Enterprise computing systems

close, direct coupling of application programs running on multiple, heterogeneous platforms in a networked environment.

These systems must be completely integrated and very reliable, in particular:

- information consistency, even in case of partial system breakdown.
- security and guaranteed privacy.
- adequate system response times.
- high tolerance in case of input and hardware/user errors (fault tolerance).
- autonomy of the individual system components.
The following factors contribute to the increasing importance of distributed systems:

- Decrease of processor and storage cost.
- High bandwidth networks
- Insufficient and often unpredictable response times of mainframe systems
- Growing number of applications with complex information management and complex graphical user interfaces.
- Growing cooperation and usage of shared resources by geographically dispersed users; caused by the globalization of markets and enterprises,
  - e.g. applying telecooperation (groupware, CSCW) and mobile communication to improve distributed teamwork.

**Informal definition**

**Methods of distribution**

In the following sections, we will focus on the latter three types of distribution, in particular on the processing distribution.

There are five fundamental methods of distribution:

1. Hardware components.
2. Load.
3. Data.
4. Control, e.g. a distributed operating system.
5. Processing, e.g. distributed execution of an application.

Distributed systems have a number of characteristics, among them are:

1. Existence of multiple functional units (physical, logical), e.g. software services.
2. Distribution of physical and logical functional units.
3. Functional units break down independently.
4. Distributed component control: a distributed operating system controls, integrates and homogenizes the distributed functional units
5. Transparency: details irrelevant for the user (e.g. distribution of data across several computers) remains hidden in order to reduce complexity.
6. Cooperative autonomy during the interaction among the physical and logical functional units ⇒ implies concurrency during process execution.
Challenges of distributed systems

The design of distributed systems poses a number of challenges:

- **Heterogeneity** applies to networks, computer hardware, operating systems, programming languages and implementations by different programmers.
  
  Use of **middleware** to provide a programming abstraction masking the heterogeneity of the underlying system.
  
  Middleware provides a uniform computational model for use by the programmers of servers and distributed applications.
  
  Middleware examples are **CORBA**, **Java RMI**.

- **Openness** requires standardized interfaces between the various resources.
  
- **Scalability**: adding new resources to the overall system.
  
- **Security**: for information resources.

- **Privacy**: protect user profile information.

Examples for development frameworks

There is a high motivation to use standardized development frameworks for the design and implementation of distributed applications.

- **Sun Network File System (NFS)** by **SUN**
  
  A distributed file system behaving like a centralized file system.

- **Open Network Computing (ONC)** by **SUN**
  
  Platform for distributed application design; it contains libraries for **remote procedure call (RPC)** and for **external data representation (XDR)**.

- **ODP (Open Distributed Processing)** by **ISO**
  
  Specification of the interfaces and the component behavior.

- **Common Object Request Broker Architecture (CORBA)** by **OMG**
  
  Defines a common architecture model for heterogeneous environments based on the object-oriented paradigm.

- **Java 2 Platform, Enterprise Edition (J2EE)** by **Sun**, e.g., **RMI**
  
  Component-based Java framework providing a simple, standardized platform for distributed applications, runtime infrastructure and a set of Java APIs.

- **.NET** framework of **Microsoft**
  
  Middleware platform especially for Microsoft environments.
  
  Consists of a class library and a runtime environment.
  
  Incorporates the distributed component object model (DCOM).

A **set of cooperating, interacting functional units**

Reasons for distribution: **parallelism** during the execution, **fault tolerance**, and **inherent distribution** of the application domain.

**Definition**

**Programmer’s perspective**

- Interfaces help to establish well-defined interaction points between the components of a distributed application.
  
- **Interfaces of a distributed application**
  
- **Interface specification**

**Distributed application vs. parallel program**
a set of cooperating, interacting functional units
reasons for distribution: parallelism during the execution, fault tolerance, and inherent distribution of the application domain.

Definition
Programmer’s perspective
Interfaces help to establish well-defined interaction points between the components of a distributed application.

Interfaces of a distributed application
Interface specification
Distributed application vs. parallel program

specifies component operations (name, functionality) and communication between components.
required parameters (including their types).
the results returned by the operation including arity and type.
visible side-effects caused by the component execution, for example data entry into a database.
effects of an operation on the results of subsequent operations.
constraints concerning the sequence of operations.

Although distributed applications might look similar to parallel programs at first glance, there are still some differences:

<table>
<thead>
<tr>
<th></th>
<th>distributed application</th>
<th>parallel program</th>
</tr>
</thead>
<tbody>
<tr>
<td>granularity</td>
<td>coarse</td>
<td>fine</td>
</tr>
<tr>
<td>data space</td>
<td>private</td>
<td>shared</td>
</tr>
<tr>
<td>failure handling</td>
<td>within the communication protocols not considered</td>
<td></td>
</tr>
</tbody>
</table>
Introduction

Issues
Issues of the following section
- Motivation for distributed systems and distributed applications.
- Basic terminology for distributed systems, e.g., terms like distributed applications, and interface.
- Introduction to some influential distributed systems, such as NFS File system, Mach and Java 2 Platform Enterprise Edition.

Background
Key Characteristics of distributed Systems
Distributed application
Influential distributed systems

Mach message exchange
Processes communicate through communication channels, called ports.

A port is realized as a message queue to which multiple senders may send messages; there is only one receiving process per queue.
Ports are protected by capabilities.
Mach supports network communication through network servers.

Mach
Mach is an operating system developed at Carnegie-Mellon University. It is characterized through its small kernel, called microkernel.

Goals of Mach
- Major design goals were
  - Emulation of Unix.
  - transparent extension to network operation.
  - portability.

Architecture
Mach message exchange
File catalogs are exported (by server subsystems) and mounted (by the client machines).

Support of a **mount service**:
- File `/etc/exports` on NFS server lists names of local filesystem available for remote mounting.
- Mounting request by client with: remote host, directory pathname and local name with which it is to be mounted.
- **automounter**: dynamically mounting of a remote directory whenever an 'empty' mount point is referenced by a client.

**NFS implementation**

NFS implementation is based on RPC calls between the involved operating systems. It can be configured to use UDP or TCP.

Earlier version of NFS was a **stateless** file server, i.e. a server subsystem does not store state information about its clients and their past operations.

Current version of NFS is a **stateful** file server, i.e. a server subsystem supports locking and delegation of actions to client to improve client-side caching.