Verteilte Systeme

...für C++ Programmierer

Kommunikation

by

Dr. Günter Kolousek

Kommunikation - Prinzip

- Im Internet (und sonst auch…):
  ”Be strict in what you send and tolerant in what you receive.”
- Zweck: Fehlertoleranz, Interoperabilität
- Beispiel HTTP:
  - Zeilen lt. Spezifikation durch CRLF getrennt
  - akzeptiere aber auch wenn nur durch LF getrennt
    - Empfehlung auch in der Spezifikation

Kommunikationsbeziehungen

- one-to-one
- one-to-many
  - Gruppenkommunikation
  - multicast
- one-to-any
  - z.B. Lastverbund
- many-to-one
- many-to-many

Kommunikationsrichtung

- simplex: nur eine Richtung
- half-duplex: nur eine Richtung, kann sich aber ändern
- duplex (oder full-duplex): beide Richtungen gleichzeitig

Kommunikation & Verbindungen

verbindungsorientierte vs. verbindungslose Kommunikation

- verbindungsorientiert $\rightarrow$ Telephon
  1. Verbindung aufbauen
  2. Verhandlung über Verbindungsparameter $\rightarrow$ QoS
  3. Nachrichtenaustausch
  4. Verbindungsabbau
  - wenn unzuverlässiges Netzwerk
- verbindungslos $\rightarrow$ Brief
  - nur Datenübertragung
  - effizient, kein Overhead
  - wenn zuverlässiges Netzwerk

Signalisierung

- Austausch der Nachrichten, die zum Aufbau, der Überwachung und dem Abbau einer Verbindung notwendig sind
- in-band signalling
  - gleicher logischer Kanal wie Nutzdaten
  - Reihenfolge! $\rightarrow$ Problem der Priorität
- out-of-band signalling
  - getrennter logischer Kanal
- getrennter logischer Kanal $\rightarrow$ Multiplexen $\rightarrow$ asynchrones Zeitmultiplexverfahren (z.B. Port bei TCP)

Protokoll

- Austausch von Nachrichten muss gewissen Regeln entsprechen, damit die Kommunikationspartner einander verstehen ($\rightarrow$ Sprache)
- Protokolle sind präzise Festlegungen aller Regeln, die für eine Kommunikation notwendig sind.
  - Format der Nachrichten
    - Format der Daten
  - Abfolge (Reihenfolge) der Nachrichten
    - Zeitliche Spezifikationen
  - Spezifikation der Fehlersituationen

zustandsbehaftet vs. zustandslos

zustandsbehaftete vs zustandslose Protokolle

- zustandsbehaftet (stateful)
  - Nachrichten hängen vom Zustand der zuvor gesendeten Nachrichten ab
- zustandslos (stateless)
  - Nachrichten unabhängig von zuvor gesendeten Nachrichten
- Zusammenhang zu Server
  - stateful server
  - stateless server

Session

Sitzung

- feste Beziehung zwischen kommunizierenden Prozessen mit vereinbarten Eigenschaften (Namen, Ressourcen, Charakteristika,…)
- gemeinsamer Zustand zwischen kommunizierenden Prozessen
- meist Mechanismen zur Authentifikation und Autorisierung
- Zusammenhang zu Zustandsfähigkeit der Protokolle
  - mit zustandsbehafteten Protokoll: OK
  - mit zustandslosem Protokoll: Zustand muss übertragen werden (siehe HTTP)

Hierarchie von Protokollen

- Komplexität
- Abstraktionen
- $\rightarrow$ protocol suite
- $\rightarrow$ ISO/OSI, TCP/IP

Kommunikationsstile

Abstraktion!

- Shared Memory, gemeinsame Dateien, DB-basierte Kommunikation, Pipe, Queue
- Nachrichten-orientierte Kommunikation (messaging)
  - Versenden von Nachrichten
  - Abstraktion!
- Entfernte Funktionsaufrufe
  - Aufruf einer Funktion, die auf einem entfernten Host
- Entfernte Methodenaufrufe
  - Aufruf einer entfernten Methode (objektgebundene Funktion!), dessen Objekt auf einem entfernten Host
- stream-orientierte Kommunikation
  - Stream von Daten

Messaging – synchron vs. asynchron

synchron vs. asynchrone Nachrichtenübermittlung

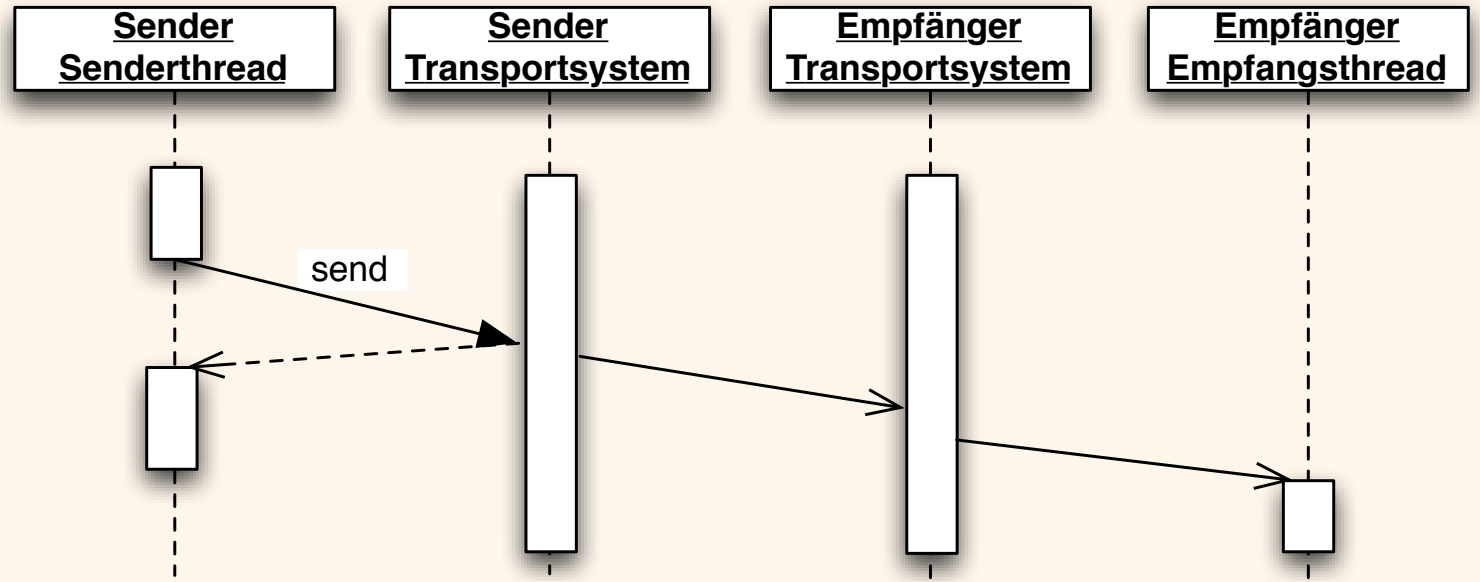
- synchron
  - Operation beginnt nur, wenn Sender die Nachricht initiiert hat und der Empfänger bereit ist die Nachricht zu empfangen
  - blockierende Aufrufe/Kommunikation
- asynchron
  - Sender initiiert die Nachricht unabhängig, ob der Empfänger bereit ist oder nicht
  - nichtblockierende Aufrufe/Kommunikation

Messaging – Semantik

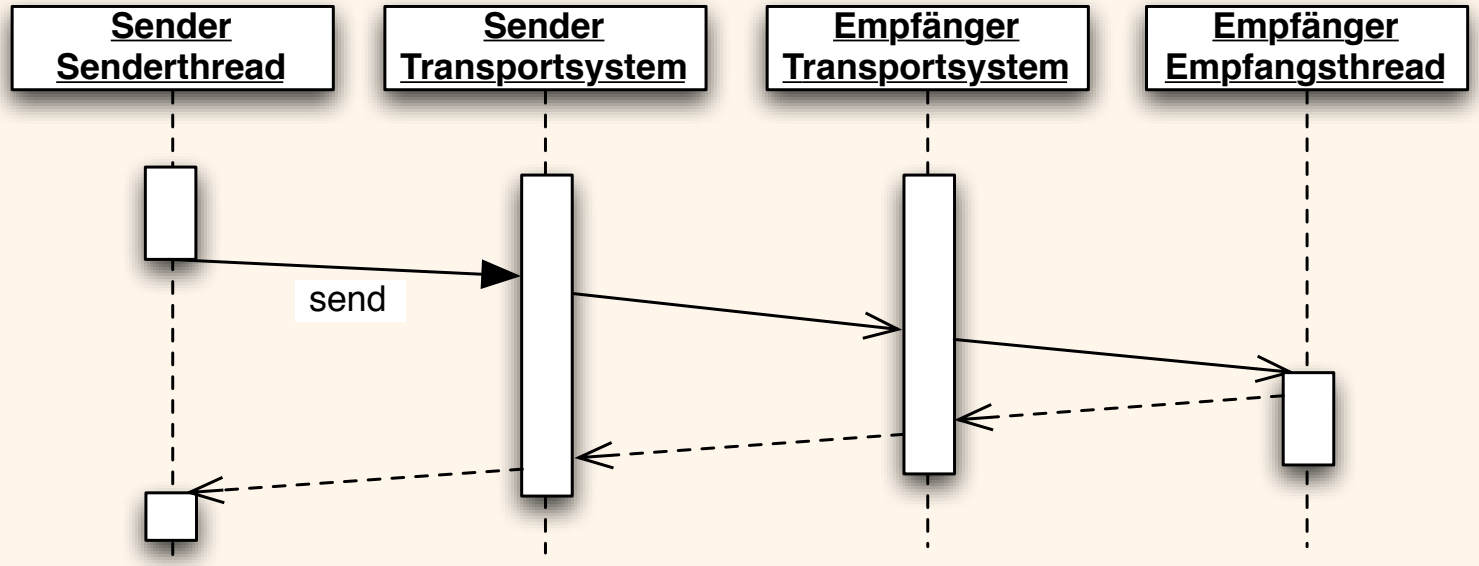
Semantik der Nachrichtenübermittlung

- no-wait send: Der Sendeprozess wartet lediglich bis die Nachricht im Transportsystem zum Absenden bereitgestellt ist
- synchronization send: Der Sendeprozess wartet bis die Nachricht vom Empfangsprozess entgegengenommen worden ist
- remote-invocation send: Der Sendeprozess wartet bis die Nachricht vom Empfangsprozess verarbeitet und beantwortet worden ist

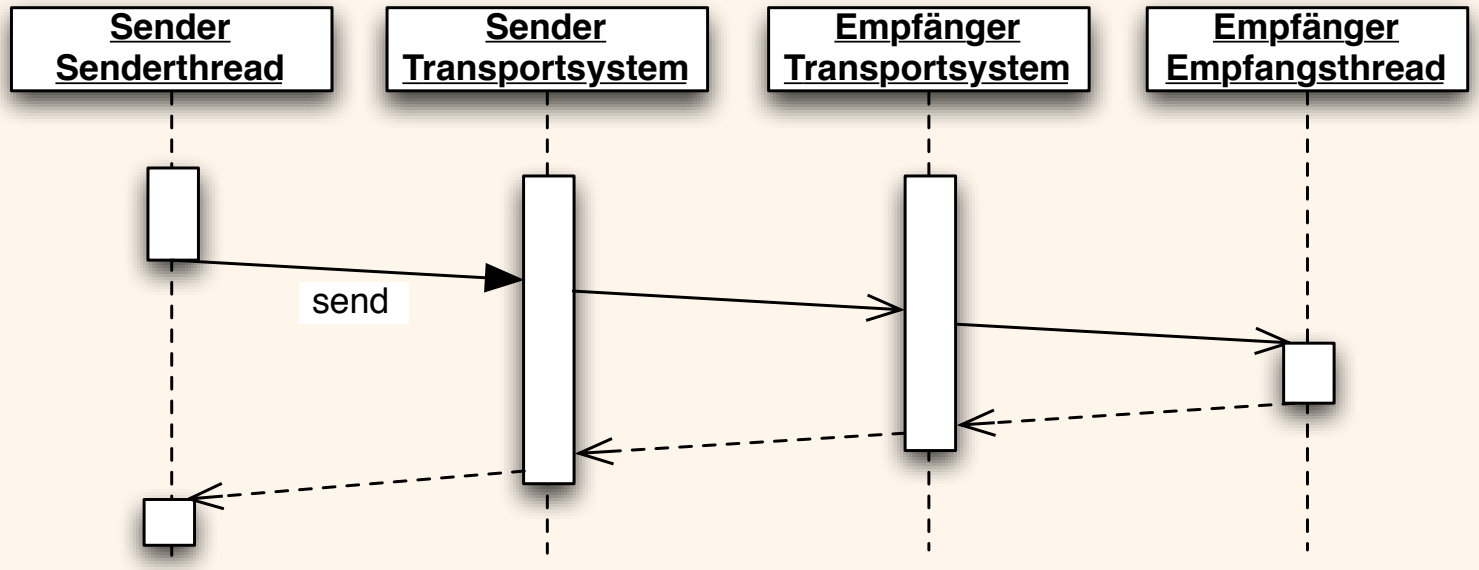
Semantik – no-wait send



Semantik – synchronization send



Semantik – remote-invocation send



Messaging – transient vs. persistent

transiente vs. persistente Kommunikation

- Transient
  - beide Kommunikationspartner online
  - Kommunikationssystem speichert Nachricht nur solange, wie die sendende und die empfangene Operation ausgeführt wird
- Persistent
  - Kommunikationssystem speichert Nachricht bis diese vollständig an den Empfänger ausgeliefert wurde
  - Motivation
    - Was ist, wenn der Server offline ist?
    - Was ist, wenn es keine Netzwerkroute zum Server gibt?

  $\rightarrow$ zeitliche Entkopplung!

Mess. – Kommunikationsmodelle

- Message Passing
  - direktes Senden der Nachricht an Empfänger
  - Senden und Empfangen ist gekoppelt
- Message Queueing
  - A stellt Nachricht in Queue, B liest von Queue
  - Senden und Empfangen ist zumindest zeitlich entkoppelt
  - meist point-to-point
    - da Client in Queue
    - Queue üblicherweise Inbox für *einen* Server
    - d.h. lediglich *schwach* entkoppelt

Mess. – Kommunikationsmodelle – 2

- Publish/Subscribe
  - one-to-many
  - A publiziert Nachricht zu einem Topic
  - B, C,… subskribieren Topic und erhalten Nachricht, wenn sie sich das nächste Mal verbinden
  - entkoppelt
    - zeitlich oft nur schwach, wenn nur Nachrichten nur an aktuelle Subskriber ($\rightarrow$ MOM)

Messaging – Patterns

- request/response, request/reply
- oneway
- batching
- publish/subscribe

Messaging – Serialisierung

Zwei Möglichkeiten wie Daten interoperabel übertragen werden können:

- Daten in ein maschinenunabhängiges Format transformieren
  - beide Kommunikationspartner müssen hin wandeln und wieder zurückwandeln wandeln
  - Overhead, wenn beide Partner, die gleiche Darstellung verwenden
    - die von maschinenunabhängigen Darstellung differiert
  - $\rightarrow$ single-canonical format
- Empfänger muss sich um eine evtl. notwendige Konvertierung kümmern
  - Sender spezifiziert den Datentyp aus einer Liste von vorgegebenen Datentypen
  - $\rightarrow$ receiver-makes-it-right
- $\rightarrow$ Folien ”Serialisierung”

Messaging – Protokolle

- properitäre Entwicklung basierend auf TCP, UDP
  - $\rightarrow$ Berkeley Socket API, `asio`, `java.net`,...
- ZeroMQ $\rightarrow$ Bibliothek zur schnellen, asynchronen Kommunikation zwischen Prozessen
  - Transport: in-process, inter-process, tcp
  - Patterns: request/reply, pair, publish/subscribe, pipeline
- Spread
  - $\rightarrow$ high performance, fehlertolerant, distributed system
  - open source
  - C++, Python, Java
- $\rightarrow$ REST
  - HTTP, HTTP/2

Message Oriented Middleware

Unter MOM versteht man

- eine Softwareinfrastruktur, die
- durch asynchrone Verbindungen charakterisiert ist und
- mehrere Systeme durch
- Nachrichten miteinander verbindet.

$\rightarrow$ Folien ”Systemarchitektur”

Protokolle für MOM

- AMQP (Advanced Message Queuing Protocol)
  - für Businessanwendungen
  - point-to-point, publish/subscribe
  - Implementierungen: Apache Apollo, Apache Qpid, RabbitMQ (Erlang),…
- STOMP (Simple Text Oriented Message Protocol)
  - Interoperabilität: zum Verbinden mit ”jedem” Broker
  - ActiveMQ, Apollo, RabbitMQ
- OpenWire
  - natives Protokoll von ActiveMQ, auch Apollo
- Java Message Service (JMS, Protokoll & API)
  - GlassFish (Oracle, open source), WildFly (Red Hat, free), IBM MQ, WebLogic (Oracle), RabbitMQ, Apollo

Protokolle für MOM – 2

- XMPP (eXtensible Messaging and Presence Protocol)
  - instant messaging
  - point-to-point, publish/subscribe
- Redis (open source)
  - network-based, inmemory DB mit publish/subscribe (eigenes Text-basiertes Protokoll)
  - C++, Java, C#, Python, JavaScript, PHP,…
- MQTT (MQ Telemetry Transport, ursprünglich IBM)
  - TCP, lightweight $\rightarrow$ IoT, M2M (machine-to-machine)
    - MQTT-SN: UDP, wenn hohe Anzahl an Paketverlusten oder bei geringen Ressourcen
  - publish/subscribe
  - Eclipse Mosquitto, HiveMQ, IBM MQ, RabbitMQ, Apollo
  - C++, Java, .Net, Python, JavaScript, PHP,…

Protokolle für MOM – MQTT

- hierarchische Topics (mit pattern matching)
  - z.B. `/etage1/wohnzimmer/temperatur/`
- QoS – delivered
  - at most once (”fire and forget”) (level 0)
  - at least once (level 1)
  - exactly once (level 2)
- Last Will And Testament
  - Verbindung zum Broker bricht ab, dann wird publiziert
- Retained Message (pro Topic)
  - persistente Nachricht, wird Client bei Verbindungsaufbau gesendet (wenn Topic subscribiert)
  - z.B. letzte Temperatur im Wohnzimmer
- Authentifizierung, TLS zur Verschlüsselung
  - auch über WebSockets

Entfernte Funktionsaufrufe – RPC

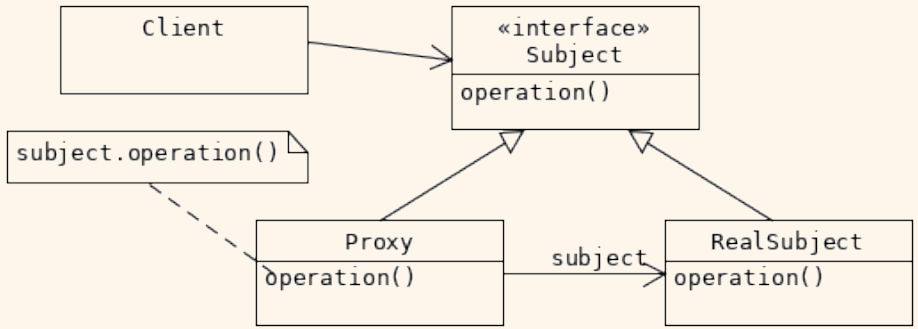
- Remote Procedure Call (RPC)
- Zugriffstransparenz bei Nachrichten-orientierter Kommunikation nicht gegeben, wenn lediglich eine Operation (Funktion) am Server ausgeführt werden soll.
- Proxy-Pattern
  - Client-Objekt
  - Client Stub (Proxy)
  - Server Stub (Skeleton)
  - Server-Objekt
- Interface Definition Language

Proxy-Pattern

Zweck Stellvertreter für ein anderes Objekt und kontrolliert Zugang zu dem Objekt

Prob/Kont. Zugang zu einem Objekt kann teuer sein oder Zugriff muss geregelt werden

Lösung



Verwendung *RemoteProxy* oder *ProtectionProxy* oder *VirtualProxy* (enthält Informationen von `RealSubject` falls Zugriff teuer)

Verweise Facade

RPC – Proxy-Pattern

- Client Stub
  - gleiches Interface wie Prozedur des Server-Objektes
  - Verbindung aufbauen (wenn notwendig)
  - Funktionsname und Parameter über Protokoll zu Server Stub
  - wartet bis Antwort
- Server Stub
  - nimmt Verbindung entgegen
  - empfängt Anforderung und ruft Funktion auf
  - schickt Antwort: Return-Wert oder Exception zurück

RPC – Probleme

- $\rightarrow$ Interoperabilitätsprobleme
- call-by-reference?
- call-by-value von Objekten zwischen verschiedenen Plattformen
  - .NET vs. JEE
  - C++ vs. C#,…
- Behandlung von Exceptions
- Transparenz kann nicht gewährleistet werden
  - lokaler Aufruf vs. entfernter Aufruf

First Law of Distributed Object Design: ”don’t distribute your objects” *Martin Fowler*

- Verbindung je Funktionsaufruf?
- Behandlung von Threads & Prozesse auf Serverseite

RPC – Fehlersemantik

- Client findet Server nicht
- Client stürzt ab, nachdem Nachricht versendet
- Nachricht geht verloren
- Server stürzt ab, nachdem er die Nachricht übernommen hat, aber
  - bevor er die Operation ausführen konnte
  - bevor er eine Antwort schicken konnte
- Antwort geht verloren
- Client stürzt ab, bevor Antwort erhalten

RPC – Fehlersemantik – 2

- Wie oft wurde eine Operation ausgeführt?
  - maybe: gar nicht, einmal, mehrmals
  - at least once: mindestens einmal (Wiederholungen, $\rightarrow$ idempotente Aufrufe!)
  - at most once: max. einmal (Seriennummern, persistent Verbindungen)
  - exactly once: genau einmal (Transaktionen)

RPC – Aufrufvarianten

- synchrone Funktionsaufrufe: remote-invocation send
- synchrone Prozeduraufrufe: synchronization send
- asynchrone Prozeduraufrufe: no-wait send
- asynchrone Funktionsaufrufe: no-wait send
  - aber Zugriff auf Rückgabewert mittels
    - polling: Objekt wird beim Aufruf mitgegeben oder als Rückgabewert zurückgeliefert $\rightarrow$ für Rückgabewert bzw. Exception
    - callback: Callback-Funktion wird beim Aufruf mitgegeben

Entfernte Methodenaufrufe – RMI

- Remote Method Invocation
  - wie RPC für entfernte Objekte
- aber weitere Anforderungen
  - Serialisierung von Objekten
    - Klasse am Server vorhanden
    - Klasse muss übertragen werden
  - Distributed Garbage Collection
  - Namensdienst um auf Serverobjekte zuzugreifen
  - Activation
    - nicht sinnvoll immer alle Objekte instanziert zu haben!
  - Versionsmanagement (der Programme, der Klasse $\rightarrow$ Objekte)
  - Nebenläufigkeit
    - Thread pro Anforderung
    - Thread pro Verbindung
    - Thread pro Objekt

RPC/RMI – Implementierungen

- ONC RPC (Open Network Computing)
- Apache Thrift
  - eigenes Protokoll
- Java RMI
- .NET Remoting ($\rightarrow$ WCF)
  - Funktionsumfang in etwa wie Java

RPC/RMI – Implementierungen – 2

- XML-RPC
  - einfach, für fast alle Programmiersprachen
- JSON-RPC
  - wie XML-RPC
  - Request

```
{ "jsonrpc": "2.0", "method": "echo",
  "params": ["hello, world"], "id": 1}
```

  - Response

```
{ "jsonrpc": "2.0", "result": "hello, world", "id": 1}
```

- SOAP (früher: Simple Object Access Protocol)
  - *nicht* objekt-basiert
  - XML
  - Teil von klassischen Webservices: WSDL, UDDI
    - .NET: WCF, Java: JAX-WS
- RESTful Webservices ($\rightarrow$ REST)

RPC/RMI – Implementierungen – 3

- Google gRPC
  - $\rightarrow$ Microservices, auch mobile Geräte und (in Zukunft) Browser
  - basierend auf protobuf und HTTP/2, Schema
    - aber (theoretisch) ”payload-agnostic” (z.B. JSON)
  - streaming! synchron und asynchron (kann abgebrochen werden), Timeouts, Metadaten, Authentifizierung, Flusskontrolle, load-balancing
  - C++, Java, Python, C#, node.js, PHP, Objective-C, Ruby, Go
  - authentication, bidirectional streaming and flow control, blocking or nonblocking bindings, and cancellation and timeouts

RPC/RMI – Implementierungen – 3

- CORBA
  - Common Object Request Broker Architecture
  - OMG (Object Management Group)
- ICE (Internet Communications Engine, open source)
  - publish/subscribe, load-balancing, failover, replication, IDL
  - TCP, TLS, UDP, WebSockets (auch über Firewalls)
  - C++, C#, Java, Python, JavaScript, PHP,…
  - Windows, Linux, OSX, Android, iOS

Stream-orientierte Kommunikation

- Stream von Daten
  - nicht abgeschlossene Informationseinheiten
  - Zeitverhalten wesentlich
- Beispiel Audio-Stream
  - unterliegendes Kommunikationssystem QoS
  - Bandbreite, Latenz, Jitter
  - aber u.U. keine Neuübertragung einzelner fehlerhafter/fehlender Pakete notwendig/sinnvoll!
- Synchronisation von Streams
  - Audio-Stream zu Video-Stream