Distributed Systems
Lecture 1: 
Introduction and System Architectures

Prof. Dr. Alexander Schill
http://www.rn.inf.tu-dresden.de
Organisational Information

- Lecture Slides and Tutorial Material at https://www.rn.inf.tu-dresden.de/ds
- Lectures: Thursdays, 14:50h (2:50p), APB/E023
- Tutorials: Tuesdays, 09:20h (9:20a), APB/E023

INF-PM-ANW:
- request a topic latest by the end of October (talk to Thomas Springer or send an e-mail)
- You will get assigned a topic for 90 hours of self study
- You need to search literature and prepare slides for that topic
- Exam is a 15 min colloquium
Contents

- Introduction and System Architectures
  - Basic Communication Mechanisms:
    - Remote Procedure Call (RPC)
    - RPC-based System Examples
  - Services for Distributed Systems:
    - Distributed Transactions
    - Security Services
    - Naming and Directory Services
  - Software Engineering for Distributed Systems:
    - Component Models
    - Development of Component-based Systems
    - Service-Oriented Architectures and Business Processes
  - Cloud Computing:
    - Basic Principles of Cloud Computing
    - Industry Lectures and Practice
Introduction: Outline

- Overview and Applications
- Definition of a Distributed System
- Models
- Architectures
Application examples

- eCommerce (Online Shops, Banking, Travel Booking)
- Enterprise software (ERP - Enterprise Resource Planning, CRM - Customer Relationship Management, SCM - Supply Chain Management)
- Teamwork (CSCW – Computer Supported Collaborative Work)
- Process control (CIM – Computer Integrated Manufacturing)
- Scientific Research, large-scale commercial applications (Computationally intensive problems – Cloud computing)

Motivation of Distribution

- Data, function and load distribution
- Scalability without changing components
- Decentralization, integration and co-operation
- Locality properties and efficiency
- Remote resource access
- Parallelization of processes for increased efficiency
- Fault tolerance: reliability and availability
“A Distributed System comprises several single components on different computers, which normally do not operate using shared memory and as a consequence communicate via the exchange of messages. The various components involved cooperate to achieve a common objective such as the performing of a business process.” Schill & Springer.

Main Characteristics:
- Multiple spatially separated components
- Components possess own memory
- Communication via messages
- Cooperation towards a common objective (such as business process execution, common database access, or resource sharing)
- Networked workstations, also organizationally integrated
- Strongly increasing communication performance
Application components (domain-specific) on top of infrastructure
Example System: E-Commerce

- Client
- Web Server
  - Application Server
    - Purchase Order Preprocessing
    - Web Service
  - Application Server
    - Administration of customer data
  - Application Server
    - Store management
- Store tables
- Customer data
- Online shop
- Supplier
  - Application Server
    - Web Service
  - Database
UML Diagram of E-Commerce System
Hardware / Firmware level (basic foundations of computer networks):
- Physical computer nodes (processor + storage)
- Direct / indirect computer coupling
  - local networks (Ethernet, WiFi/WLAN etc.)
  - high-performance networks (Gigabit Ethernet)
  - gateways / bridges / switches
  - global radio networks (GSM, UMTS, LTE, 5G)
- Transport-oriented communication protocols (TCP/IP, UDP/IP)

System and Application Level (focus of this lecture series):
- Communicating processes
  - complete logical connection
  - no complete physical connection (communication via gateways)
- System oriented resources (file system, threads, system programs)
- Distributed storage, decentralized, co-operative
- Distributed applications (domain-specific) on top of distributed systems
Distributed System Example: Infrastructure

- **Characteristics:**
  - Decentralized system structure
  - Heterogeneity of systems

- **Requirements**
  - Scalability
  - Fault tolerance
  - Security concepts
  - Consistency - Transaction processing
Required bridging between heterogeneous networks, system platforms and applications

→ Middleware
Def. of Middleware: “Infrastructure services implementing commonly used functionality for distributed applications in a reusable way for bridging of heterogeneity of different systems and networks”
Models: Client/Server

- **Roles:**
  - Client - Service user (e.g. Device / interface used by the customer)
  - Server – Service provider (e.g. Hosting Company website)

- **Hierarchical System** – Server can be client to another server

- **Client accesses functionality of server through remote invocations** (e.g. Remote Procedure Call)
  - Synchronous communication
  - Asynchronous communication

---

Control thread and data transfer

Separate address spaces
Objects of arbitrary granularity (e.g. a server or a customer record)
Local and remote objects
Objects can be passed using Value and Reference Parameter Semantics.
Dynamic object migration e.g. to facilitate local communication to avoid high data traffic over the network.
Realised through Java Remote Method Invocation and other programming languages
Cross-Programming Language approach – transfer objects in XML
## Comparison of system models

<table>
<thead>
<tr>
<th></th>
<th>Client/Server-Model</th>
<th>Distributed Object-Oriented Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>processing model</td>
<td>procedural model with restrictions</td>
<td>object communication</td>
</tr>
<tr>
<td>data access type</td>
<td>indirect data access via RPC-Server</td>
<td>direct access to objects</td>
</tr>
<tr>
<td>data transfer</td>
<td>value parameter semantics</td>
<td>reference parameter semantics</td>
</tr>
<tr>
<td>identity</td>
<td>not system-wide unique</td>
<td>system-wide unique</td>
</tr>
<tr>
<td>granularity</td>
<td>server of coarse granularity</td>
<td>objects of any granularity</td>
</tr>
<tr>
<td>placement</td>
<td>fixed placement</td>
<td>modifiable placement</td>
</tr>
</tbody>
</table>

Higher level of abstraction and improved influence on distribution with object-oriented model
- Lack of flexibility using inheritance in object oriented approach
- Components possess larger granularity than objects
- Simplified development by even higher level of abstraction
- Object-oriented programming languages can be used to implement components (components can consist of many objects)
Models: Component Based

- Separation of code and configuration

- Provide simplified programming interfaces for the access of system services (like transactions or security services)

- Specific tools for component based software development

- Examples of component based approaches: Enterprise JavaBeans, OSGi and Microsoft .NET-Components
Models: Service Oriented

- Bank
- Business A
- Supplier
- Payment Processing
- Service Interface
- Order Processing
- Customer Administration
- Discount Administration

- Warehouse Management
- Service Interface
- Order Processing
Models: Service Oriented

- Even higher level of abstraction than objects and components (e.g. technology independence)

- Principles of Service Oriented Architecture (SOA) and Web Services (WS); services are offered, searched for and used

- Loosely coupled; interoperability across platform and business borders

- Composition of services for complex processes

- Programming Language Independence
  - Accessible via service interfaces and SOAP protocol
  - Web Services Description Language based on XML
Architectures: Two/Three Tier

- **Client** (e.g. Point of Sale)
- **Client** (e.g. Fieldworker)
- **Client** (e.g. Home Access)
- **Server** (E.g. preprocessing)
- **Server** (E.g. Customer history database)

**Application Logic** *(Processing Tier)*

**Data management** *(Persistence Tier)*

- 2-tier: two-level structure (user-interface <-> server); simple, but less flexible
- 3-tier: three-level structure; preferable for complex applications

User interface, if necessary pre-processing (thin client vs. fat client)

*(Presentation Tier)*
Architectures: Clustering

Server Cluster

Preprocessing

Customer Base Data

Load balancing

Client (E.g. Point of Sale)

Client (E.g. Fieldworker)

Client (E.g. Home Access)
Replication by multiple Servers
- Processing, persistence and/or application data can be offered by several servers

Advantages:
- Load balancing
- Fault tolerance
- Parallel processing
- Higher processing speed

Challenges:
- Consistency between replicated data
  - Complexity due to synchronisation and conflict handling
Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. (NIST)

Cloud is a virtualized and often widely distributed environment (as compared to more centralized cluster)

- Five essential characteristics:
  1. On-demand self-service
  2. Broad network access
  3. Measured service
  4. Resource pooling
  5. Rapid elasticity
### Architectures: Cloud

**Service models:**
- Cloud Infrastructure as a Service (IaaS)
  - Amazon Web Services, OpenStack
- Cloud Platform as a Service (PaaS)
  - Azure Service Platform, Google App Engine, Heroku
- Cloud Software as a Service (SaaS)
  - Google Apps, Salesforce

**Implementation**
- Public Cloud
- Private Cloud
- Community Cloud
- Hybrid Cloud
- Decentralized architecture: direct communication between peers
- Peers act as service providers and users
- Mechanism for finding service providing peers necessary
- Application areas:
  - Parallel Applications
  - Content Management and Sharing
  - Collaboration
Summary: Technologies for Distributed Systems

Data Integrity

Transaction Monitors
Message Oriented Middleware

Client/Server, Remote Procedure Call (RPC)
(for instance DCE - Distributed Computing Environment)

Object Transaction Monitor

RMI-/ .NET/SOAP- Object-oriented Basic comm.

Application Server / SOA / Cloud
Component-Frameworks (Enterprise JavaBeans, OSGi, .NET)

Usability by application developer

Flexibility
References