# Literature



## Literature (1)

### Distributed Systems

- G. Coulouris, J. Dollimore, T. Kindberg: Distributed Systems: Concepts and Design, Addison-Wesley [Verteilte Systeme: Konzepte und Design]
- A. Tanenbaum, M. van Steen: Distributed Systems: Principles and Paradigms, Prentice-Hall / Pearson [Verteilte Systeme: Prinzipien und Paradigmen]
- Günther Bengel: Grundkurs Verteilte Systeme, Vieweg





## Literature (2)

### Programming for Distributed Systems

- Steffen Heinzl, Markus Mathes: Middleware in Java, Vieweg
- Dietmar Abts: Masterkurs Client/Server-Programmierung mit Java, Springer
- Rainer Oechsle: Parallele und verteilte Anwendungen in Java, Hanser

### Programming in Java

- The Java API-Documentation!!! A lot of books like
- Guido Krüger: Handbuch der Java-Programmierung, Addison-Wesley, also: http://www.javabuch.de
- Christian Ullenboom: Java ist auch eine Insel, Galileo Computing, also: http://openbook.rheinwerk-verlag.de/javainsel/



# Introduction

Justus Klingemann

# What is a Distributed System?

- A distributed system consists of multiple autonomous systems that do not share a common main memory but cooperate by exchanging messages via a network (Bal)
- A distributed system is a collection of independent computers that appears to its users as a single coherent system (Tanenbaum)





# Problems in Distributed Systems

Network transmission can induce errors

- Messages can get lost or damaged
- Messages can change their order
- Network latency

Subsystems can break

No common state

No common time





# Why do Distributed Systems exist?

They allow resource sharing

- hardware, data, services

Economy

- Buying a set of custom of the shelf (COTS) systems is usually cheaper than one big system

Set of resources can be easily adapted

- supports flexibility, scalability and reliability

Support for applications that are inherently distributed

- Examples: different branches of a bank, factory automation, distributed information services (WWW)

Reliability





# Issues in Distributed Systems

What are we trying to achieve when we construct a distributed system?

Certain goals are common to many distributed systems

- Resource Sharing
- Openness
- Concurrency
- Scalability
- Fault Tolerance
- Transparency

# Resource Sharing

Ability to use any hardware, software or data anywhere in the system.

Resource manager controls access, provides naming scheme and controls concurrency.

Resource sharing model (e.g. client / server or object-based) describing how

- resources are provided,
- they are used and
- provider and user interact with each other.





# Openness

Openness is concerned with extensions and improvements of distributed systems.

- New components have to be integrated with existing components.

Two aspects:

- Interoperability: systems using heterogeneous technologies or from different vendors cooperate
- Portability: systems can be ported to different platforms

Detailed interfaces of components need to be published.

- Examples: Sockets, CORBA

Differences in data representation of interface types on different processors (of different vendors) have to be resolved.





# Concurrency

Components in distributed systems are executed in concurrent processes.

Components access and update shared resources (e.g. variables, databases).

Integrity of the system may be violated if concurrent updates are not coordinated.

- Lost updates
- Inconsistent analysis





# Scalability

Adaption of distributed systems to

- accommodate more users
- respond faster (this is the hard one)

Usually done by adding more and/or faster processors.

Components should not need to be changed when scale of a system increases.

# Fault Tolerance

Hardware, software and networks can fail

Distributed systems must maintain availability even at low levels of hardware/software/network reliability.

Fault tolerance is achieved by

- recovery
- redundancy





# Transparency

Distributed systems should be perceived by users and application programmers as a whole rather than as a collection of cooperating components.

Transparency has different dimensions

These represent various properties that distributed systems should have.





# Transparency Dimensions (1)

Access Transparency

- Enables local and remote objects to be accessed using identical operations.

Location Transparency

- Enables objects to be accessed without knowledge of their location.
- access via a location-independent name

Migration Transparency

- Allows the movement of objects within a system without affecting the operations of users or application programs

Replication Transparency

- Enables multiple instances of objects to be used to increase reliability and performance without knowledge of the replicas by users or application programs





# Transparency Dimensions (2)

Concurrency Transparency

- Enables several processes to operate concurrently using shared objects without interference between them.

Failure Transparency

- Enables the concealment of faults
- Allows users and applications to complete their tasks despite the failure of other components.

Scaling Transparency

- Allows the system and applications to expand in scale without change to the system structure or the application algorithms.