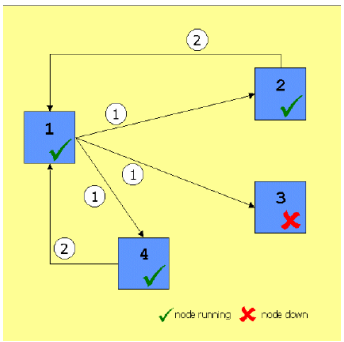
TEICS

Peer-to-Peer-System zur Veranschaulichung von unterschiedlichen Netzwerktechnologien. ShareMe ist nicht für den praktischen ein Satz gedacht.

Was heißt Peer?
"Gleichgestellter", "Ebenbürtiger" oder "Altersgenosse/in"

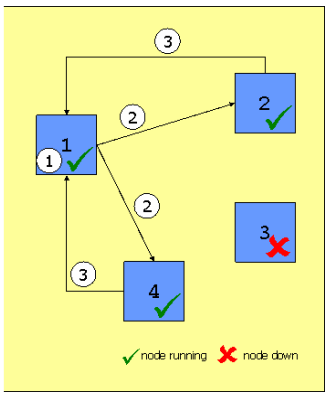
Definition Peer-to-Peer-System:
In einem Peer-to-Peer-Netz sind alle Computer gleichberechtigt und können sowohl Dienste in Anspruch nehmen als auch Dienste zur Verfügung stellen. Die Computer können als Arbeitsstationen genutzt werden, aber auch Aufgaben im Netz anbieten.

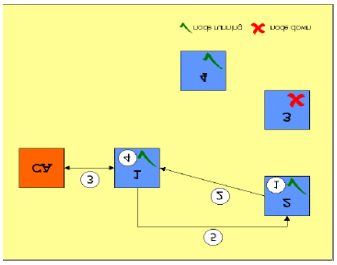
ShareMe-Tasks



Aufbau des p2p Netzes durch UDP Multicast. Es werden in regelmäßigen Abständen IsAlive Nachrichten via UDP Multicast versendet. GarbageCollector streicht abgeschaltete Peers aus der Hostliste.

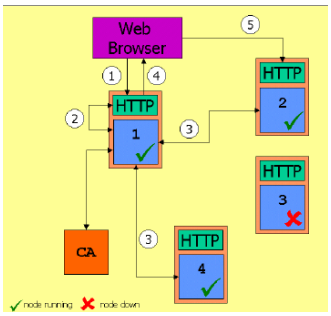
Die Suche nach Files erfolgt über RMI. Jeder Peer implementiert einen RMI Client und Server. Die Informationen wie auf den RMI Server zugegriffen werden kann ist in der IsAlive Nachricht von Punkt 1 gespeichert.





Die Suchanfragen und Suchergebnisse sollen signiert werden. Es werden asymmetrische Schlüssel und die MD5 Hashfunktion verwendet. Die öffentlichen Schlüssel werden in der Certification Authority(CA) gespeichert. Um Signaturen zu verifizieren müssen die Peers via CORBA auf die CA zugreifen.

Der Download von Files erfolgt über HTTP. Jeder Peer implementiert einen HTTP Server. Das ShareMe System wird über den Web Browser Form benutzt. Der HTTP GetRequest wird in eine ShareMe RMI Suchanfragen umgewandelt. Das Suchergebnis wird als HTML Seite aufbereitet. Die HTML Seite verlinkt auf die Files im Suchergebnis.



Task 1.1 Property File

Das Verhalten eines ShareMe Peers wird durch ein Konfigurationsfile bestimmt. (Ports, Service Name, etc.)

- ShareMe verwendet Java Property Files.
- Java Property File Format:
  - propertyName=value
  - ropertyName2=value2
  - propertyName3=value3

Task 1.2 ShareMe Grundgerüst

Die ShareMeImplKlasse ist die Basisklasse in der das PropertyFile eingelesen wird und die Threads für andere Aufgaben gestartet werden.

- Die Klasse befindet sich im **at.ac.tuwien.infosys.rnue.implementation** Package.
- ShareMeImplimplementiert das IShareMeInterface.
- Aufruf zum start eines ShareMe Peers:
  - javaat.ac.tuwien.infosys.rnue.helpers.ShareMeMain{propertyfile}

Task 1.3 Beenden von ShareMe

Wenn ShareMe fertig implementiert ist, dann laufen mehrere Threads parallel. Um den ShareMe Peer zu stoppen müssen alle Threads terminieren. Der ShutdownListenter wartet auf ein UDP Packet mit dem richtigen ShutdownPassword und veranlasst, dass alle Threads benachrichtigt werden.

- IConstants.EXIT_FLAG+ " " + password;

StopShareMe kann verwendet werden um ein UPD Packet mir einem Passwort an ShareMe zu senden.

- javaat.at.tuwien.infosys.rnue.helpers.StopShareMe–p {port} [-h {host}]

Threads

Threads sind Programmteile, die parallel, als unabhängig von einander, ablaufen. Es gibt zwei Arten Threads in Java zu Programmen zu deklarieren:

- ThreadKlasse
- RunnableInterface

Der Code in der run()Methode wird innerhalb des Threads ausgeführt. Ein Thread wird mit der start()Methode gestartet.

Stoppen von Threads: Ein Thread wird beendet, wenn die Abarbeitung des Codes in der run()Methode beendet ist. Typische Implementierung: Schleife die durch ein Stopp-Flag kontrolliert wird.

```
boolean keepRunning= true;
void run() {
        while keepRunning {
//do something
        }
}
```

Stopp-Flag wird durch zusätzliche Methode kontrolliert

```
void Stop() {
keepRunning = false;
}
```

Klassen um UDP Package zu senden und zu empfangen

- **DatagramSocket**
- **DatagramPacket**

Task 1.4 Empfangen von IsAlive Nachrichten

Unser ShareMe Peer muss **IsAlive Nachrichten** von anderen Peers empfangen, um zu wissen welche Peers online sind. Dies geschieht im **IsAliveReceiver** der das **IsAliveReceiverInterface** implementiert. IsAlive Nachrichten werden als UDP Multicast Nachrichten versendet. Die IsAlive Nachrichten enthalten ein serialisiertes Objekt welches das **IHostInfoMessage** implementiert. **IHostInfoMessage** enthält alle Informationen um eine Peer via RMI eine Suchanfrage zu senden.

Klassen um UDP Multicast Package zu senden und zu empfangen

- **InetAdress**
- **MulticastSocket**
- **DatagramPacket**

Serialisieren und Deserialisieren von Objekten

Damit ein Objekt serialisiert werden kann, muss es das **SerializableInterface** implementieren. Das Interface Serializable ist ein so genanntes **Marker-Interface**, d.h. man muss keine Methoden implementieren, sondern es kommt nur darauf an, ob das Interface geerbt wird oder nicht. In einem serialisiertem Objekt bleiben alle Variablenzuweisungen erhalten. Serialisierte Objekte können über das Netzwerk versendet werden oder auf der Festplatte gesichert werden.

Klassen zu Serialisieren und Deserialisieren von Objekten

- **ObjectInputStream**
- **ObjectOutputStream**

Task 1.5 Die Liste der aktiven Peers verwalten

Die empfangenen IhostInfoMessage werden in der HostListImpl gespeichert. Die HostListImpl ist eine Hashtable mit dem {RMIRegistryHost}:{RMIRegistryPort}als Key und einem Objekt welches das IHostInfoInterface implementiert als value. IHostInfo speichert die IhostInfoMessage und einen Timestap. Ein GarbageCollector(GC), der als Inner Class implementiert ist, durchläuft die HostList in regelmäßigen Abständen und löscht abgelaufene Einträge.

IsAlive Intervall	GC Intervall	Kommentar
20s	1s	
5s	5s	
5s	20s	
1s	20s	

Aufgaben in regelmäßigen Abständen erledigen: wait()

Der GarbageCollector ist ein Thread der in regelmäßigen Abständen laufen soll. Dies wird durch ein wait(longtimeout) in der Schleife der run()Methode implementiert. Die Bearbeitung des Thread wird gestoppt und er kommt in die Warteschlange. Nach der Zeit timeout wird der Thread wieder aktiv. Ein notify()in der stop()Methode des GC aktiviert den Thread wieder damit er terminieren kann.

Task 1.6 IsAlive Nachrichten versenden

Der IsAliverSender versendet in regelmäßigen Abständen IsAlive Nachrichten. Die IsAlive Nachrichten werden per UDP Multicast versendet und enthalten eine serialisierte IHostInfoMessage.

Aufgaben in regelmäßigen Abständen erledigen: timertask

Die Klasse muss nicht von Thread erben oder Runnable implementieren wie andere Threads, sondern von Timertask erben. Den Timertask startet man über die Timer Klasse und die scheduleAtFixedRate()Methode. Einen Timertask stoppt man über die cancel()Methode.

Software-Architecture

Outline:

- Motivation
- Foundations
- Software Architecture Styles

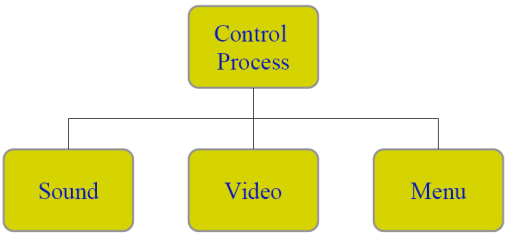
Goals:

- A framework to support the development of software
- Integration platform for future enhancements
- Interface definition for collaboration of components

Motivation:

If the size and complexity of a software system increase, the global structure of the system becomes more important than the selection of specific algorithms and data structures.

Example: System diagram



System Diagram Components

- What characterizes the components?
- Do they run on separate processors or at different times?
- Do they correspond to programs and/or processes?
- Do they describe a project organization or runtime entities?
- Are they modules, objects, tasks, functions, processes, distributed computations, etc.?

System Diagram Connections

What are the connections?

- communication
- control
- usage
- procedure call
- synchronization

System Diagram Layout

- What is the semantics of the layout?
- What do the different levels mean?
- What is the semantics of the hierarchy?
- Was there not enough space to place all four components on the same level?

Structure issues of software architecture

- Global organization and control structures
- Protocols for communication, synchronization and data access
- Physical distribution
- Composition of design elements
- Scalability and performance
- Selection of design alternative

Rationale for software architecture

- Communication support for reviews
- Documentation of early design decisions
- Transferable model of the whole system

Support for communications

- Different stakeholders
  - Customer, user
  - Project manager, programmer, tester
  - Helpdesk, quality assurance
- Similar interests from different viewpoints

Documentation of design decisions

- Constraints for the implementation
- Organizational structure for development and maintenance
- Achievability of planned quality attributes
- Reference for new project members
- Assessment of changes based on the architecture

Transferable Model of the system

- Product families have a common architecture
- Integration of existing components in case of compatibility
- Function of a component is independent of the communication mechanisms
- Definition of a design vocabulary makes cooperation easier

Software Architecture Granularity

Architectures for single systems

- Applications used in specific departments
- Programs for particular business areas

Architecture of the whole system

- architecture of a company
- Overview of interrelationships of systems

Foundations

Basic elements of software architecture

- Components
- Connectors
- Constraints
- Rationale

Components

- Decomposition of a system (multi-version, multi-person)
- Criteria for component decomposition
    - Modularization, encapsulation, information hiding, abstraction
- Functional Decomposition
- Optimization of performance (e.g. distribution onto parallel processors)

Connectors

Component A ... Component B

- sends data to / communicates with
- calls / uses
- is performed before / after
- shares common knowledge with
- is an instance of (e.g. object and class)
- runs in parallel with
- must not run in parallel with

Constraints

- Components must be constrained to provide that
  - the required functionality is achieved
  - no functionality is duplicated
  - the required performance is achieved
  - the requirements are met
  - modularity is realized (e.g. which modules
  - interact with the operating system)
- Assignment of functionality

Basic explanation (Why?)

- Often disregarded
- For multi-version software its design rationales must be documented:
  - Decomposition into components
  - Connections between components
  - Constraints on components and connections
- Serves as plan for future enhancements
- Serves as support/aid for maintainers

Commitment of the architecture

- As one of the early design decisions it is difficult to change the architecture
- The organization of the project is influenced substantially:
  - teams, documentation, configuration, management, maintenance, integration and tests
- The architecture must not prevent a beneficial implementation

Impact onto the life-cycle

- The architecture substantially impacts performance and available system resources
- The architecture determines the simplicity of future changes and adaptations
- A successful architecture can be used to build similar systems:
  - “product family” and
  - “domain specific software architectures”

Requirements for software architecture

- Fulfillment of functional requirements
  - Input/Output behavior
- Fulfillment of desired performance
  - Timing, preciseness, stability
  - Memory workload, other resources
- Can be verified by observation of the running system

Non-Functional Requirements

- Software architectures must also fulfill the following requirements:
  - Adaptability
  - Flexibility
  - Portability
  - Interoperability
  - Reusability within “related” projects

Static and dynamic structures

- Module structure
  - for configuration, non-existent at run-time
- Distribution structure
  - at run-time
- Dynamic structures influence
  - non-functional Requirements
  - functional Requirements and system
  - performance

Views of software architectures

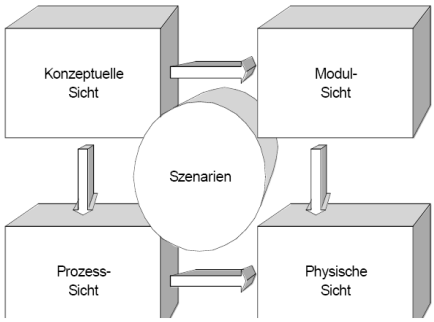
4+1 Views

Problem

- ambiguous diagrams
- overloaded diagrams

Solution/approach

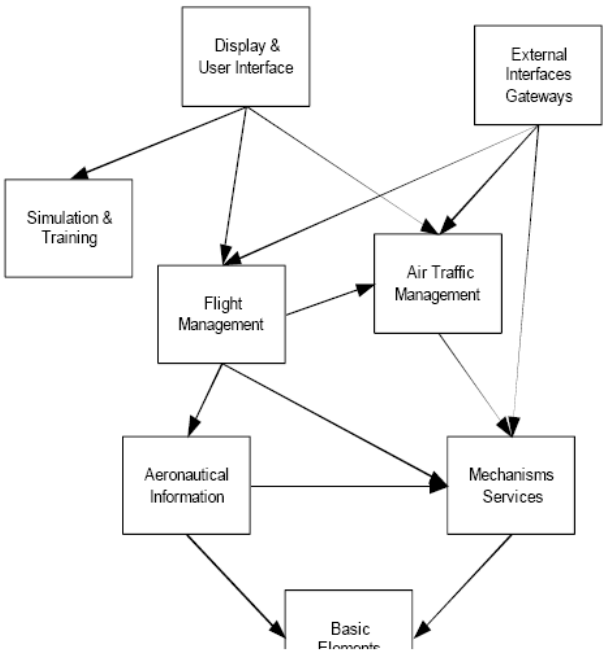
- Different perspectives
- Connection between single views



Conceptual (logical) view

- Functional requirements
- Orientation on problem domain
- Communication with experts
- Independence of implementation decisions
- “Frameworks”

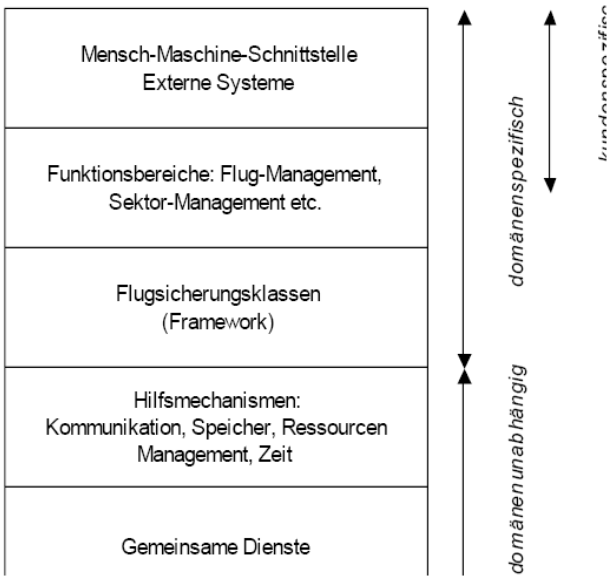
Example (Air traffic management system):



Modul (Development) View

- Organization of modules
  - Subsystems
  - Coherent parts in the development
  - Allocation of effort (development, maintenance)
- Organization in hierarchical layers
  - OSI communication protocols
- Compile-time structure
  - marginal for the operation of the system

Example (Air traffic management system):



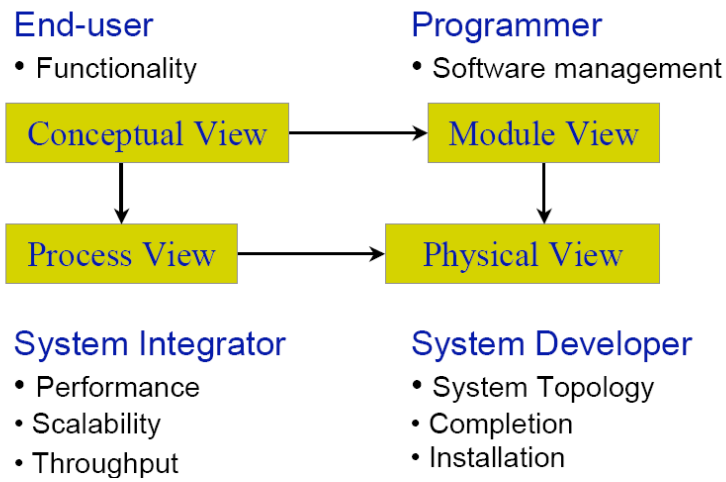
Process and coordination view

- Dynamic aspects of the run-time processes
  - Process creation
  - Synchronization
  - Concurrency
- Components of this view are processes: instructions and separate execution logic
- At run-time different reconfigurations can be done
- Estimates for process allocation etc.

Physical View

- Mapping of software on existing/available hardware
  - e.g. distribution of computations in a distributed system
- Impact on
  - availability, reliability, performance and scalability
- Structuring should have little or no influence on the implementation of the
- components

Integration of Views



Aspects of 4+1 views

Sicht	konzeptuelle	Prozess	Modul	physische	Szenarien
Kompo-nenten	Klasse	Task	Module, Subsystem	Rechner	Schritt, Ab-lauf
Konnekt oren	Assoziation, Vererbung, Komposition, Aggregation	Nach-richt, Broad-cast, RPC, etc.	Kompilier-abhängig-keit, „inclu-de“ etc.	Kommunika-tionsmedi-um, LAN, WAN, Bus etc.	
Contai-ner	Klassenkate-gorie	Prozess	Subsystem (Bibliothek)	Physisches Subsystem	Web
Betei-ligte	Endbenutzer	System-Designer, Integrie-rer	Entwickler, Leiter	System-Designer	Endbenut-zer, Ent-wickler
Concerns	Funktionalität	Perfor-manz, Verfüg-barkeit, Fehlerto-leranz, Integrität	Organisati-on, Wieder-verwendung, Portabilität, Produkt-Familie	Skalierbar-keit, Perfor-manz, Ver-fügbarkeit	Verständ-lichkeit

Software Architecture Styles

What is it?

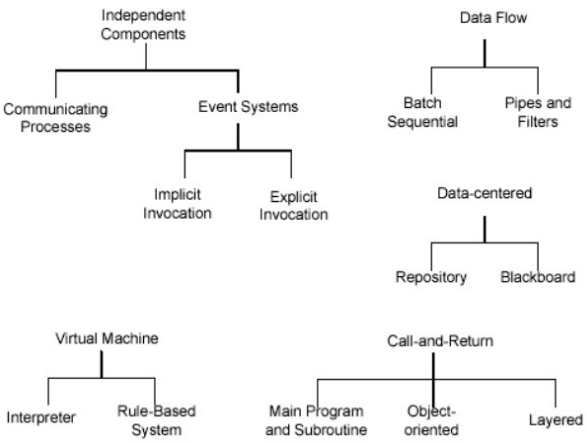
- Eine Design-Sprache für eine Klasse von Systemen
- Vokabular für Design-Elemente, z.B. Pipes, Filter, Server, Parser, DBs etc.
- Design-Regeln und Einschränkungen (constraints), z.B. C/S-Verhältnis n:1 etc.
- Semantische Interpretation
- Analysen zur Überprüfung der Entsprechung eines Entwurfs, z.B. Deadlock-Erkennung für C/S, Schedulability-Analyse etc.

Definition eines Arch Styles (Perry u. Wolf 1992)

Ein architektureller Stil definiert eine Familie von Software-Systemen bezüglich ihrer strukturellen Organisation.

Ein architektureller Stil drückt die Komponenten und Relationen zwischen diesen gemeinsam mit den Einschränkungen ihrer Anwendung, der assoziierten Komposition und den Design-Regeln für deren Konstruktion aus.

Katalog von Arch Styles



Software Architecture Styles

- Dataflow systems
  - Batch sequential
  - Pipes and filters
- Call-and-return systems
  - Main program and subroutine
  - Hierarchical layers
  - OO systems
- Independent components
  - Communicating processes
  - Event systems
- Virtual machines
  - Interpreters
  - Rule-based systems
- Data centered systems
  - Databases
  - Hypertext systems
  - Blackboards

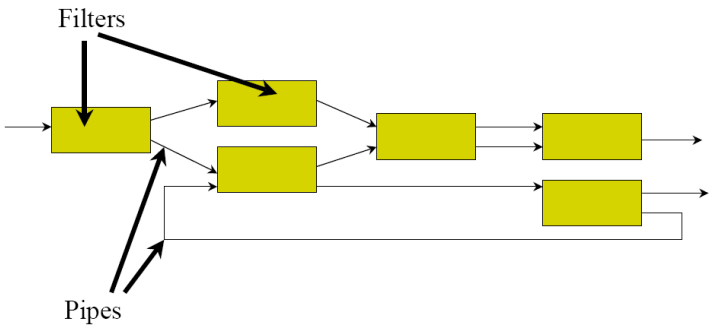
Pipes & Filters

Pipes:
Provide the output of a filter as input to another filter

Filters:
Read in input data stream and transform it into an output data stream

Pipes & Filters System Design

Pipes & Filters



Filters are independent components that

- do not share status with other components
- do not know the identity of their neighbors (input/output)

Pipelines

- Constrain the topology to a linear configuration of filters

Bounded Pipes

- Constrain the amount of data that a pipe can store temporarily

Typed Pipes

- The data stream must have a specific type

Examples for Pipes & Filters

- Unix shell: piping of components (commands) via "|“
  - cat {myfile} | grep “arch” | sort … | more
- Signal processing
- Parallel programming
- Functional programming
- Distributed Systems
- Specializations
  - pipelines
  - bounded pipes (limited memory)
  - typed pipes (for specific type of data)

Advantages

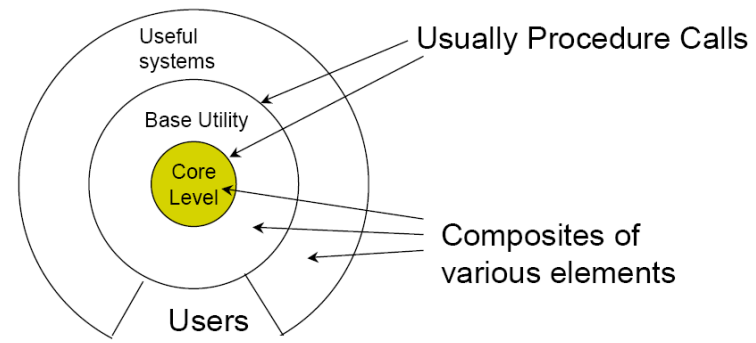
- A designer can define the input/output behavior of the whole system as combination of single filters
- simple; no complex component interactions
- filters as black-box and, therefore, substitutable
- Reusability
  - Two filters can be arranged arbitrarily, as long as they support the same data format / stream
- Maintenance
  - Integration of new filters
  - Substitution of existing/integrated filters
- Hierarchical structures easy to compose
- Analysis of
  - Throughput and potential deadlocks
- Concurrent execution
  - Filters are synchronized by the data transfer

Disadvantages

- Batch processing Charakteristik, aber ungeeignet für interaktive Applikationen
- Handling von unabhängigen Datenströmen
- Filter verlangen nach gemeinsamen Datenformat
- Wird der Datenstrom in Tokens analysiert, so hat jeder Filter das Parsen & Un-parsen extra
- Kommt ein Filter nicht ohne vollständiges Einlesen des Datenstroms aus (z.B. Sortieren) => Puffer!
- Ist jeder Filter ein separater Prozess =>Overhead

Layered Architecture

- Hierarchically organized system
- Each layer can only interact with the directly connected upper and lower layers
- Interfaces and protocols describe the communication between layers
- Each layer represents and implements an abstract virtual machine



Advantages

- Support of abstraction levels by layering
  - A larger problem is decomposed into several smaller ones
- Changes in one layer affect at most the two neighboring layers (interface, protocol)
- Reusability
  - Standard interfaces can be reused often
  - Different implementations of the same layer and their substitution

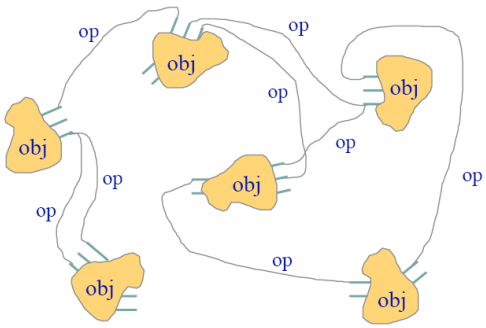
Disadvantages

- Not all systems can be decomposed into layers
- Communication between non-neighboring layers can be necessary
- Abstractions of some layers can be difficult to comprehend
- Skipping of several layers can cause difficulties

Object-oriented Organisation

- Supports data abstraction
- Encapsulation of data and corresponding operations
  - Attributes and methods
- "Manager" who ensures consistency of the data
  - Objects are self dependent for their integrity (invariant)
  - Internal representation of data is hidden (no direct manipulation of data)

Objects



General Properties

- Objects can be active or passive
  - parallel computations
- Objects can have different interfaces (role and client dependent)

Advantages

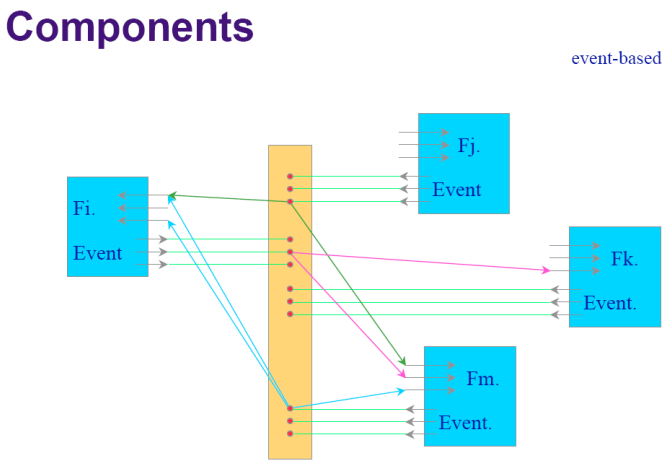
- Hiding of implementation, only the interface is visible for the outside
- Changes to one object do not affect other objects (as long as the interface remains unchanged) Objects are a good design tool
- Data and access operations are put together

Disadvantages

- To communicate an object has to know the identity of the other object
- If ids change all "clients" must be adapted accordingly
- Side-effects and mutual influence in case of concurrent object access

Event-based Systems

- Functions are not executed through a direct procedure/method call
- Components raise an event (publishers)
- Other “interested” components (subscribers) are notified and react accordingly
- Correlation of events and event handling is unknown to the components



Advantages

- Extensibility and Reusability
  - A new component can be easily integrated into the system
  - Subsequent registration for other events and announcement of its own events
- Exchangeability of components
  - Without influence on the interfaces of other components

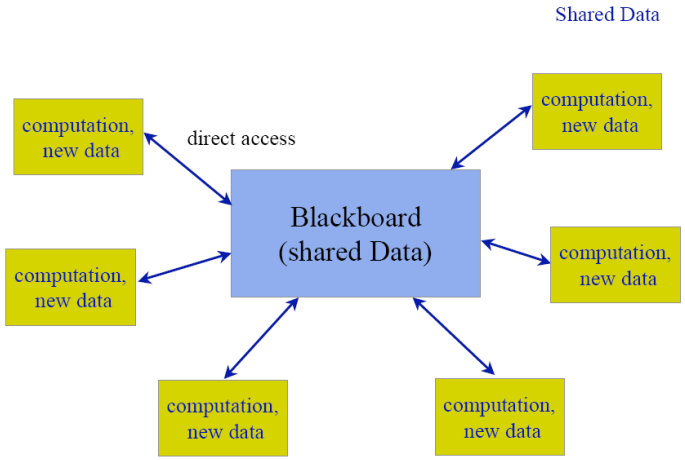
Disadvantages

- If an event is published, it is not assured that it is being handled by others
  - processing sequence
- Data exchange other than with events is problematic
- Behavior of components is tightly coupled with the execution environment (e.g. event Model)

Shared Data

- Two kinds of components:
  - Central data management
  - Independent components for computation
- Activation of computation
  - When inserting (storing) new data (database trigger)
  - Trough the actual state

Blackboard



Rating

- Control can be realized in different parts of the architecture
- This style can also be used to model batch processing with a shared database

Faustregeln für Datenfluss

Architektureller Stil	anzuwenden, wenn …
Datenfluss	Das System produziert einen wohl definierten, einfach identifizierbaren Output, der das direkte Resultat von sequentiellen Transformationen aus wohl definierten Inputs in einer Zeit-unabhängigen Art und Weise ist. Die Integrierbarkeit (resultierend aus einfachen Schnittstellen der Komponenten) ist wesentlich.
• Batch sequential	• Eine singuläre Output-Operation ist das Resultat des Lesens einer Menge von Input-Daten und deren Transformationen sind sequenziell.
• Datenfluss	• Input und Output treten als wiederkehrende Serie auf, und es gibt einen direkten Zusammenhang zwischen entsprechenden Teilen jeder Serie.
– azyklisch	– … und die Transformationen beinhalten keine Feedback Loops.
– nur Fan-out-Komponenten	– … und die Transformationen beinhalten keine Feedback Loops und ein Input führt jeweils zu mehr als einem Output.
– Pipeline, Pipe-and-Filter	– Die Berechnung besteht aus Transformationen von kontinuierlichen Datenströmen. Die Transformationen sind inkrementell; eine Transformation kann beginnen, bevor der vorangehende Schritt abgeschlossen ist.
• Closed Loop Control	Das System überwacht andauernde Aktivität, ist in einem physikalischen System eingebettet und ist unvorhersehbaren externen Störungen ausgesetzt, sodass vordefinierte Algorithmen fehlschlagen können.

Faustregeln für Call-and-Return

Call-and-Return	Die Reihenfolge der Berechnung ist fixiert und die Komponenten können ohne Ergebnisse von anderen Komponenten keine sinnvolle Weiterverarbeitung durchführen.
• Objektorientiert, abstrakte Datentypen	• Die gesamte Modifizierbarkeit ist eine wesentliche Qualitätsanforderung. • Die Integrierbarkeit (über die Schnittstellen) ist ebenfalls eine wesentliche Qualitätsanforderung.
– *Abstrakte Datentypen* – *Objekte*	– Es gibt zahlreiche Datentypen, deren Repräsentation sich ändern kann/wird. – Das Ausnützen von Gemeinsamkeiten über die Schnittstellen von Komponenten (Klassen) durch das *information hiding*-Prinzip bringt Vorteile für Entwicklung und Testen des Systems.
– *Call-and-Return-basierter Client-Server*	– Die Modifizierbarkeit hinsichtlich der Produktion von Daten und wie diese konsumiert werden, ist wesentlich.
• Schichtung (*layering*)	• Die Aufgaben im System können aufgeteilt werden auf jene, die spezifisch für die Anwendung sind, und jene, die generisch für viele Anwendungen, aber spezifisch für die vorliegende Hardware sind. • Portabilität über Plattformen hinweg ist wesentlich. • Bereits existierende Infrastrukturen können verwendet werden (z.B. Betriebssystem, Netzwerk-Management etc.)

Faustregeln für unabh Komp

Unabhängige Komponenten	Das System soll auf einer Multiprozessor-Plattform lauffähig sein. Das System kann als eine Menge von lose gekoppelten Komponenten strukturiert werden (i.e. eine Komponente kann weitgehend unabhängig vom Zustand anderer Komponenten weiterarbeiten). *Performance tuning* durch Arbeitsaufteilung (auf Prozesse oder Prozessoren) ist wesentlich.
• Kommunizierende Prozesse – *Lightweight-Prozesse* – *Verteilte Objekte* – *Client-Server Request-Reply* – *Broadcast* – *Token passing*	• *Message passing* (Nachrichtenaustausch) ist als Interaktionsmechanismus ausreichend. – Der Zugriff auf gemeinsame Daten ist kritisch für das Erreichen der Performanz-Ziele. – Die bekannten Gründe für Objektorientierung und interagierende Prozesse treffen zu. – Aufgaben können zwischen Produzenten und Konsumenten von Daten bzw. zwischen Aufrufern und Aufgerufenen geeignet aufgeteilt werden. – Alle Komponenten müssen von Zeit zu Zeit synchronisiert werden. – Verfügbarkeit ist eine wesentliche Anforderung. – Es ist für alle Aufgaben des Systems sinnvoll, zwischen den Komponenten in einem vollständig verbundenen Graphen zu kommunizieren. – Der Gesamtzustand des Systems muss gelegentlich zugreifbar sein (wie in einem fehlertoleranten System) und die Komponenten arbeiten asynchron.
• Dezentralisierte Server	• Verfügbarkeit und Fehlertoleranz sind die wesentlichen Anforderungen und die Daten und Dienste der Server sind kritisch für die Funktionalität des Systems.
• Replizierte Worker	• Die Berechnungen können im Teile-und-Herrsche(*divide et impera*)-Ansatz durch Parallelverarbeitung durchgeführt werden.
• Event-Systeme	• Die Konsumenten von Events (Informationen) sollen von deren Erzeugern entkoppelt werden. • Skalierbarkeit soll durch Hinzufügen von weitgehend eigenständigen Komponenten, die durch Events angestoßen werden, erreicht werden.

Faustregeln für Daten zentriert

Daten zentriert	Zentraler Punkt ist die Speicherung, die Repräsentation, das Management sowie der Zugriff auf große Mengen von langlebigen Daten im System.
• Transaktionale Datenbank / Repository	• Die Reihenfolge der Ausführung von Komponenten wird bestimmt durch die ankommenden Anfragen, um Daten zuzugreifen oder zu aktualisieren. Die Daten sind dabei tief strukturiert.
• Blackboard	• Skalierbarkeit soll durch Hinzufügen von Datenkonsumenten ohne Änderung der Produzenten erfolgen. • Modifizierbarkeit soll durch die Änderung, wer Daten produziert und wer konsumiert, erreicht werden.

Final comment

Success comes from wisdom.
Wisdom comes from experience.
Experience comes from mistakes.

RMI

Warum RMI?

- Protokolle wie UDP oder TCP können zur Kommunikation zwischen Rechnern verwendet werden.
  - Warum also eine neue Kommunikationstechnik?
- UDP und TCP sind „Low-Level“ Protokolle. Der Programmierer muß sich um viele Aufgaben selbst kümmern.
- Das Ziel von RMI ist es, dem Programmierer die Implementierung von Client/Server Applikationen zu erleichtern.
- RMI ist eine „High-Level“ Netzwektechnologie.

Aufgaben von Client und Server

- Der Client schickt einen Request an den Server.
  - Der Client bereitet den Request und die dafür notwendigen Daten auf, damit sie über das Netz versendet werden können.
  - Der Client sendet den Request an den Server.
  - Der Server parst die einlangende Nachricht und stellt fest um welchen Request es sich handelt.
- Der Server bearbeitet den Request und schick die Response zurück and den Client.
  - Der Server bereitet die Response auf, damit sie über das Netz versendet werden kann.
  - Der Server sendet die Response an den Client.
  - Der Client parst die Response und stellt fest, was damit zu tun ist.
- Der Client verarbeitet die Antwort (z.B. stellt sie am Bildschirm dar).

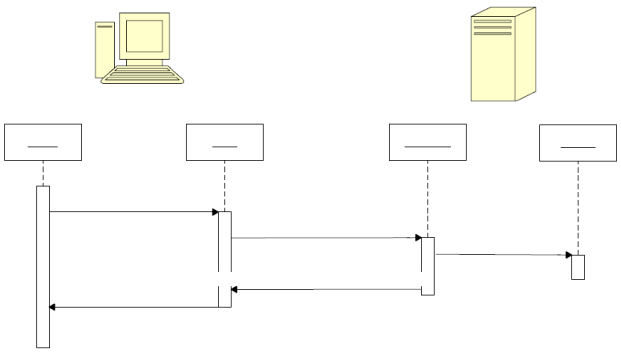
=>RMI erleichtert diese Aufgaben.

RMI Remote Method Invocation

- RMI erlaubt es Methode eines Objekts aufzurufen, das auf einem anderen Rechner liegt.
  - Die Parameter zum Methodenaufruf müssen dazu zu dem anderen Rechner übertragen
  - werden.
  - Das Remote Object wird informiert, dass eine Methode aufgerufen werden soll.
  - Der Return Value wird zum Aufrufenden Java Programm zurück geliefert.

=>RMI übernimmt diese Aufgaben.

Stubs and
Parameter Marshalling



- Die Stub Methode auf der Client Seite erzeugt einen Datenblock bestehend aus:
  - Einem Identifier für das Remote Object das verwendet werden soll.
  - Die Beschreibung der aufzurufenden Methode.
  - Die marshalled Parameter.

- Der Stub sendet den Datenblock an den Server. Der Receiver auf der Server Seite führt folgende Schritte aus:
  - Unmarshalled die Parameter.
  - Lokalisiert das Object auf dem die Methode ausgeführt werden soll.
  - Führt den gewünschten Methodenaufruf durch.
  - Übernimmt den Return Value oder die Exception und marshalled sie.
  - Sendet ein Datenpaket zurück, das den Return Value oder die Exception enthält.
- Der Client Stub unmarshalled den Return Value oder die Exception und gibt diese Als Rückgabewert an den Client weiter.

Marshalling

- Die Parameter müssen deviceindependend übertragen werden.
  - Zahlen werden immer mit big-endian byte ordering übertragen.
  - Objekte werden serialisiert übertragen.
- Der Prozess des Encodens der Parameter wird Parameter Marshalling genannt.

Aufrufsemantik

- **Call by Reference**
  - Bei einem Methodenaufruf wird nur die Referenz eines Objects als Parameter übergeben
- **Call by Value**
  - Bei einem Methodenaufruf wird eine Kopie des Objects als Parameter übergeben.

=>RMI verwendet Call by Value.

Remote Interface

```
import java.rmi.Remote;
import java.rmi.RemoteException;

// Interface shared by client and server.
public interface Product extends Remote {
        public String getDescription()
                throws RemoteException;
}
```

RMI Server Implementierung

```
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;

public class ProductImpl extends UnicastRemoteObject implements Product {
// The class is a server since it extends
// UnicastRemoteObject, which
// makes the object remotely accessible.

        private String descr;

        protected ProductImpl(String d) throws RemoteException {
                descr = d;
        }
        public String getDescription() throws RemoteException {
                return "I am a " + descr + ". Buy me!";
        }
}
```

RMIC

- RMIC Aufruf:
- rmic –v1.2 –d classes ProductImpl
- RMIC wir auf die Klasse aufgerufen, die das Remote Interface implementiert und vom UnicastRemoteObject erbt.
- -d classes ist optional und gibt an in welchem Directory das generierte Stub File abgelegt werden soll.
- Die generierte Stub Klasse hat den Namen ProductImpl_stub.class
- Diese Klasse muss auf den Client kopiert werden.

Locating Server Objects

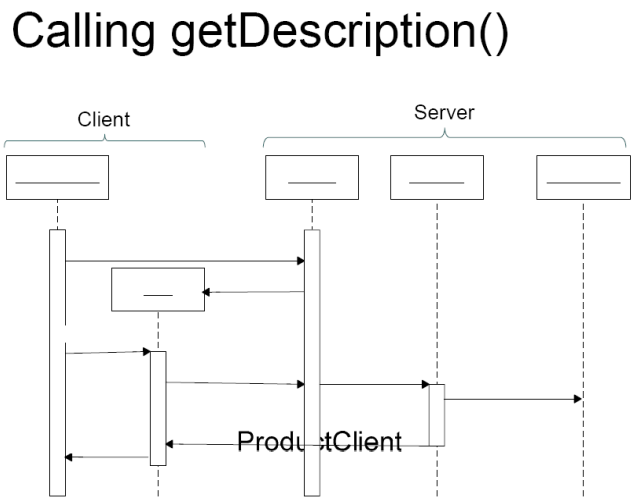
- Um auf ein Remote Object auf dem Server zuzugreifen, braucht der Client ein lokales Stub Object.
  - Wo bekommt er das her?
- Die Java Bibliothek bietet ein *bootstrap registry service* um ein Remote Object zu lokalisieren.
- Ein Server Programm registriert Objekte beim bootstrap registry service und der Client kann von dort Stubs von den registrierten Objekten beziehen.
- Server
  - *ProductImpl p1 = new ProductImpl("The best Toaser");*
  - *Naming.bind("toaster", p1);*
- Client
  - *Product p = (Product)*
  - *Naming.lookup("rmi://example.com:1234/toaster");*

Registrieren von Objekten

```
public class ProductServer {
        public static void main(String[] args) {
                try {
                        ProductImpl p1 = new ProductImpl("The best Toaster");
                        ProductImpl p2 = new ProductImpl("The best Microwave");

                        try {
                        Registry localRegistry = LocateRegistry.createRegistry(1099);
                        } catch (Exception e) { System.out.println(e); }

                        //register the products
                        Naming.bind("toaster", p1);
                        Naming.bind("microwave", p2);

                        System.out.println("Waiting ...");
                        // Waiting from invocations from clients
                        // In the UnicastRemoteObject a separate Thread is started.
                } catch (Exception e) {System.out.println("Error: " + e);}
        }
}
```

RMI Client Implementierung

```
import java.rmi.Naming;

public class ProductClient {
	public static void main(String[] args) {
		String url = "rmi://localhost:1099/";
		Try {
			Product p1 = (Product) Naming.lookup(url +"toaster");
			Product p2 = (Product) Naming.lookup(url +"microwave");

			System.out.println(p1.getDescription());
			System.out.println(p2.getDescription());
		} catch (Exception e) {
			e.printStackTrace();
		}
	}
}
```



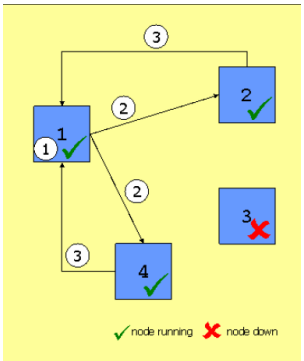
Zusammenfassung RMI

- Mit der Hilfe von RMI können Methoden von Objekten aufgerufen werde, die auf anderen Rechner liegen.
- Der Stub und der Receiver kümmern sich um das marshallen/unmarshallen der Parameter und Rückgabewerte.
- RMI funktioniert nur wenn Client und Server in Java implementiert sind.

ShareMe Überblick :

Task 2 RMI Client/Server

Die Suche nach Files erfolgt über RMI. Jeder Peer implementiert einen RMI Client und Server. Die Informationen wie auf den RMI Server zugegriffen werden kann ist in der IsAlive Nachricht von Punkt 1 gespeichert.



Task 2.1: Durchsuchen von anderen Peers

- Durchlaufen der HostList in einer Schleife.
- Für alle Peers in der HostList einen RMI lookup und ein Aufruf der search() Methode.
- Die einzelnen Suchergebnisse zu einem Gesamtergebnis zusammenfassen.

Taks 2.2: FileBase zum Durchsuchen anbieten

- Die *SerachEngineImpl* Klasse implementiert den RMI Server.
- Das RMI Interfece heißt *ISearchEngine*.
- Hilfskassen werden verwendet um die Festplatte zu durchsuchen.

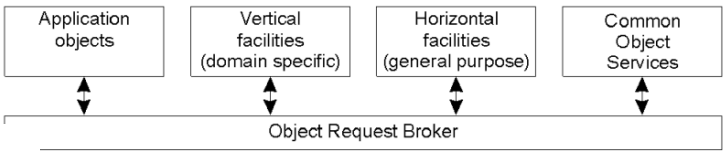
CORBA

- **CORBA: C**ommon **O**bject **R**equest **B**roker **A**rchitecture
- **Background:**
  - Developed by the **Object Management Group (OMG)** in response to industrial demands for object-based middleware
  - Currently in version #2.4 with #3 (almost) done
  - CORBA is a *specification*: different implementations of CORBA exist
  - Very much the work of a committee: there are over 800 members of the OMG and many of them have a say in what CORBA should look like
- **Essence:** CORBA provides a simple distributed-object model, with specifications for many supporting services

=>it may be here to stay (for a couple of years)

Overview of CORBA

- The global architecture of CORBA



- CORBA reference model consists of 4 architectural elements connected by the **Object Request Broker (ORB)**
  - The ORB forms the core of any CORBA distribution (library)
  - it is responsible for enabling communication between objects and their clients while hiding issues of distribution and heterogeneity.

CORBA facilities

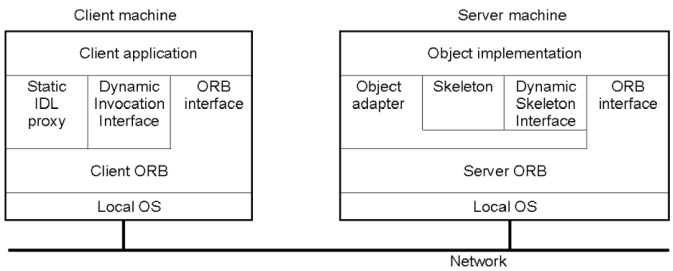
- The references also distinguishes CORBA facilities
  - facilities are constructed as compositions of CORBA services
  - horizontal facilities: general-purpose high-level services independent of application domains (for user interfaces; information, system and task management)
  - vertical facilities: targeted to a specific application domain such as e-commerce, banking, etc.

CORBA IDL

- CORBA uses the remote-object model, the implementation of an object resides in the server
- Objects and services are specified in the **CORBA Interface Definition Language (IDL)**
  - CORBA IDL is similar to other IDLs:
    - it provides a precise syntax for expressing methods and their parameters, but it is not possible to describe semantics
    - an interface is a collection of methods, and objects specify which interfaces they implement.
    - Interface specifications can only be given by means of IDL
- For CORBA, exact rules for mapping of IDL specifications to existing programming languages is necessary
  - At present: C, C++, Java, Smalltalk, Ada, COBOL.

Object Model

- The general organization of a CORBA system



CORBA Overview

- Object Request Broker (ORB): CORBA's object broker that connects clients, objects, and services
- Proxy/Skeleton: Precompiled code that takes care of (un)marshaling invocations and results
- Dynamic Invocation/Skeleton Interface (DII/DSI): To allow clients to "construct" invocation requests at runtime instead of calling methods at a proxy, and having the server-side "reconstruct" those request into regular method invocations
- Object adapter: Server-side code that handles incoming invocation requests.
- Interface repository: Database containing interface definitions and which can be queried at runtime
- Implementation repository: Database containing the implementation (code, and possibly also state) of objects. Effectively: a server that can launch object servers.

CORBA Object Model

- **Essence:** CORBA has a "traditional" remote-object model in which an object residing at an object server is remote accessible through proxies
- **Observation:** All CORBA specifications are given by means of interface descriptions, expressed in an IDL.
- CORBA follows an interface-based approach to objects:
  - Not the objects, but interfaces are the really important entities
  - An object may implement one or more interfaces
  - Interface descriptions can be stored in an interface repository, and looked up at runtime
  - Mappings from IDL to specific programming are part of the CORBA specification (languages include C, C++, Smalltalk, Cobol, Ada, and Java.

CORBA Services

Service	Description
Collection	Facilities for grouping objects into lists, queue, sets, etc.
Query	Facilities for querying collections of objects in a declarative manner
Concurrency	Facilities to allow concurrent access to shared objects
Transaction	Flat and nested transactions on method calls over multiple objects
Event	Facilities for asynchronous communication through events
Notification	Advanced facilities for event-based asynchronous communication
Externalization	Facilities for marshaling and unmarshaling of objects
Life cycle	Facilities for creation, deletion, copying, and moving of objects
Licensing	Facilities for attaching a license to an object
Naming	Facilities for systemwide name of objects
Property	Facilities for associating (attribute, value) pairs with objects
Trading	Facilities to publish and find the services on object has to offer
Persistence	Facilities for persistently storing objects
Relationship	Facilities for expressing relationships between objects
Security	Mechanisms for secure channels, authorization, and auditing
Time	Provides the current time within specified error margins

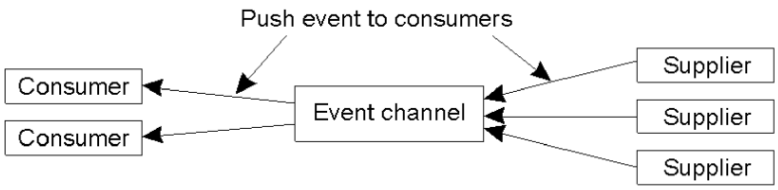
Object Invocation Models

- **Object invocations:** CORBA distinguishes three different forms of direct invocations:

Request type	Failure semantics	Description
Synchronous	At-most-once	Caller blocks until a response is returned or an exception is raised
One-way	Best effort delivery	Caller continues immediately without waiting for any response from the server
Deferred synchronous	At-most-once	Caller continues immediately and can later block until response is delivered

Event and Notification Services (1)

- Event communication: There are also additional facilities by means of event channels
- The logical organization of suppliers and consumers of events, following the push-style model.

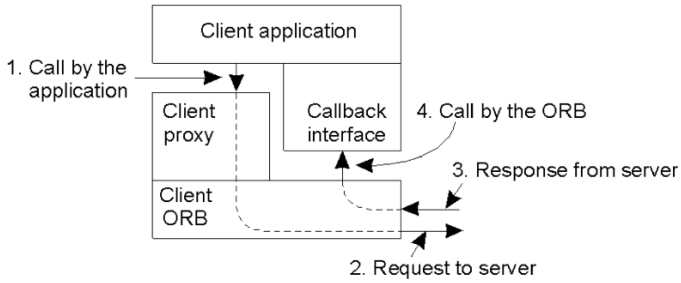


Event and Notification Services (2)

- The pull-style model for event delivery in CORBA

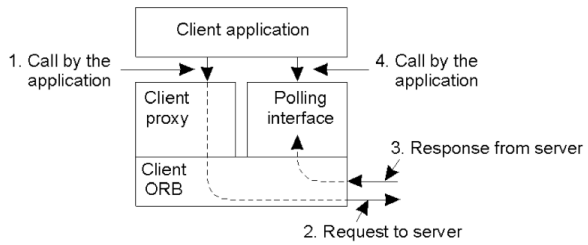
Messaging (1)

- CORBA's callback model for asynchronous method invocation
- Messaging facilities: reliable asynchronous and persistent method invocations



Messaging (2)

- CORBA'S polling model for asynchronous method invocation.



Interoperability

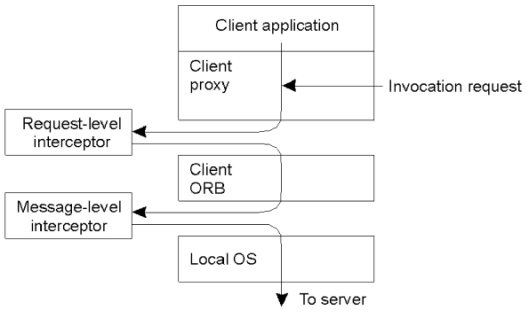
- General Inter-ORB Protocol (GIOP) message types
- Realization on top of TCP: Internet Inter-ORB Protocol (IIOP)

Message type	Originator	Description
Request	Client	Contains an invocation request
Reply	Server	Contains the response to an invocation
LocateRequest	Client	Contains a request on the exact location of an object
LocateReply	Server	Contains location information on an object
CancelRequest	Client	Indicates client no longer expects a reply
CloseConnection	Both	Indication that connection will be closed
MessageError	Both	Contains information on an error
Fragment	Both	Part (fragment) of a larger message

- A mechanism to use the proxies as generated by an IDL compiler in connection with an existing client-side ORB
- An interceptor is
  - a mechanism by which an invocation can be intercepted on its way from client to server, and adapted as necessary before letting it continue.
  - a piece of code that modifies an invocation request on its way from client to server and accordingly adapts the associated response

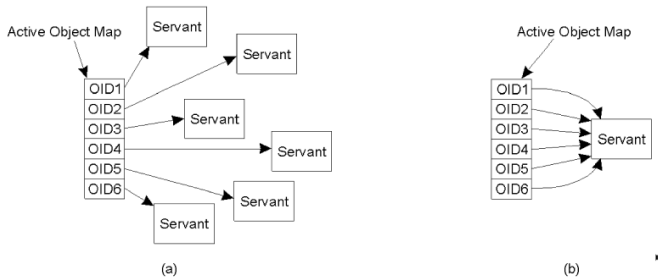
Logical placement of Interceptors

- Request-level: Allows one to modify invocation semantics (e.g., multicasting)
- Message-level: Allows one to control message-passing between client and server (e.g., handle reliability and fragmentation)



Portable Object Adaptor (1)

- The Portable Object Adaptor (POA)
  - is a component that makes server-side code appear as CORBA objects to clients (i.e. a wrapper) and therefore allows to write server-side code independently of a specific ORB
- Mapping of CORBA object identifiers to servants.
  - The POA supports multiple servants (a)
  - The POA supports a single servant (b)



(c) H.Gall, 2005

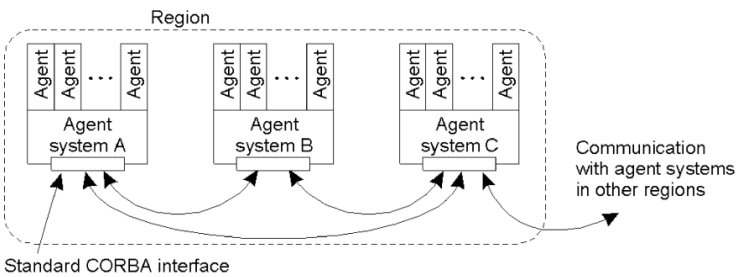
Portable Object Adaptor (2)

- Changing a C++ object into a CORBA object
- The oid returned by activate_object() is generated by the POA

```
My_servant *my_object;       // Declare a reference to a C++ object
CORBA::Objectid_var oid;     // Declare a CORBA identifier

my_object = new MyServant; // Create a new C++ object (= a servant)

// Now Register C++ object as CORBA OBJECT at the POA
oid = poa->activate_object (my_object);
```

Agents

- CORBA's overall model of agents, agent systems, and regions

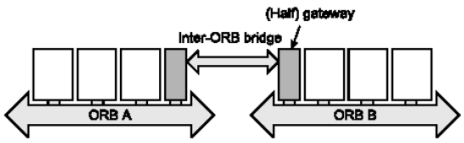


Naming

- **Important:** In CORBA, it is essential to distinguish specification-level and implementation-level object references
- **Specification level:** An object reference is considered to be the same as a proxy for the referenced object □□ having an object reference means you can directly invoke methods; there is no separate client-to-object binding phase
- **Implementation level:** When a client gets an object reference, the implementation ensures that, one way or the other, a proxy for the referenced object is placed in the client's address space:
  - ObjectReference objRef;
  - objRef = bindTo (object O in Server S at host H);
- **Conclusion:** Object references in CORBA used to be highly **implementation dependent**: different implementations of CORBA could normally not exchange their references.

Interoperable Object References (1/3)

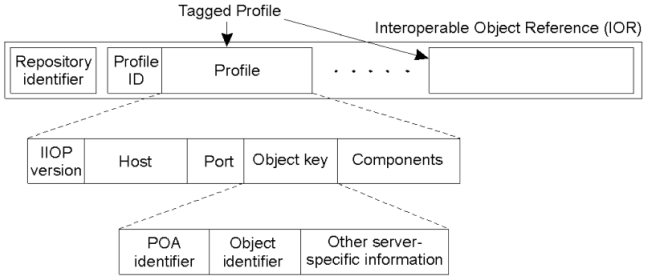
- **Observation:** Recognizing that object references are implementation dependent, we need a separate referencing mechanism to cross ORB boundaries
- **Solution:** Object references passed from one ORB to another are transformed by the bridge through which they pass (different transformation schemes can be implemented)



- **Observation:** Passing an object reference `refA` from ORB A to ORB B circumventing the A-to-B bridge may be useless if ORB B doesn't understand `refA`

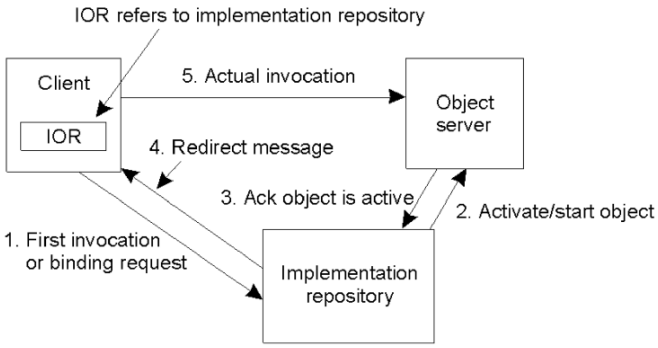
Interoperable Object References (2/3)

- The organization of an IOR with specific information for IIOP
- To allow all kinds of different systems to communicate, we standardize the reference that is passed between bridges



Object References (3/3)

- Indirect binding in CORBA
  - a binding request is first sent to an implementation repository (acting as a registry to locate an activate an object before sending invocation requests to it).



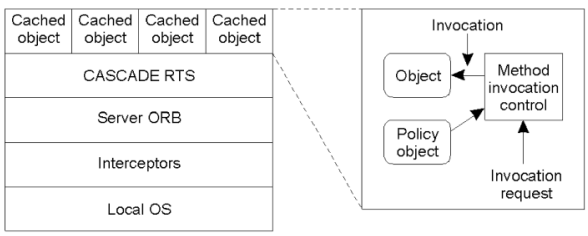
(c) H.Gall,

Naming Service

- **Essence:** CORBA's naming service allows servers to associate a name to an object reference, and have clients subsequently bind to that object by resolving its name
- **Observation:** In most CORBA implementations, object references denote servers at specific hosts; naming makes it easier to relocate objects
- **Observation:** In the naming graph all nodes are objects; there are no restrictions to binding names to objects CORBA allows arbitrary naming graphs
- **Question:** How do you imagine cyclic name resolution stops?
- **Observation:** There is no single root; an initial context node is returned through a special call to the ORB.
- Also: the naming service can operate *across* different ORBs => **interoperable naming service**

Caching and Replication

- No CORBA support for generic caching and replication
  - CASCADE system offers a caching service implemented as a large collection of object servers, each referred to as Domain Caching Server (DCS)
  - Each DCS is an object server running on a CORBA ORB
  - The collection of DCSs may be spread across WANs (or the Internet)
  - Each cached object has its own Policy Object to control method invocations

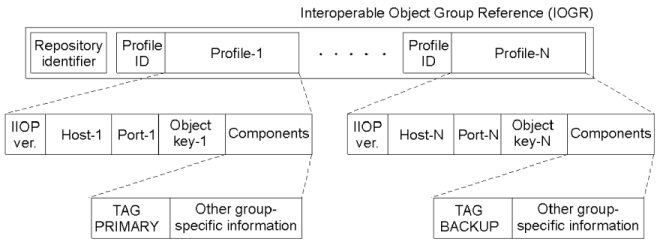


(c) H.Gall, 2005

2

Object Groups

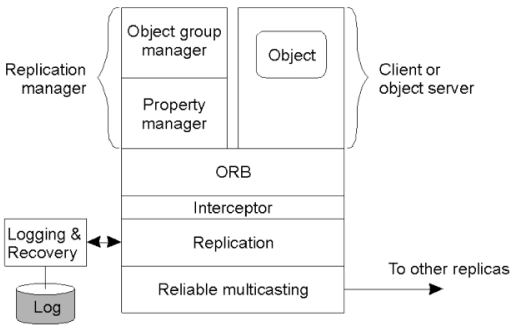
- **Essence:** Mask failures through replication, by putting objects into **object groups**. Object groups are transparent to clients: they appear as "normal" objects.
- This approach requires a separate type of object reference: **Interoperable Object Group Reference**



- A possible organization of an IOGR for an object group having a primary and backups.

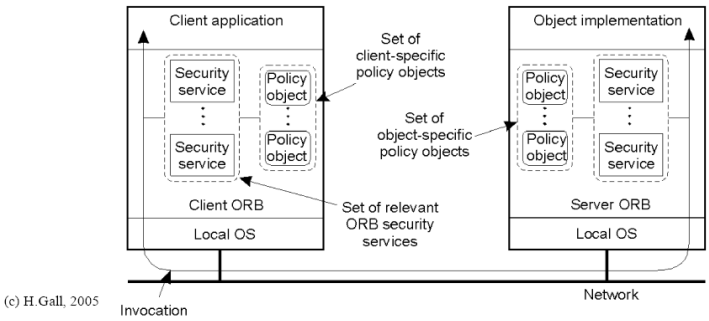
An Example Architecture

- An example architecture of a fault-tolerant CORBA system



Security (1)

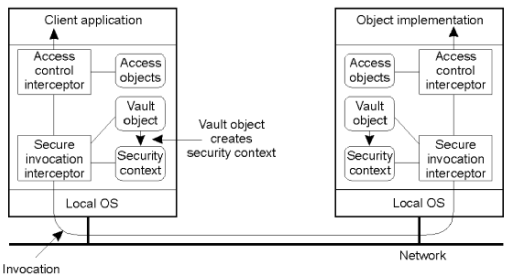
- The general organization for secure object invocation in CORBA
- Allow the client and object to be mostly unaware of all the security policies, except perhaps at binding time; the ORB does the rest. Specific policies are passed to the ORB as (local) objects and are invoked when necessary



(c) H.Gall, 2005

Security (2)

- The role of security interceptors in CORBA
  - The access control interceptor is a request-level interceptor that checks the access rights associated with an invocation.
  - A secure invocation interceptor takes care of implementing the message protection (interceptor encrypts requests and responses for integrity and confidentiality).

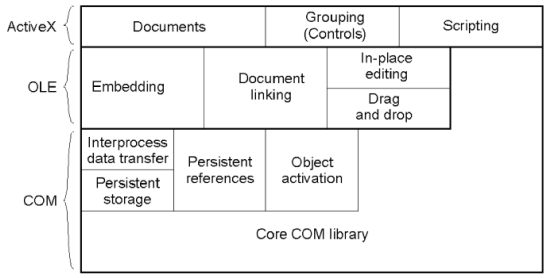


DCOM

- **DCOM** = Distributed Component Object Model
- Microsoft's solution to establishing inter-process communication, possibly across machine boundaries.
- Supports a primitive notion of distributed objects
- Evolved from early Windows versions to current NT-based systems (including Windows 2000)
- Comparable to CORBA's object request broker
- (ORB)

Overview of DCOM

- DCOM is related to many things that have been introduced by Microsoft in the past couple of years:



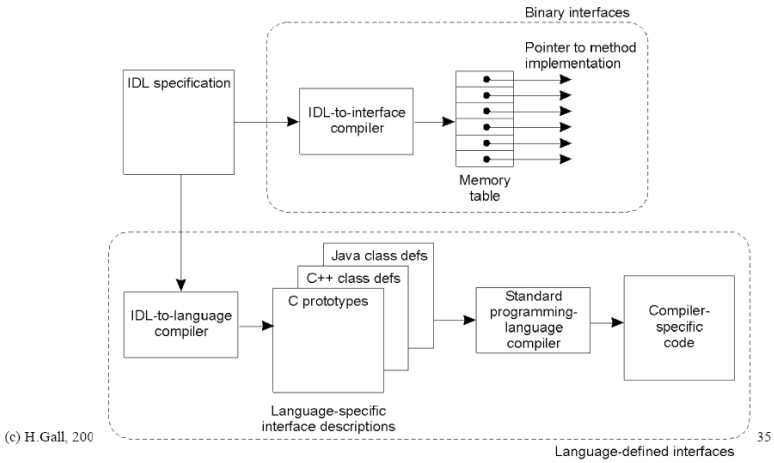
- **DCOM**: Adds facilities to communicate across process and machine boundaries.

COM Object Model

- An interface is a collection of semantically related operations
- Each interface is typed, and therefore has a globally unique **interface identifier**
- A client always requests an implementation of an interface:
  - Locate a class that implements the interface
  - Instantiate that class, i.e., create an object
  - Throw the object away when the client is done

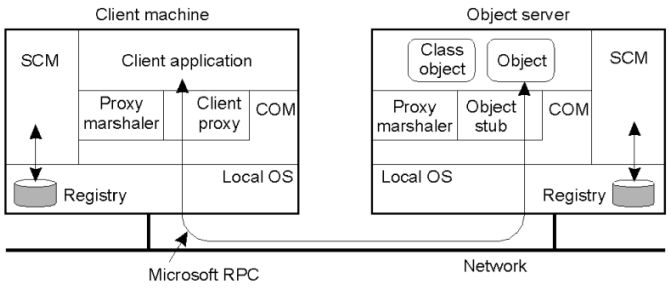
Object Model

- The difference between language-defined and binary interfaces



Type Library and Registry

- The overall architecture of DCOM
- SCM: Service Control Manager, responsible for activating objects (cf., to CORBA's implementation repository).
- Proxy marshaler: handles the way that object references are passed between different machines



DCOM Services

- Overview of DCOM services in comparison to CORBA services

CORBA Service	DCOM/COM+ Service	Windows 2000 Service
Collection	ActiveX Data Objects	-
Query	None	-
Concurrency	Thread concurrency	-
Transaction	COM+ Automatic Transactions	Distributed Transaction Coordinator
Event	COM+ Events	-
Notification	COM+ Events	-
Externalization	Marshaling utilities	-
Life cycle	Class factories, JIT activation	-
Licensing	Special class factories	-
Naming	Monikers	Active Directory
Property	None	Active Directory
Trading	None	Active Directory
Persistence	Structured storage	Database access
Relationship	None	Database access
Security	Authorization	SSL, Kerberos
Time	None	None

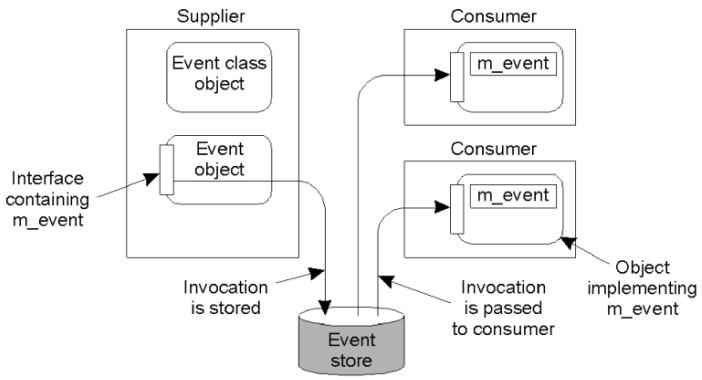
37

Communication Models

- **Object invocations:**
  - Synchronous remote-method calls with at-most-once semantics.
  - Asynchronous invocations are supported through a polling model, as in CORBA.
- **Event communication:**
  - Similar to CORBA's push-style model
- **Messaging:**
  - Completely analogous to CORBA messaging

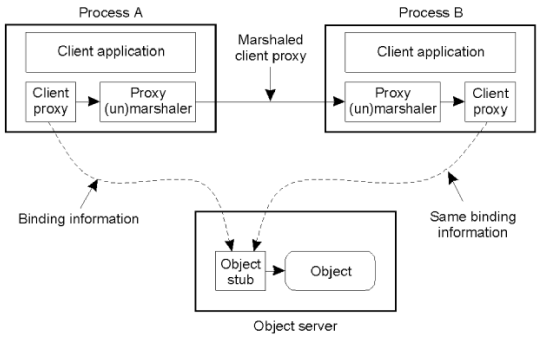
Communication: Events

- Event processing in DCOM



Processes: Passing Object References

- Objects are referenced by means of a local interface pointer. The question is how such pointers can be passed between different machines·



- Passing an object reference in DCOM with custom marshaling.

Naming: Monikers (1/3)

- DCOM can handle only objects as temporary instances of a class. To accommodate objects that can outlive their client, something else is needed.
- **Moniker:** A hack to support real objects
  - A moniker associates data (e.g., a file), with an application or program
  - Monikers can be stored
  - A moniker can contain a **binding protocol**, specifying how the associated program should be "launched" with respect to the data.

Naming: Monikers (2/3)

- DCOM-defined moniker types:

Moniker type	Description
File moniker	Reference to an object constructed from a file
URL moniker	Reference to an object constructed from a URL
Class moniker	Reference to a class object
Composite moniker	Reference to a composition of monikers
Item moniker	Reference to a moniker in a composition
Pointer moniker	Reference to an object in a remote process

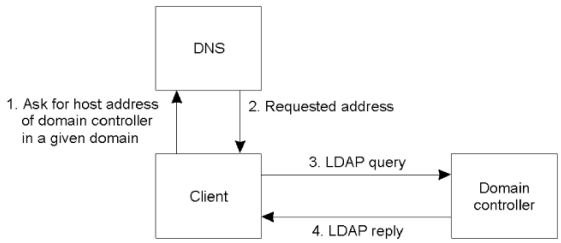
Naming: Monikers (3/3)

- Binding to a DCOM object by means of file moniker.

Step	Performer	Description
1	Client	Calls BindMoniker at moniker
2	Moniker	Looks up associated CLSID and instructs SCM to create object
3	SCM	Loads class object
4	Class object	Creates object and returns interface pointer to moniker
5	Moniker	Instructs object to load previously stored state
6	Object	Loads its state from file
7	Moniker	Returns interface pointer of object to client

Active Directory

- A worldwide distributed directory service, but it does not provide location transparency.
- **Basics:** Associate a directory service (called **domain controller**) with each domain; look up the controller using a normal DNS query:



- Note: Controller is implemented as an LDAP server

Fault Tolerance

- **Automatic transactions:** Each class object (from which objects are created), has a transaction attribute that determines how its objects behave as part of a transaction:

Attribute value	Description
REQUIRES_NEW	A new transaction is always started at each invocation
REQUIRED	A new transaction is started if not already done so
SUPPORTED	Join a transaction only if caller is already part of one
NOT_SUPPORTED	Never join a transaction
DISABLED	Never join a transaction, even if told to do so

- Transaction attribute values for DCOM objects.

Declarative Security (1)

- **Declarative security:**
  - Register per object what the system should enforce with respect to authentication.
  - Authentication is associated with users and user groups.
  - There are different authentication levels in DCOM:

Authentication level	Description
NONE	No authentication is required
CONNECT	Authenticate client when first connected to server
CALL	Authenticate client at each invocation
PACKET	Authenticate all data packets
PACKET_INTEGRITY	Authenticate data packets and do integrity check
PACKET_PRIVACY	Authenticate, integrity-check, and encrypt data packets

Declarative Security (2)

- **Delegation:**
  - A server can impersonate a client depending on a level:

Impersonation level	Description
ANONYMOUS	The client is completely anonymous to the server
IDENTIFY	The server knows the client and can do access control checks
IMPERSONATE	The server can invoke local objects on behalf of the client
DELEGATE	The server can invoke remote objects on behalf of the client

- Impersonation levels in DCOM

Programmatic Security

- There is also support for programmatic security by which security levels can be set by an application, as well as the required security services

Service	Description
NONE	No authentication
DCE_PRIVATE	DCE authentication based on shared keys
DCE_PUBLIC	DEC authentication based on public keys
WINNT	Windows NT security
GSS_KERBEROS	Kerberos authentication

(a) Default authentication services supported in DCOM

Service	Description
NONE	No authorization
NAME	Authorization based on the client's identity
DCE	Authorization using DEC Privilege Attribute Certificates (PACs)

(b) Default authorization services supported in DCOM

Summary (1)

- Comparison of CORBA, DCOM, and Globe

Issue	CORBA	DCOM	Globe
Design goals	Interoperability	Functionality	Scalability
Object model	Remote objects	Remote objects	Distributed objects
Services	Many of its own	From environment	Few
Interfaces	IDL based	Binary	Binary
Sync. communication	Yes	Yes	Yes
Async. communication	Yes	Yes	No
Callbacks	Yes	Yes	No
Events	Yes	Yes	No
Messaging	Yes	Yes	No
Object server	Flexible (POA)	Hard-coded	Object dependent
Directory service	Yes	Yes	No
Trading service	yes	No	No

Summary (2)

- Comparison of CORBA, DCOM, and Globe.

Issue	CORBA	DCOM	Globe
Naming service	Yes	Yes	Yes
Location service	No	No	Yes
Object reference	Object's location	Interface pointer	True identifier
Synchronization	Transactions	Transactions	Only intra-object
Replication support	Separate server	None	Separate subobject
Transactions	Yes	Yes	No
Fault tolerance	By replication	By transactions	By replication
Recovery support	Yes	By transactions	No
Security	Various mechanisms	Various mechanisms	More work needed

User Datagram Protocol UDP

- Das User Datagram Protocol (UDP) ist ein minimales, verbindungsloses Netzwerkprotokoll.
- Verbindungslos bedeutet, dass nicht erst eine Verbindung zum Gegenüber aufgebaut wird (mittels Handshaking wie bei TCP), sondern dass sofort die Daten zu der Gegenstelle geschickt werden.
- Es wird nicht garantiert, dass ein einmal gesendetes Paket ankommt oder dass Pakete in der gleichen Reihenfolge ankommen, in der sie gesendet wurden; eine Quittierung ist nicht vorgesehen.
- Die Daten können sofort aus dem empfangenen *DatagramPacket* als *byte* Array ausgelesen werden.

Transmission Control Protocol TCP

- Das Transmission Control Protocol (TCP) ist ein zuverlässiges, verbindungsorientiertes Transportprotokoll.
- Beim Aufbau einer TCP-Verbindung kommt der so genannte Drei-Wege-Handshake zum Einsatz.
- TCP stellt einen virtuellen Kanal zwischen zwei Endpunkten (Sockets) her.
- Auf diesem Kanal können in beide Richtungen Daten übertragen werden.

TCP in Java

- Um einen TCP Server zu implementieren verwendet man einen *ServerSocket* und gibt an, an welchem Port der Server lauschen soll.
- Ein *.accept()* blockiert den Prozess bis es eine eingehende Verbindung gibt.
- *.accept()* liefert einen *Socket* zurück über den dann die Kommunikation abläuft.
- Da es sich bei TCP um ein verbindungsorientiertes Protokoll handelt, erfolgt die Kommunikation nicht über *DatagramPackets* wie bei UDP, sondern über *Streams*.
- Da Daten in beide Richtungen übertragen werde können, erhält man von einem *Socket* einen InputStream und eine OutputStream.

TCP Beispiel

- Schreiben Sie einen TCP Server der eingehende Zeichen zeilenweise auf der Console ausgibt. Empfängt der Server den String „exit“ in einer Zeile, soll die Verbindung zum Client geschlossen werden. Das Server soll in einer Endlosschleife laufen.

- Erweiterung: Der empfangene String soll wieder and den Client zurück geschickt werden.

TCP Beispiel Lösung

Zeilenweises lesen von einem Stream:

- InputStream is = socket.getInputStream();
- InputStreamReader isr = new InputStreamReader(is);
- BufferedReader br = new BufferedReader(isr);

Zeilenweises Schreiben auf eine Stream:

- OutputStream os = socket.getOutputStream();
- BufferedOutputStream bos =
- new BufferedOutputStream(os);
- PrintWriter pw = new PrintWriter(bos);

!!!!!!!pw.flush() nicht vergessen!!!!!!!!

TCP Client

Will man einen TCP Client implementieren, braucht man keinen *ServerSocket*.
Einem Socket mit dem Constructor erzeugen und den OutputStream zum Schreiben nutzen.

- Socket socket = new Socket(host, port);
- OutputStream os =socket.getOutputStream();

Socket schließen nicht vergessen!

Multithreaded TCP Server

- Singlethreaded Server:
  - Anfragen können nur nach einander bearbeitet werden. Dauert die Bearbeitung der Anfrage länger (z.B. wegen blockierender System Calls (IO)) können keine weitern Requests bearbeitet werden. Der Server ist blockiert.
- Multithreaded Server
  - Dispather Thread
    - Wartet auf eingehende Verbindungen und leitet die eingehenden Anfragen an einen Worker Thread weiter.
  - Worker Thread
    - Bearbeitet die eigentliche Anfrage.

run() im Dipatcher Thread

```
public void run() {
        while(keepRunning) {
                Socket socket = null;
                try {
                        socket = sSocket.accept();

                        // start new worker thread
                        WorkerThread worker = new WorkerThread(socket);
                        Thread workerTread = new Thread(worker);
                        workerThread.start();

                } catch (InterruptedIOException iioe) {
                        // ignore this - it was done on purpose by the
                        timeout
                }
        }
        sSocket.close();
}
```

Worker Thread

```
public class WorkerThread implements Runnable {
	privat Socket socket = null;
	privat BufferedReader br = null;
	privat PrintWriter pw = null;
	privat BufferedOutputStream bos = null;

	public WorkerThread(Socket client) throws ShareMeException {
		this.socket = client;

		try {
		br = new BufferedReader(new
		InputStreamReader(socket.getInputStream()));
		bos = new BufferedOutputStream(socket.getOutputStream());
		pw = new PrintWriter(bos);

		} catch (Exception e) {
			e.printStackTrace();
		}
	}

	public void run() {
		// do the work; read form input stream; write to output stream
		socket.close();
	}
}
```

This document was created with Win2PDF available at http://www.daneprairie.com.
The unregistered version of Win2PDF is for evaluation or non-commercial use only.