Script **generated by TTT**

Title: Distributed_Applications (24.04.2012)

Date: Tue Apr 24 14:30:07 CEST 2012

Duration: 89:46 min

Pages: 19

System Models

A distributed system can be described in form of descriptive models.

Architectural model

defines the interaction between components and the mapping onto the underlying network.

- Software layers
- System architectures

Interaction model

Failure model

Security model

The following sections of the course will discuss in more detail various aspects of these system models.

Generated by Targeteam

The failure model defines the ways in which failures may occur and how they are handled.

distinction between failures of processes and communication channels.

different types of failures:

- crash faults: the process simply stops due to Hardware failures or Software errors.
- message loss: messages may be lost due to buffer overflow of routers or network congestion.
- fail stop failures: the process fails by crashing; system notifies relevant partners.
- timing failures: a local clock exceeds the bounds on its rate of drift from real time or transmission takes longer than the specified bound.
- arbitrary failures (non-malicious Byzantine failure): a process arbitrarily omits intended processing steps, takes unintended processing steps or sends corrupted messages.
- malicious Byzantine failure: an attacker who has studied the system attempts to break it. Examples are the corruption or replay of messages, or the modification of the program (install hacked version).

Generated by Targeteam

The security model defines the possible threats to processes and communication and the ways how they are handled.

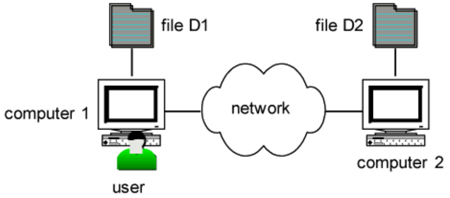
- secure communication channels, e.g. use of cryptography.
- protecting objects against unauthorized access.
- authentication of messages; proving the identities of their senders.

Generated by Targeteam

Access transparency

Problem : How to access objects in a distributed system.

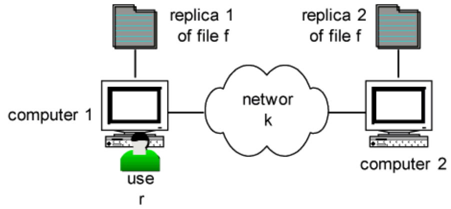
⇒ Access transparency provides access to local and remote objects in exactly the same way.



For reasons of availability or fast access, resources, e.g. objects may be replicated.

Problem : Management of several copies of an object in a distributed system.

⇒ Replication transparency means that the user is unaware of whether an object is replicated or not. The user accesses replicated objects as if they exist only once.



A variety of protocols have been proposed that deal with the problem of consistency among replicated files (Update of replicated files) .

Problem : Object relocation in distributed systems.

⇒ Objects may migrate from one computer to another without influencing the correct behavior of running applications.

Host migration transparency

Problem : Computer migrates from one subnetwork to another subnetwork, e.g. if a user connects his laptop computer to different subnetworks. This requires a dynamic assignment of the IP address (e.g. DHCP), a name server, etc.

⇒ The computer supports the same environment, the same applications, and the same look-and-feel, no matter where the mobile workers are currently connected to the network.

Types of migration

off-line migration.

on-line migration.

Transparency

key concept for better exploitation of resources within a distributed, heterogeneous system.

Location transparency

Access transparency

Replication transparency

Migration transparency

Language transparency

Other transparencies

Goal for distributed applications

There are a number of other transparencies relevant for distributed systems.

Failure transparency

Problem : Partial failure in distributed systems, for example computer crashes or network failures.

⇒ Up to a certain degree, failures are masked by the system.

Concurrency transparency

concurrent access to shared resources by distributed users or application components.

Problem : shared access to objects in distributed systems.

⇒ Several users or application programs can access objects simultaneously (for example shared data) without mutual influence.

Execution transparency

Execution transparency implies that processes may be processed on different runtime systems.

Performance transparency

allows for dynamic reconfiguration of the system to improve the overall system performance when changes in load characteristics are detected.

Scalability transparency

supports extensions and enhancements of the system or the applications without the need of modifications to the system structure or changes to the application algorithms.

Transparency

key concept for better exploitation of resources within a distributed, heterogeneous system.

Location transparency

Access transparency

Replication transparency

Migration transparency

Language transparency

Other transparencies

Goal for distributed applications

key concept for better exploitation of resources within a distributed, heterogeneous system.

Location transparency

Access transparency

Replication transparency

Migration transparency

Language transparency

Other transparencies

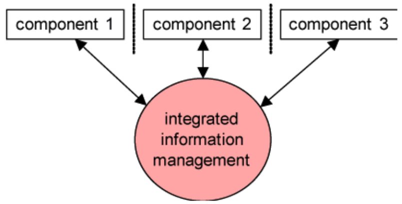
Goal for distributed applications

Generated by Targeteam

Components of a distributed application communicate through shared, integrated information management.
Examples

sharing documents, URLs: BSCW Workspace

sharing objects, software components: JavaSpaces



No direct communication between components, e.g. distributed shared memory.

Paradigms for distributed applications

Message exchange takes place between a sending and a receiving process.

Basic functionality

```
send(E: receiver, N: message);

receive(S: sender, B: buffer);
```

Communication perspectives

We can distinguish between different perspectives with respect to the communication among the involved processes:

- the sender's view, and
- the receiver's view

Assumption: Sender S has invoked the operation `send(E, N)`; receiver E performs the operation `receive(S, B)`.

Remote-invocation send

Sender S suspends execution until the receiver has received and processed the submitted request that was delivered as part of the message.

Advantages of asynchronous message exchange

Advantages

- useful for real-time applications, especially if the sending process should not be blocked.
- supports parallel execution threads at the sender's and the receiver's sites.
- it can be used for event signaling purposes.

Disadvantages

- management of message buffers, handling of buffer overflow, access control problems, and of process crashes (receiver).
- notification of S in case of failures may be a problem, since mostly S has already continued with its regular processing.
- design of a correct system is difficult. The failure behavior depends heavily on buffer sizes, buffer contents, and the time behavior of the exchanged messages.

Advantages of asynchronous message exchange

Advantages

- useful for real-time applications, especially if the sending process should not be blocked.
- supports parallel execution threads at the sender's and the receiver's sites.
- it can be used for event signaling purposes.

Disadvantages

- management of message buffers, handling of buffer overflow, access control problems, and of process crashes (receiver).
- notification of S in case of failures may be a problem, since mostly S has already continued with its regular processing.
- design of a correct system is difficult. The failure behavior depends heavily on buffer sizes, buffer contents, and the time behavior of the exchanged messages.

Asynchronous message exchange (nonblocking)

Sender S can resume its processing immediately after the message N is put forward into the message queue NP (NP is also called message buffer).

- S will ***not*** wait until the receiver E has received the message N.
- A `receive` operation indicates that the receiver is interested in receiving a message.

Example

Advantages of asynchronous message exchange

Asynchronous message exchange (nonblocking)

Synchronous message exchange (blocking)

Remote-invocation send

Sender S suspends execution until the receiver has received and processed the submitted request that was delivered as part of the message.

Paradigms for distributed applications

Information Sharing

Message exchange

Naming entities

Bidirectional communication

Producer-consumer interaction

Client-server model

Peer-to-peer model

Group model

Taxonomy of communication

Message serialization

Levels of Abstraction