Distributed Systems

RPC System Examples, Messaging, and Stream-based Communication

Tenshi Hara
tenshi.hara@tu-dresden.de
https://www.rn.inf.tu-dresden.de/hara
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 & displays the new (X)HTML page

**Server**

**Client**

Time

Registration

username

john

password

xxxx

confirm pw

xxxx

submit

Registration

username

john_doe

password

xxxx

confirm pw

xxxx

submit

Registration

User 'john_doe' was successfully registered.

User 'john_doe' was successfully registered.

continue

Distributed Systems – RPC System Examples, Messaging, Stream-based Communication
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 is received

- High degree of dependence between HTTP Request and 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’ (CWR)
  - 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
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
Introduction of intermediary — the AJAX engine

- Eliminates CWR user interaction paradigm
- Responsible for rendering UI and communicating with the server

- 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 be also utilise other transfer protocols; 
e.g., XML via WebSocket 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.
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 required

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 middleware
1. Publication

2. Search

3. Reference to Service

4. Examination of Description

5. Service Use

*: originally ‘Simple Object Access Protocol’, now just 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 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>
<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

Client

Application

Stub

Stub Generation

Web Server

Application Server

Service Implementation (e.g. EJB)

Web Service Container

UDDI

Search Query

Registration of Service URL

URL

HTTP Request (URL)

HTTP Response (WSDL)

SOAP

Search Query

Stub Generation

Web Service Container

WSDL

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 (e.g., EJB, .NET)
  - Web Services just provide comfortable access technology from client to server
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 consumers
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 (before 2014: WebSphere 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, supports SOAP
- Integration of mobile devices (via MQ Telemetry Transport (MQTT))

- MQ Server
  - Provides queuing 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
   • Authorization 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
Queue Manager 1
1. MQOPEN - Message with destination details
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 (incl. header info)
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 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();
```
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

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 cached until time for display
  - Isochronous
    - Maximum AND Minimum lag time observed
    - Limiting of jitter (fluctuation of lag time) between packets
  - Complex streams
    - Combination of different media within one stream
    - The different media must be synchronized with each other
Application Classes and 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**
Application Classes and Requirements

- **Unidirectional**

- **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
Streams transferred in general process regardless of streaming class

3 Phases

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 in streaming server
   - Switching to lower data quality due to increased network load

3. Connection Closure
   - Termination of data streams
   - Release of occupied resources
Stream 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 Network Services

- **Differentiated Services**
  Bundling of streams of different applications
  → packets placed in transport classes
  → preferences matched to classes

- **Integrated Services**
  Reservation of resources for individual connections
  → guaranteed service quality
  → no scalability
Stream-based Communication on the basis of SIP and RTP protocols

Stream based communication on the basis of SIP and RTP protocols

Distributed Systems – RPC System Examples, Messaging, Stream-based Communication
| Timely Coupling of Sender & Receiver | synchronous, through extensions also asynchronous | asynchronous | synchronous |
| Comm. Direction | bidirectional | unidirectional | unidirectional and bidirectional |
| Transported Data | messages according request/response principle | messages according publish/subscribe principle | periodic data streams |
| Abstraction Level | high, calls embedded in programming language | medium, separate programming model | low, special programming interfaces |
| Interoperability | high, with IDL independent of programming language → web services also platform independent | to some extent, supported through standards like JMS → extended functionality is proprietary | high, through standards for protocols like SIP and RTP as well as for codecs |
| Mass Data Transfer | supported through extensions | well supported | specially as isochronous stream |
- Crane, D.: Ajax in Action. Manning, 2005
- Davies, S; Broadhurst, P; WebSphere MQ V6 Fundamentals. IBM Redbooks, 2005