the classic web application with additional features, e.g. auto completion (Google Suggest)
Introduction of intermediary — the AJAX engine

- Eliminates “click, wait, and refresh user interaction paradigm”
- Responsible for rendering UI and communicating with the server

User actions are decoupled from communication with server

Asynchronous JavaScript + XML (AJAX)

- AJAX not a technology itself, rather a combination of several technologies (Definition by J.J. Garret).
  - Standard-based presentation using XHTML and CSS
  - Dynamic display and interaction using the Document Object Model (DOM)
  - Data interchange and manipulation using XML and XSLT
  - Asynchronous data retrieval using XMLHttpRequest object
  - And JavaScript is binding everything together

- Functionality of the AJAX engine
  - User actions generate a JavaScript call to the AJAX engine instead of an HTTP Request
  - Not every response to user actions requires a request to the server; the engine can operate on its own, e.g. simple data validation, editing data in memory, and even some navigation.
  - If the engine needs something from the server in order to respond to a user action, then it sends an HTTP Request to the server to get the needed data for the processing of this response
  - The engine makes those requests asynchronously
AJAX: Non Blocking Model

- Asynchronous with respect to the user actions
  - To update the displayed data the user actions *are not blocked* by waiting for the HTTP Response from the server

- High degree of dependence between HTTP Request and HTTP Response
  - For each server-side processed update a full round trip of request/response between the AJAX engine and the server is needed

- User activity and server-side processing are decoupled
Web Services

- Manufacturer-independent initiative for Web-based services
- Framework for encapsulated, loosely coupled components, which are accessible network-wide over **standard protocols**
  - Communication via **SOAP/XML**
  - Interface Description using **WSDL** (Web Services Description Language)
  - Binding of services via **UDDI** (Universal Description, Discovery and Integration); comparable with Directory Service

- Standardization through W3C and OASIS (Organization for the Advancement of Structured Information Standards)
- Specialization by manufacturers of system solutions (IBM, Microsoft, Oracle, BEA Systems etc.)
Web Services: Infrastructure

Service Index

1. Publication
2. Search
3. Reference to Service
4. Examination of Description
5. Service Use

Service User

Service Provider

UDDI

WSDL

SOAP
Web Services: SOAP

- **Used technologies**
  - Encoding of calls and parameters via XML
  - Realization via HTTP
    - Transferable via firewalls (Use of Port 80)
    - Use of reliability features

- **Security mechanisms**
  - Firewall extensions at the application layer for authentication and authorization of the communication
  - WS-Security: XML Encryption, XML Signature, etc.

+ Independence of special programming languages
+ Support of synchronous and asynchronous interactions (e.g. Message Passing)
  - Less efficient than direct communication due to HTTP overhead
  - No automatic garbage collection
POST /CustomerServer HTTP/
Host: www.onlinetrader.com
Content-Type: application/soap+xml
Content-Length: nnnn
SOAPMethodName: http://example.com/Customer#getCustomerProfile

<?xml version="1.0"?>
<SOAP: Envelope xmlns: SOAP="http://www.w3.org/2001/12/soap-envelope">
  <SOAP: Header>
    <t: Transaction xmlns: t="http://example.com/CustTrans"
      SOAP: mustUnderstand="1">
      328
    </t: Transaction>
  </SOAP: Header>
  <SOAP: Body>
    <m: getCustomerProfile xmlns: m="http://example.com/Customer">
      <CustomerIdentification>
        <CustomerId>jsmith@tu-dresden.de</CustomerId>
        <password>******</password> No password encryption!
      </CustomerIdentification>
    </m: getCustomerProfile>
  </SOAP: Body>
</SOAP: Envelope>
HTTP 200 ok
Connection: close
Content-Type: text/xml
Content-Length: nnnn

<?xml version="1.0"?>
<SOAP: Envelope xmlns:SOAP= "http://www.w3.org/2001/12/soap-envelope">
  <SOAP: Body>
    <m:getCustomerProfile xmlns:m= "http://www.example.org/Customer">
      <return>
        <name>John Smith</name>
        <dob>10/5/1971</dob>
        <telephone>017224466397</telephone>
        <address>2 Somestreet Dresden</address>
        <registeredOn>1/3/2006</registeredOn>
      </return>
    </m:getCustomerProfile>
  </SOAP: Body>
</SOAP: Envelope>
SOAP: Data Type Definition (example)

```xml
<element name="CustomerIdentification">
  <complexType>
    <element name="customerId" type="xsd:string"/>
    <element name="password" type="xsd:string"/>
  </complexType>
</element>
```

- Specification of all essential data types possible
  - Representation of data types of common programming languages
  - e.g. (variable) arrays, enumerations etc.
Web Services Description Language (WSDL)

- Description of interfaces of network services as a set of endpoints operating on messages containing document-oriented or procedure-oriented information.

- Example (shortened):
  ```xml
  <interface name="Productcatalogue">
    <operation name="provideProduct"
      pattern="http://www.w3.org/2006/01/wsdl/in-out">
      <input element="tns:ProductDescription"/>
      <output element="tns:Product"/>
    </operation>
    <operation name="searchProduct">
      ...
    </operation>
  </interface>
  ```

- Call modes: oneway; request-response (Client/Server)
  notification; solicit-response (Server/Client)

- Possibility of automatic generation of interface descriptions from design representations through tools
Web Services: Binding Process

- **UDDI**: Registration of Service URL
- **Client**: Application, Stub Generation
- **Web Server**: Web Service Container, WSDL
- **Application Server**: Service Implementation (e.g., EJB)

**Flow:***
- **Search Query**: URL
- **Registration of Service URL**: UDDI
- **Stub Generation**: Application
- **HTTP Request (URL)**: Web Service Container
- **HTTP Response (WSDL)**: WSDL
- **SOAP**: Web Service Container
- **Application Server**: Service Implementation (e.g., EJB)
Web Services: Summary

- No new technology, but well standardized approach
- Comfortable, web-based call mechanism
- Also applicable via firewalls using SOAP / HTTP
- Enables technology-independent implementation of application logic
- No replacement of server component models (eg. EJB, .NET)
  - Web Services provide access technology from client to server, especially over Internet
Messaging Concept: Publish/Subscribe

- Asynchronous, m:n message-based communication
- Basic communication principle → message passing
  - No confirmation for message receipt (one-way interaction)
- Publish/Subscribe interaction scheme:
  - Supplier/Publisher → creates messages
  - Consumer/Subscriber → receives and processes messages
  - Subscription → flexible binding between supplier and consumer

Consumer

Subscribe
(Channel/Subject/Content)

Supplier

Publish (Message)
Mediation Component (Message Queue)
- Simplification of indirect message exchange - loose coupling
- Persistent in-between storage of messages for reliability
- Secure message forwarding, better error semantics, support of transactions

Asynchronous
- Logical and temporal decoupling of sender and receiver
- ACK messages guarantee, that message has reached the Receiver
- N:M – communication with multiple Suppliers and multiple Consumers
Message Oriented Middleware

- Based on messaging principles
  - Messages, Queues, Message channels
- Decoupling of sender and receiver
  - example: orders sent from purchase order creation component to purchase order processing component
- Dynamic coupling between application and local queues based on logon / logout
- Abstraction level similar to object-oriented approaches

- Products and technologies
  - IBM WebSphere MQ, BEA MessageQ, Tibco, etc.
  - C++ and Java-Support (conformant to JMS)
  - Object Orientation - messages and queues embedded as objects within the programming environment
  - XML (eXtensible Markup Language) for description of transferred content
  - support of essential operating system platforms
MOM – Product Example: WebSphere MQ

- **Features:**
  - Communication with Point-to-Point (1:1), Publish/Subscribe (N:M), Multicast (1:N)
  - Support of transactions, persistence, security and one-time-only delivery
  - Message priorities and filtering
  - Load balancing (via selective delivery) and parallel processing
  - Compatible with JMS, integrates with SOAP, EJB, REST, .NET, supports SOAP
  - Integration of mobile devices (via MQ Telemetry Transport (MQTT))

- **WebSphere MQ Server**
  - Installation of queuing manager, providing queuing services to clients and other queue managers

- **WebSphere MQ Client**
  - Component for communication between application and queue manager
1. Establish connection to queue manager (local or remote) -> MQCONN
   • Authentication performed during connection
2. Open a particular queue object -> MQOPEN
   • Authorization is checked
3. App passes message to queue (=MQPUT)
4. Message is forwarded to machine 2 via the (unidirectional) message channel
5. App gets message via MQGET command
6. Reply (optional) follow via separate queues and channels
**Queue Manager 1**
1. MQOPEN - Message with destination details from
2. Queue name resolution → destination is on remote queue manager
   → message (with header) is placed on transmission queue

**Transmission**
3. Sending MCA retrieves message from queue (including header info) and
4. Sends message to the receiving MCA across the network.

**Queue Manager 2**
5. Receiving MCA removes header and uses info in it to perform an MQOPEN on queue manager 2
6. Queue name resolution determines the destination for the message → local or remote queue
• Java Messaging Service (JMS)
  • Programming interface for clients to access a MOM with different messaging functionalities
  • Possible through standard object oriented interfaces

- Example: Order preprocessing department as sender (Supplier)

  ```java
  Context initialContext = new InitialContext();
  QueueConnectionFactory factory =
    initialContext.lookup("ConnectionFactory");
  QueueConnection connection = factory.createQueueConnection();
  QueueSession session =
    connection.createQueueSession(false, Session.AUTO_ACKNOWLEDGE);
  Queue orderQueue = (Queue) initialContext.lookup("Order");
  QueueSender sender = session.createSender(orderQueue);
  ObjectMessage order = session.createObjectMessage(...);
  sender.send(order);
  ```
Example: Order Execution server as receiver (Consumer)

... 
Queue orderQueue = (Queue) initialContext.lookup("Order");
QueueReceiver receiver = session.createReceiver(orderQueue);
ObjectMessage order = (ObjectMessage) receiver.receive();
...
order.acknowledge(); //Optional confirmation sent back to sender
Advantages
+ Simple manageability
+ Robust message delivery
+ Flexible application fields (for instance load balancing, parallelization, batch-transmission of branch data etc.)
+ Relevant for loose coupling of programs, for instance via Internet, or for Mobile Computing

Disadvantages
- Limited communication semantics
- Interaction model is different than with procedures/method invocations
- Limited accessibility of higher services
Stream Based Communication: Streaming Classes

- Unidirectional
  - On-Demand Stream
  - Live Stream

- Bidirectional
  - Point-to-Point Conversation
  - Multipoint Conference
Stream Based Communication

- **Data Stream**
  - Transfer and presentation of data in a time dependant sequence
  - Time dependent media: Video and Audio

- **Transfer methods:**
  - Asynchronous
    - No time relationship between sending and receiving
    - Data sent as fast as possible
  - Synchronous
    - Packets must comply with maximum lag time
    - Early arriving packets saved until time for display
  - Isochronous
    - Maximal AND Minimum lag time observed
    - Limiting of fluctuation of lag time between packets (jitter)
  - Complex streams
    - Combination of different media within one stream (e.g. video and audio)
    - The different media must be synchronized with each other
Application Classes & Requirements

- **Unidirectional**
  - Stream from server to one or more clients
  - No interaction relationship between sender and receiver
  - On-Demand Streams
    - Buffering possible to compensate for high jitter
  - Live Streams
    - Only limited buffering possible to maintain up-to-dateness

- **Bidirectional**
  - Two way interaction relationship (Sending and receiving by each participant)
  - Timely connection
  - Point-to-Point – two participants
  - Conferencing – more than two participants
  - Stricter requirements
    - Delay and jitter – affects of high values on conversation quality are quickly noticeable
    - Voice and lip synchronisation
Connection Control

- Streams are transferred in a general process regardless of streaming class

- 3 Phases:
  1. Connection establishment
     - Exchange of connection information (IP addresses, Port No.)
     - Negotiation of quality parameters (image resolution, refresh rate, codecs and data formats)
     - Reservation of resources
  2. Usage - Reaction to system changes
     - Transfer of Streams
     - Integration of further communication participants
     - Change in streaming server
     - Switching to lower data quality due to increased network load
  3. Connection Closure
     - Termination of data streams
     - Release of occupied resources
Protocols

- No universal approach for the exchange of multimedia data streams → various protocols available, including: **SIP**, **RTP** and **RTCP**.

- **Session Initiation Protocol (SIP)**
  - Vision: allow every form of Multimedia Communication over IP through a modular, extensible concept
  - Localisation of participants, establishment, control and closing of connections → no transfer of stream data (see RTP)
  - Every SIP component contains a User Agent Client (UAC) and User Agent Server (UAS)
  - User agents facilitate direct communication between two or more partners (Peer-to-Peer protocol)
  - More functionality through extensions, e.g. Redirect Proxy to Forward Calls
Stream Based Communication

- **Real-Time Transport Protocol (RTP)**
  - Transfer of stream data in the form of packets
  - Identification of the media source \(\rightarrow\) source-id
  - Synchronization between sender and receiver \(\rightarrow\) timestamp
  - Placement in correct order \(\rightarrow\) sequence number

- **Real-Time Control Protocol (RTCP)**
  - Dynamic flow control of RTP connection \(\rightarrow\) send control info to sender
  - Information includes current quality of the connection
  - Enables sender to optimise transfer e.g. by changing the codec and image refresh rate

- **Network Services**
  - **Differentiated Services**: Packets placed in transport classes (bundling of streams of different applications) \(\rightarrow\) preferences to classes
  - **Integrated Services**: Reservation of resources for individual connections \(\rightarrow\) guaranteed service quality, but no scalability
Stream based communication on the basis of SIP and RTP protocols
<table>
<thead>
<tr>
<th></th>
<th>RPC Based Communication</th>
<th>Message Based Communication</th>
<th>Stream Based Communication</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Timely Coupling of Sender and Receiver</strong></td>
<td>synchronous, through extensions also asynchronous</td>
<td>asynchronous</td>
<td>synchronous</td>
</tr>
<tr>
<td><strong>Communication Direction</strong></td>
<td>bidirectional</td>
<td>unidirectional</td>
<td>uni- and bidirectional</td>
</tr>
<tr>
<td><strong>Transported Data</strong></td>
<td>messages according to the request/response principle</td>
<td>messages according to the publish/subscribe principle</td>
<td>periodic data streams</td>
</tr>
<tr>
<td><strong>Abstraction Level</strong></td>
<td>high, calls embedded in programming language</td>
<td>medium, separate programming model</td>
<td>low, special programming interfaces</td>
</tr>
<tr>
<td><strong>Interoperability</strong></td>
<td>high, with IDL independent of programming language, web services also platform independent</td>
<td>to some extent, supported through standards such as JMS, however extended functionality is proprietary</td>
<td>high, through standards for protocols such as SIP and RTP as well as for codecs</td>
</tr>
<tr>
<td><strong>Mass Data Transfer</strong></td>
<td>supported through extensions</td>
<td>well supported</td>
<td>specially as isochronous stream</td>
</tr>
</tbody>
</table>
References

- Crane, D.: Ajax in Action. Manning, 2005
- Davies, S; Broadhurst, P; WebSphere MQ V6 Fundamentals. IBM Redbooks, 2005