Distributed Systems
Lecture 3:
RPC System Examples, Messaging, and Stream-based Communication
Prof. Dr. Alexander Schill
http://www.rn.inf.tu-dresden.de
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 & displays the new (X)HTML page
AJAX: Non blocking communication model

**Intentions of AJAX:**
- Web applications that look and feel like desktop applications
- Extend the classic web application with additional features; e.g., auto completion, immediate user feedback, faster responses
Introduction of intermediary — the AJAX engine

- Eliminates “click, wait, refresh” user interaction paradigm
- Responsible for rendering user interface and for server communication

- User actions are decoupled from communication with server
AJAX is not a single technology itself, but a combination of several (as defined by J.J. Garret)

- Standard-based presentation using XHTML and CSS
- Dynamic display & interaction using Document Object Model (DOM)
- Data interchange and manipulation using XML and XSLT
- Asynchronous data retrieval using XMLHttpRequest object
- JavaScript is binding everything together

- The concept can be applied to other exchange formats; e.g., JSON instead of XML
- Further, the concept can also utilise other transfer protocols; e.g., XML via WebSockets instead of XMLHttpRequest
Functionality of the AJAX engine

- User actions generate JavaScript calls to the AJAX engine instead of HTTP Requests to the server
- Not every response to user actions requires a request to the server; the AJAX engine operates on its own as long as it is able to compute on the local data
- Request to the server are minimised to cases when the AJAX engine
  - needs something from the server in response to user actions (i.e., local data insufficient or lack of required procedure)
  - a data synchronisation with the server is necessary or forced (e.g., result of a set of actions needs finalisation on server)

- The AJAX 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
- 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 Directory Service (formerly UDDI - Universal Description, Discovery and Integration)
- Standardisation by W3C and OASIS (Organization for the Advancement of Structured Information Standards)
- Specialisation by manufacturers of middleware
Web Services: Infrastructure

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

Service Index

UDDI

Service User

Service Provider

WSDL

SOAP*: originally ‘Simple Object Access Protocol’, now just SOAP
Web Services: SOAP

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

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

+ Independence of special programming languages
+ Support of sync. & async. interactions (e.g., Message Passing)

- Less efficient than direct communication due to HTTP overhead
- No automatic garbage collection
SOAP: Example (Method Call)

POST /CustomerServer HTTP/
Host: www.onlinetrader.com
Content-Type: application/soap+xml
Content-Length: 502
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: 436

<?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
  - (variable) arrays, enumerations, and many more
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/ns/wsdl/in-out">
      <input element="tns:ProductDescription" />
      <output element="tns:Product" />
    </operation>
    <operation name="searchProduct">
      <!-- ... -->
    </operation>
  </interface>
  ```

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

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

- No new technology, but well standardised 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 (e.g., EJB, .NET)
  - Web Services just provide comfortable access technology from client to server
Messaging: Message Queuing

- Mediation Component (Message Queue)
  - Simplification of indirect message exchange $\rightarrow$ 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 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

- Logon/logout-based dynamic coupling of application and local queues

- Abstraction level similar to object-oriented approaches

- Products and technologies
  - IBM MQ, Tibco, etc.
  - C++ and Java-Support (conformant to JMS)
  - Object orientation → messages and queues embedded as objects within the programming environment
  - XML for description of transferred content
  - support of essential operating system platforms
Product Example: IBM MQ

Features:

- Communication with Point-to-Point (1:1), Publish/Subscribe (m:n), Multicast (1:n), Funnel Aggregation (m:1)
- 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
- Integration of mobile devices (via MQ Telemetry Transport (MQTT))

- MQ Server
  - Provides queueing services to clients and other MQ Servers
  - Integrated or distributed Queue Manager

- MQ Client
  - Provides communication between application and MQ Server
1. Establish connection to queue manager (local or remote) → MQCONN
   - Authentication performed during connection
2. Open a particular queue object → MQOPEN
   - Authorisation is checked
3. App passes message to queue → MQPUT
4. Message is forwarded via the (unidirectional) message channel
5. App gets message → MQGET
6. (optional) Reply follows via separate queues and channels
Java Messaging Service (JMS)

- Programming interface for clients to access a MOM with different messaging functionalities
- Possible through standard object oriented interfaces

Example: Order pre-processing 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)

```java
... Queue orderQueue = (Queue) initialContext.lookup("Order"); QueueReceiver receiver = session.createReceiver(orderQueue); ObjectMessage order = (ObjectMessage) receiver.receive(); ...
/* optional confirmation sent back to sender */
order.acknowledge();
```
Messaging: Assessment

**Advantages**

+ Simple manageability
+ Robust message delivery
+ Flexible application fields (e.g., load balancing, parallelisation, batch-transmission)
+ Relevant for loose coupling of programmes, especially for Mobile Computing

**Disadvantages**

- Limited communication semantics
- Interaction model is different than with procedures/method invocations
- Limited accessibility of higher services
Stream-based Communication: Classification

- **Stream**: Sequences of time dependent video and audio packets
  - **Synchronous**: Packet stream with maximum packet delay time
  - **Isochronous**: Maximum AND minimum packet delay time observed
    - Limiting of jitter (variation of packet delay) by buffering
- **Complex streams**: combined and synchronized media
Application Classes & Requirements

- **Unidirectional**
  - Stream from server to one or more clients
  - No interaction relationship between sender and receiver
  - On-Demand Streams: Buffering possible for jitter limitation
  - Live Streams: Only limited buffering possible to avoid delays

- **Bidirectional**
  - Two-way interaction with timely connection
  - Point-to-Point or Multipoint Conferencing – several participants
  - Stricter requirements
    - Delay and jitter
    - Strong effects of high values on conversation quality; voice and lip synchronisation also important
Phases of Connection Control (similar for all streaming classes):

1. Connection establishment
   - Exchange of connection information (IP, Port)
   - 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 of streaming server
   - Switching to lower data quality due to increased network load

3. Connection Closure
   - Termination of data streams
   - Release of occupied resources
Stream-based Communication

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
    - Only stream (session) management
    - Actual stream (data) transfer via different protocol (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
Protocols

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

- **Real-Time Control Protocol (RTCP)**
  - Dynamic flow control of RTP connection → send control info back to sender
  - Information includes current quality of the connection
  - Enables sender to optimise transfer; e.g., by changing the codec and image refresh rate
Stream-based Communication on the basis of SIP and RTP protocols

Participant A

INVITE

100 Trying

180 Ringing

200 OK

SIP-Proxy A

INVITE

100 Trying

180 Ringing

200 OK

SIP-Proxy B

INVITE

180 Ringing

200 OK

Participant B

TRYING

ACK

BYE

Use Phase (e.g., with RTP)

Connection Establishment

Use

Connection Closure

Distributed Systems – Lecture 3

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</strong></td>
<td>synchronous,</td>
<td>asynchronous</td>
<td>synchronous</td>
</tr>
<tr>
<td><strong>Sender &amp; Receiver</strong></td>
<td>through extensions</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>also asynchronous</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Comm. Direction</strong></td>
<td>bidirectional</td>
<td>unidirectional</td>
<td>unidirectional and bidirectional</td>
</tr>
<tr>
<td><strong>Transported Data</strong></td>
<td>messages according</td>
<td>messages according</td>
<td>periodic data streams</td>
</tr>
<tr>
<td></td>
<td>to request/response</td>
<td>to publish/subscribe</td>
<td></td>
</tr>
<tr>
<td></td>
<td>principle</td>
<td>principle</td>
<td></td>
</tr>
<tr>
<td><strong>Abstraction Level</strong></td>
<td>high, calls embedded</td>
<td>medium, separate</td>
<td>low, special</td>
</tr>
<tr>
<td></td>
<td>in programming language</td>
<td>programming model</td>
<td>programming interfaces</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Interoperability</strong></td>
<td>high, with IDL independent of programming language</td>
<td>to some extent, supported through standards like JMS</td>
<td>high, standards for protocols like SIP and RTP, and for codecs</td>
</tr>
<tr>
<td></td>
<td>→ web services also</td>
<td>→ extended functionality is proprietary</td>
<td></td>
</tr>
<tr>
<td></td>
<td>platform independent</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Mass Data Transfer</strong></td>
<td>supported through</td>
<td>well supported</td>
<td>especially as isochronous stream</td>
</tr>
<tr>
<td></td>
<td>extensions</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

  *IETF*, 2002, RFC 3261