

ZUSAMMENFASSUNG

Lernziele

Verteilte Software-Systeme (VSS) sind allgegenwärtig in beruflichem und privatem Alltag; sie führen Konzepte und Technologien aus unterschiedlichen Bereichen der Informatik zusammen. Nach dem Besuch dieses Moduls sind Sie in der Lage, VSS zu analysieren, zu entwerfen und mit Hilfe von Middleware und Frameworks zu realisieren:

- Sie können die charakteristischen Eigenschaften von VSS und die grundlegenden Fragestellungen beim Entwurf von VSS nennen.
- Sie können synchrone Remote Procedure Calls (RPC) und asynchrone Messaging-Kommunikationskanäle konzipieren und implementieren.
- Sie können die Konzepte und Funktionsweisen der wichtigsten Middleware-Dienste zum Bau von VSS erklären und beispielhafte Middleware-Implementierungen vergleichen und bewerten.
- Sie können operationale Infrastrukturen für verteilte Systeme unter Verwendung von Deployment Patterns entwerfen.

Unterlagen / Bücher (nicht Pflicht)

- Tanenbaum, M. van Steen, Distributed Systems, 2nd Edition, Pearson Education, 2007 (zweite Hälfte der Vorlesung; auch auf Deutsch verfügbar)
- J. Dunkel, A. Eberhart, S. Fischer, C. Kleiner, A. Koschel, Systemarchitekturen für verteilte Anwendungen, Hanser, 2008 (als E-Book in der Bibliothek vorhanden)
- P. Brown, Implementing SOA – Total Architecture in Practice, Addison Wesley 2008 (ausgewählte Kapitel, im letzten Teil der Vorlesung)
- M. Nygard, Release It!, <http://www.infoq.com/articles/nygard-release-it>
- MSDN Deployment Patterns, Operational (Topology) Modeling White Paper von IBM und weitere Assets zum Operational Modelling (z.B. IBM Hardware Sizing Tool, Palladio Developer Wiki)
- INCOSE Systems Engineering for Large Infrastructure Projects
- Online Ressourcen wie highscalability.com und Udacity Web Application Design Lecture (mit Reddit-Architekturrevolution)

Lerninhalte

Charakteristische Eigenschaften verteilter Software-Systeme:

- Vom Programm zum System: Verteilungsdimensionen, Kopplungsarten, Designherausforderungen
- VSS-Anwendungsgebiete in der Praxis, z.B. Unternehmensanwendungen, Distributed Control Systems, World-Wide Web (WWW)
- Wichtige Architekturstile und Anwendungstopologien: Client-Server, Hub-and-Spoke, Peer-to-Peer

Remoting (Netzwerkprogrammierung):

- Synchrone Kommunikation: TCP/IP Sockets, Remote Procedure Calls, HTTP Web Services
- Asynchrone Kommunikation mit Message-Oriented Middleware (MOM), Enterprise Integration Patterns
- weitere Message Exchange Patterns und Kommunikationsprotokolle im Überblick

Verteilte Software-Systeme

Zentrale Konzepte für den Entwurf verteilter Systeme:

- Naming
- Synchronization

Verteilte Algorithmen und Datenstrukturen:

- Namensauflösung
- Verteilte Hash-Tabellen
- Lamport-Uhr
- Vektor-Uhr

Anwendung verteilter Systeme. Zum Beispiel:

- BitTorrent
- Bitcoin
- Distributed batch processing

Operationale Modelle und Deployment Patterns für das qualitätsgtriebene Infrastrukturdesign:

- Performance und Skalierbarkeit
- Robustheit und Verfügbarkeit
- Systemmanagement und Auditierbarkeit
- Middleware-Produkte und Frameworks zum Bau von VSS
- Plattform-Auswahlkriterien und Designentscheidungen im VSS-Entwurf

Ausblick auf spezielle VSS – Event-Driven Architectures, Cloud Computing, Service-Oriented Architectures

Fallstudien aus der Industriepraxis

Einen Teil des Moduls wird in Englisch gehalten. Die Zusammenfassung ist in diesem Bereich aus Einfachheitsgründen ebenfalls auf Englisch.

EINFÜHRUNG	8
MOTIVATION.....	8
<i>Software Programm und Software System.....</i>	8
<i>Definition Distributed System (zu Deutsch: Verteiltes System)</i>	8
<i>Centralization and Transparency Types (ISO)</i>	9
DESIGN-HERAUSFORDERUNGEN UND LÖSUNGSANSÄTZE	9
<i>Vorteile verteilter Software-Systeme (VSS).....</i>	9
<i>Herausforderungen bei der Entwicklung verteilter Anwendungen</i>	9
<i>Middleware als VSS Enabler</i>	10
<i>ACID</i>	10
<i>Problemstellungen und Lösungsstrategien.....</i>	11
ARCHITEKTURSTILE ZUM BAU VON VSS.....	11
<i>Verschiedene Stile</i>	11
<i>Client-Server.....</i>	11
<i>Peer-to-Peer VSS Architecture Style.....</i>	13
WWW UND NETZWERKPROGRAMMIERUNG IN JAVA (ERSTE SCHRITTE)	13
<i>WWW ist meistens eine REST-basierte Architektur</i>	13
<i>REST Beispiel</i>	14
<i>Media Typen (MIME Types)</i>	14
ARCHITEKTONISCHE UND DESIGNTECHNISCHE ASPEKTE FÜR VERTEILTE SOFTWARE.....	14
IDEMPOTENT.....	15
BEISPIEL ZUR WICHTIG VON TIME-OUTS UND ERROR HANDLING	15
NETZWERKPROGRAMMIERUNG	16
ARCHITEKTURSTILE IM VERGLEICH.....	16
INTER-PROCESS KOMMUNIKATION VARIANTEN.....	16
TCP/IP SOCKETS	16
<i>Sockets im Überblick.....</i>	16
<i>Generic TCP Application</i>	17
<i>Socket-Programmierung: Grundkonzepte</i>	17
JAVA SOCKET API	17
<i>Beispiel</i>	17
<i>Berkeley Sockets und das Message Passing Interface (MPI)</i>	18
<i>IPv4 vs. IPv6</i>	18
<i>Vorteile und Nachteile von Java Sockets</i>	18
WEBSOCKETS	18
<i>Das Protokoll</i>	18
<i>Client API und Support</i>	18
<i>Server Support.....</i>	18
UDP (USER DATAGRAM PROTOCOL) → VERGLEICH MIT TCP	19
<i>Struktur eines UDP Programms</i>	19
<i>Beispiel - Sender und Empfänger</i>	19
<i>UDP vs. TCP.....</i>	20
<i>TCP/IP vs. UDP/IP</i>	20
MESSAGE EXCHANGE PATTERNS.....	21
ASYNCHRONOUS MESSAGING	22
EINFÜHRUNG IN MESSAGING	22
<i>Nachrichtenkonzept und die Komponenten.....</i>	22

<i>Message-Queuing Model und RPC's</i>	22
<i>Nachrichten nutzen um Applikationen zu verbinden und Daten zu verteilen</i>	22
<i>Message-Oriented-Middleware (MOM)</i>	23
<i>Hub-and-Spoke Architecture with CMB</i>	23
JAVA MESSAGE SERVICE (JMS) API	23
<i>Reliability levels</i>	23
<i>JMS Message-Struktur</i>	23
<i>Beispiel für einen JMS Message Provider</i>	24
<i>Andere APIs oder Provider</i>	24
MESSAGING MIDDLEWARE PROVIDER	24
<i>IBM MQ</i>	24
<i>MS MQ</i>	26
<i>Active MQ</i>	26
<i>MOM API Primitives (Platform-Independent Level)</i>	26
ENTERPRISE INTEGRATION PATTERNS.....	26
RABBITMQ (BASED ON AMQP).....	26
<i>Patterns</i>	26
RMI UND WEB SERVICES	28
REMOTE PROCEDURE CALLS (RPC)	28
<i>Events während einem RPC</i>	28
<i>Potentielle Fehler</i>	28
RMI (JAVA REMOTE METHOD INVOCATION)	29
<i>Zwei Programme – Client und Server</i>	29
<i>RMI Registry benutzen um eine Referenz eines Java-Objekts zu erhalten</i>	29
<i>Businesslogik nur auf dem Server</i>	29
<i>Architektur</i>	30
<i>Klassen und Interfaces</i>	30
<i>Innvolvierte Schritte</i>	30
<i>Schritte vor einem RMI Aufruf</i>	30
<i>Stub-Klasse in RMI (Remote Proxy)</i>	31
<i>Skeleton-Klasse in RMI</i>	31
<i>RMI-Programmierung auf der Serverseite</i>	31
<i>RMI-Programmierung auf dem Client</i>	31
<i>Schwächen von RMI</i>	31
INTERFACE DESCRIPTION LANGUAGES (IDLs)	32
<i>Einführung</i>	32
<i>DCE (Distributed Computing Environment) from the 1990s</i>	32
WEB SERVICES	32
<i>Einführung</i>	32
<i>Architektur</i>	32
<i>RPC vs. Document</i>	33
<i>Vor- und Nachteile von RPC und Document WSDL Styles</i>	33
<i>Schwächen von Web Services</i>	33
OPERATIONALE MODELLE, DESIGN FOR PERFORMANCE	34
NON-FUNCTIONAL REQUIREMENTS (NFRs)	34
<i>Constraints</i>	34
<i>Qualities</i>	34
<i>SEI Software Engineering Institute</i>	34
<i>Relevante Standards</i>	34

<i>QoS Summary</i>	35
<i>Beispiel für Design Tradeoff – Security vs. Performance</i>	35
PERFORMANCE NFR	36
<i>Was ist Performance</i>	36
<i>Performance Mythen</i>	36
<i>Die drei Hauptaspekte der Performance</i>	36
<i>End-zu-End Antwort Zeit</i>	36
COMPONENT AND OPERATIONAL MODELING	37
<i>Functional Model (Components) / Operational Model (Nodes)</i>	37
<i>UML Komponenten Modelle</i>	37
<i>Operational Model</i>	38
MSDN DEPLOYMENT PATTERNS	38
<i>Nicht-verteiltes Deployment</i>	38
<i>Verteiltes Deployment</i>	38
<i>Webfarm</i>	39
<i>Load-balancing cluster</i>	39
<i>Failover Cluster</i>	39
DESIGN FOR SCALABILITY AND AVAILABILITY	40
SCALABILITY NFR	40
<i>Scalability</i>	40
<i>Scalability Types</i>	40
<i>Horizontal Scaling (Scale Out)</i>	40
<i>Load Balancer</i>	40
<i>Edge Servers – IP Spraying</i>	41
<i>Session Affinity / Sticky Sessions</i>	41
AVAILABILITY NFR	41
<i>Availability – Gründe für Fehler</i>	41
<i>Key Availability Terms</i>	42
<i>Was heist 99.99x Availability?</i>	42
<i>Cost</i>	43
<i>Using components</i>	43
<i>Serial vs. Parallel Availability</i>	44
TECHNIQUES TO IMPROVE THE AVAILABILITY OF A (DISTRIBUTED) SYSTEM	44
<i>Avoid Single Points of Failure</i>	44
<i>Redundancy</i>	44
<i>Detect failures as fast as possible</i>	44
VSS-MANAGEMENT	45
WIESO UND WAS?	45
<i>IT Infrastructure Library (ITIL) → Service Management Framework</i>	45
SYSTEM MANAGEMENT PATTERNS	45
JAVA MANAGEMENT EXTENSIONS (JMX)	48
<i>Managed Beans (MBeans)</i>	48
<i>JConsole – JMX Client</i>	48
LOGGING	49
<i>Die Herausforderungen beim verteilten Logging</i>	49
<i>Logging Szenarios und Log Formate</i>	49
<i>Splunk</i>	49
<i>Tips und Tricks</i>	50

DER ZWECK VON CFIA UND WIE DIE RESULTATE GENUTZT WERDEN KÖNNEN.....	50
HAUPTZIELE.....	51
TYPISCHES RESULTAT.....	51
HIGH-LEVEL STEPS.....	52
BEISPIEL.....	52
BASICS CFIA MATRIX	53
<i>Example of Configurations Items (CIs)</i>	54
<i>How to read the CFIA matrix.</i>	54
NODE ANALYSIS.....	55
RISK LOG.....	56
RESILIENCE SCALE TABLE	56
RECOVERY SCALE TABLE	57
CHEAT SHEET – «MOST POPULAR» PROBLEMS.....	57
NAMING	58
NAME RESOLUTION.....	58
<i>Terminology</i>	58
<i>From a Name to an Entity</i>	59
<i>Other Examples</i>	59
<i>More Terminology</i>	59
NAMING OVERVIEW.....	60
<i>Flat Naming</i>	60
<i>Structured Naming</i>	61
<i>Attributed-based Naming</i>	61
DISTRIBUTED HASH TABLES	63
MOTIVATING EXAMPLE – P2P CLOUD STORAGE	63
<i>A First Attempt</i>	63
<i>Solution – Consistent Hashing</i>	63
<i>Chord</i>	63
<i>Lookup</i>	65
<i>Chord – Correctness</i>	66
<i>Chord – Stabilization</i>	66
<i>Leaving (unplanned)</i>	67
<i>Chord – Replication</i>	67
SYNCHRONIZATION	68
TWO TYPES OF CLOCKS	68
<i>Physical clocks (Basis: Astronomy, Mechanics)</i>	68
<i>Logical clocks (Basis: Causality)</i>	69
BITCOIN / DIGITAL MONEY	74
GOALS	74
DIGITAL SIGNATURES.....	74
A FIRST ATTEMPT AT DESIGNING A BITCOIN-LIKE SYSTEM.....	74
<i>Setting</i>	74
<i>Basic Idea</i>	74
<i>Sending Bitcoins</i>	74
<i>Double Spending</i>	75
<i>Consensus Protocol</i>	75

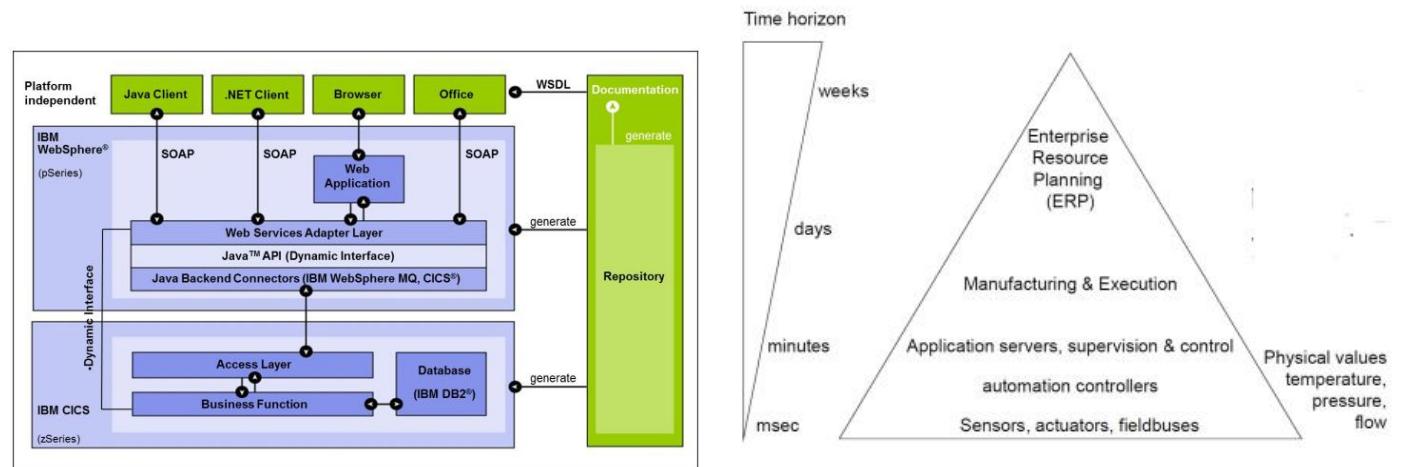
RANDOM HASH FUNCTIONS	76
BITCOIN'S CONSENSUS PROTOCOL.....	76
<i>Step 1 - How does the protocol work?</i>	76
<i>Step 2 - What happens if people cheat?</i>	78

Einführung

Motivation

Verteilte Software-Systeme (VSS) sind allgegenwärtig in beruflichen und privaten Alltag. Sie führen Konzepte und Technologien aus unterschiedlichen Bereichen der Information zusammen. Dazu zählen die Netzwerkprogrammierung (Remoteing), Betriebssysteme und Middleware, User Interfaces, Datenbanken und Algorithmen.

Beispiele dafür sind Unternehmensanwendungen im Core Banking oder auch Prozessautomation und Überwachung aus der Industrie.



Software Programm und Software System

Programm

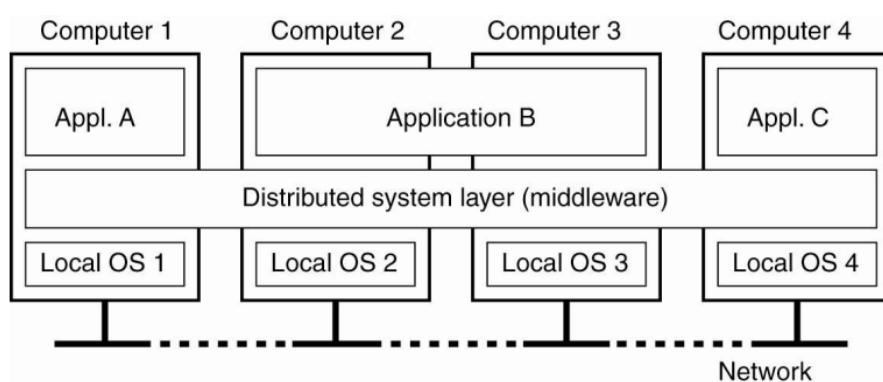
- Hat ein Anfang und ein Ende
- Wird meist anhand von Parametern ausgeführt.
- Kann offline geupdated werden (Stop, Neustart)
- Ist Zeitlich gesteuert, hat einen Callstack

System (von Programmen/Systemen)

- Hört vielleicht gar nie auf.
- Akzeptiert den Input zu jeder Zeit (Asynchron über externe Schnittstellen)
- Meistens wird es dynamisch während der Laufzeit rekonfiguriert.
- Ist Event-gesteuert.

Definition Distributed System (zu Deutsch: Verteiltes System)

Ein verteiltes System ist eine Sammlung von unabhängigen Computer die gegenüber dem Benutzer als ein ganzes System auftreten.



Konzept	Beispiel
Centralized services	A single server for all users
Centralized data	A single on-line telephone book
Centralized algorithms	Doing routing based on complete information

Transparency	Beschreibung
Access	Hide differences in data representation and how a resource is accessed.
Location	Hide where a resource is located
Migration	Hide that a resource may move to another location
Relocation	Hide that a resource may be moved to another location while in use
Replication	Hide that a resource is replicated
Concurrency	Hide that a resource may be shared by several competitive users
Failure	Hide the failure and recovery of a resource.

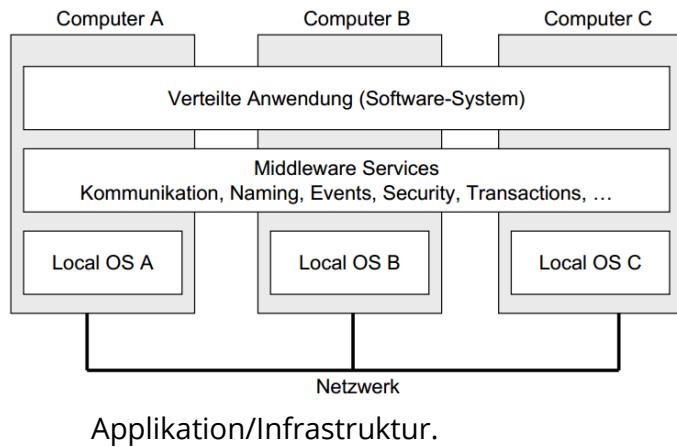
Design-Herausforderungen und Lösungsansätze

Vorteile verteilter Software-Systeme (VSS)

VSS bilden die verteilte Realität ab. Die Benutzer arbeiten an verschiedenen Arbeitsplätzen. Die Unternehmensstandorte und Abteilungen sind in der Regel verteilt. VSS führen zu einer Performance Steigerung, wenn mehrere Rechner an einer Aufgabe parallel arbeiten können. Eine Skalierbarkeit ist dadurch gegeben, dass Leistungsengpässe durch zusätzliche Hardware kompensiert werden können. Zudem ist ein System so fehlertoleranter, da Software auf mehreren Rechnern repliziert werden.

Herausforderungen bei der Entwicklung verteilter Anwendungen

- Komplexe Kommunikation / Systeme
 - o Interprozess-Kommunikation statt lokaler Methodenaufrufe
 - o Heterogene Systeme (Rechner, Betriebssysteme, Hardware, Sprachen)
- Performanz Problem
 - o Kommunikation über langsame Netzwerke (Latenz, Durchsatz)
- Zuverlässigkeit → CFIA – Component Failure Impact Analysis
 - o Alle Netzzugriffe sind potentiell unsicher
 - o Maschinenausfälle
- Transaktionssicherheit
 - o Mehrere Benutzer greifen gleichzeitig auf Daten zu (zur Sicherstellung der Konsistenz ist Isolation von Transaktionen erforderlich).



Eine Middleware ist eine infrastrukturelle Software zur Kommunikation zwischen Software-Komponenten und Anwendungen auf verschiedenen Computern.

- Verteilungsplattform mit entsprechenden Protokollen
- Höheres Abstraktionsniveau als einfacher Datenaustausch
- Verbirgt Komplexität der zugrundeliegenden Applikation/Infrastruktur.

Gründe für die Einführung einer Middleware sind die Überwindung von der Heterogenität oder die Vereinfachung der Erstellung verteilter Anwendungen.

Kommunikationsorientierte Middleware

Der Schwerpunkt liegt in der Abstraktion (Vereinfachung) von der Netzwerkprogrammierung (TCP/IP, Socket APIs). Beispiele sind Java Remote Method Invocation (RMI) oder Webservices wie SOAP.

Anwendungsorientierte Middleware

Der Schwerpunkt liegt in der weiterreichenden Unterstützung verteilter Anwendungen. Ein komplexer Aufbau führt zu zahlreichen Zusatzdiensten wie Discovery, Sicherheit, Zuverlässigkeit, verteilte Transaktionen und Session. Beispiele dafür sind CORBA, J2EE oder verteilte Betriebssysteme.

ACID

Atomicity

Von einer atomaren Operation spricht man, wenn eine Sequenz von Daten-Operationen entweder ganz oder gar nicht ausgeführt wird (Alles-oder-nichts-Eigenschaft)

Consistency

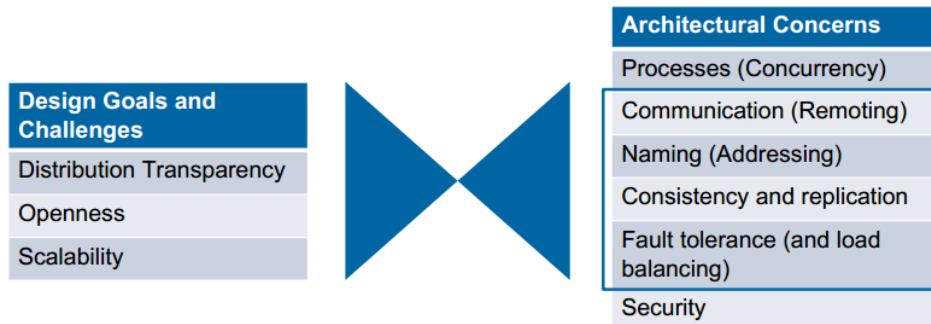
Konsistenz heißt, dass eine Sequenz von Daten-Operationen nach Beendigung einen konsistenten Datenzustand hinterlässt, falls die Datenbank davor auch konsistent war.

Isolation (Abgrenzung)

Durch das Prinzip der Isolation wird verhindert/eingeschränkt, dass sich gleichzeitig in Ausführung befindliche Daten-Operationen gegenseitig beeinflussen.

Durability

Der Begriff Dauerhaftigkeit sagt aus, dass Daten nach dem erfolgreichen Abschluss einer Transaktion garantiiert dauerhaft in der Datenbank gespeichert sind. Die dauerhafte Speicherung der Daten muss auch nach einem Systemfehler (Software-Fehler oder Hardware-Ausfall) garantiiert sein. Insbesondere darf es nach einem Ausfall des Hauptspeichers nicht zu Datenverlusten kommen. Dauerhaftigkeit kann durch das Schreiben eines Transaktionslogs sichergestellt werden. Ein Transaktionslog erlaubt es, nach einem Systemausfall alle in der Datenbank fehlenden Schreib-Operationen zu reproduzieren.

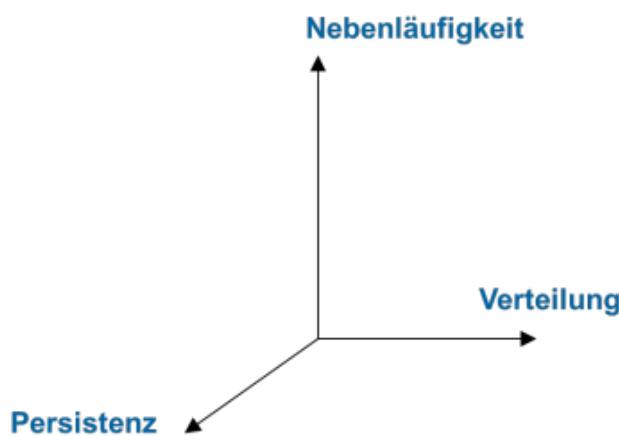


Architekturstile zum Bau von VSS

Verschiedene Stile

- Client-Server
- Distributed objects
- Hub-and-spoke
- Event-Driven Architecture
- Peer-to-Peer (P2P)

Die unterschiedlichen Architektur-Stile werden durch Integrations-Optionen beeinflusst.



Transparenz betreffend Nebenläufigkeit

Daten oder andere Ressourcen möglicherweise gleichzeitig von konkurrierenden Benutzern bzw. Applikationen verwendet werden.

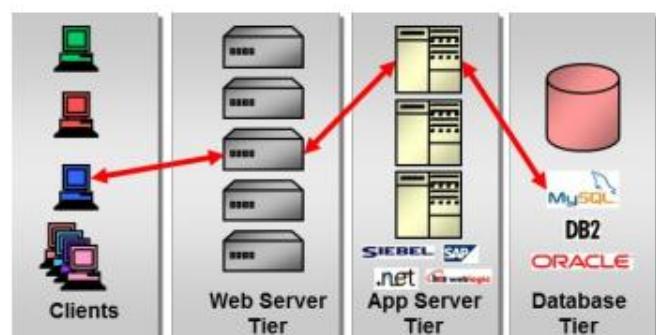
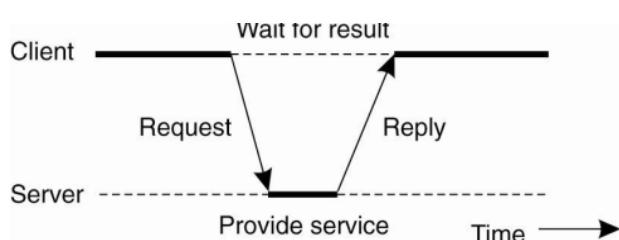
Transparenz betreffend Persistenz

verbirgt, ob Ressource im lokalen Speicher (Memory) oder auf einer Festplatte ist.

Transparenz betreffend Verteilung

verbirgt, ob die aufgerufene Komponente lokal oder entfernt ist.

Client-Server



Traditional 3-tier architectures are designed to scale linearly

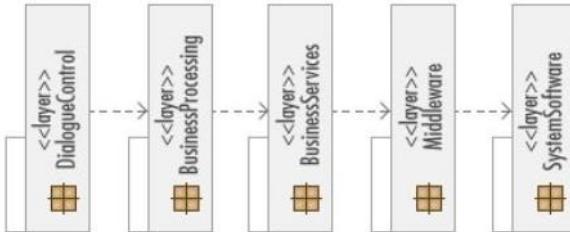
Tier 1 User PC running browsers (front-end)

Tier 2 Web server and presentation logic, Application server und business logic, integration

Tier 3 Database and other backend systems

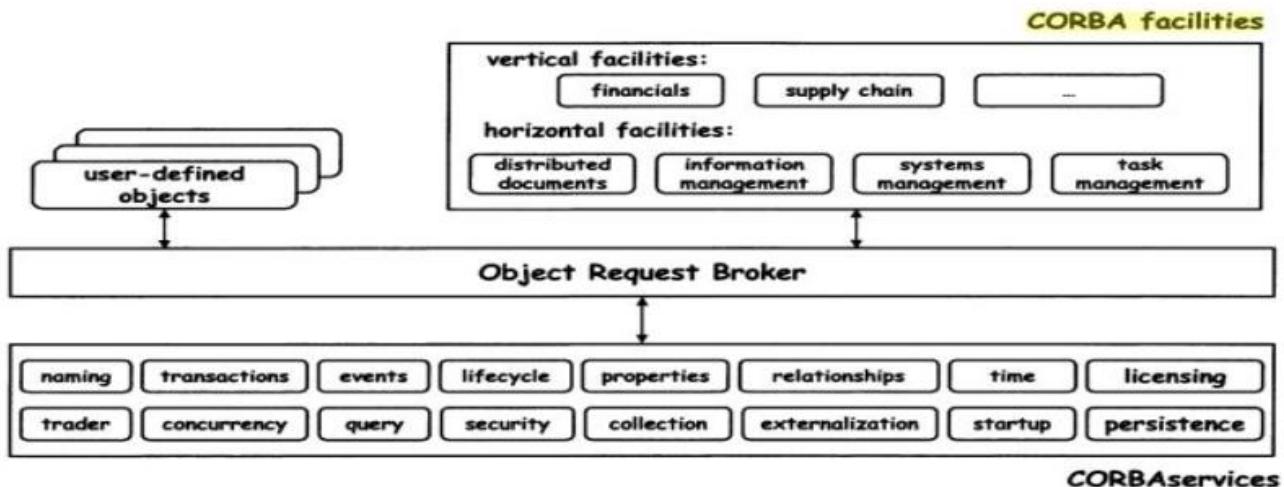
Jedes Tier hat seine eigenen Layer Architektur. Es kann gleiche Teile geben, muss es aber nicht.

Beispiel einer geschichteten Architektur

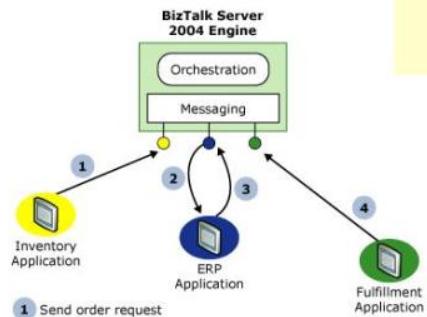


DialogueControl	The dialogue control layer handles user-system interactions and use case logic.
BusinessProcessing	The business processing layer contains application-specific services that handle use case step logic and choreography
BusinessServices	The business services layer contains more general business components that may be used in several applications.
Middleware	The middleware layer contains components such as interfaces to databases and platform-independent operating system services.
SystemSoftware	The system software layer contains components such as operating systems and databases.

CORBA (Common Object Request Broker Architecture)

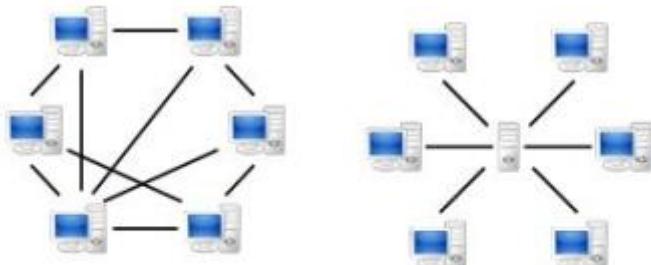


Service	Description
Object life cycle	Defines how CORBA objects are created, removed, moved, and copied
Naming	Defines how CORBA objects can have friendly symbolic names
Events	Decouples the communication between distributed objects
Relationships	Provides arbitrary typed n-ary relationships between CORBA objects
Externalization	Coordinates the transformation of CORBA objects to and from external media
Transactions	Coordinates atomic access to CORBA objects (→ ACID, Slide 20)
Concurrency Control	Provides a locking service for CORBA objects in order to ensure serializable access
Property	Supports the association of name-value pairs with CORBA objects
Trader	Supports the finding of CORBA objects based on properties describing the service offered by the object
Query	Supports queries on objects

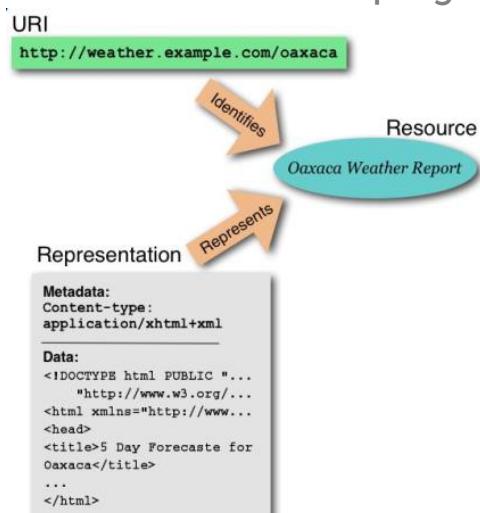


Peer-to-Peer VSS Architecture Style

P2P Systeme implementieren eine verteilte Applikationsarchitektur, welche den Workload in Partitionen zwischen den Peers aufteilt. Die Peers sind alle gleichberechtigt. Ein P2P System verbindet dynamisch alle Nodes um Traffic zu routen und somit Tasks auszulagern. Es bietet den Vorteil, dass man skalieren kann ohne die zentralen Ressourcen auszubauen. Es werden die Prozessor- und Netzwerkpower der Client genutzt. Meist sind dort mehr als genug Leistung vorhanden. Dabei gibt es keinen Single Point of Failure. Beispiele sind Skype oder BitTorrent.



WWW und Netzwerkprogrammierung in Java (erste Schritte)



Das weltweit grösste verteilte Software Systeme, ist das WWW. Es entspricht in etwa dem Client-server architectural style. Ein Naming findet mit Uniform Resource Identifiers (URIs) statt. Die Kommunikation geschieht mit http über TCP/IP. Im Internet gibt es eigentlich keinen Point of Failure, da unglaublich viel Redundanz vorhanden ist.

WWW ist meistens eine REST-basierte Architektur
REST (Representational State Transfer = Architekturstil für verteilte Applikationen. REST unterstützt alle vier CRUD Operationen. Es basiert auf einem «stateless» Client-Server Kommunikationsprotokoll. Logisch gesehen ist http simple und einfach dazu da, Aufrufe zwischen zwei Maschinen

auszutauschen. Es ist weniger kompliziert als CORBA, RPC oder SOAP.

```
<?xml version="1.0"?>
<soap:Envelope
  xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
  soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
  <soap:body pb="http://www.acme.com/phonebook">
    <pb: GetUserDetails>
      <pb: UserID>12345</pb: UserID>
    </pb: GetUserDetails>
  </soap:Body>
</soap:Envelope>
```

Das Resultat könnte eine XML-Datei sein, welche als Payload in der SOAP-Antwort eingebettet ist.

Nutzung von Rest

```
http://www.acme.com/phonebook/UserDetails/12345
```

Der URL wird mit einem GET Request zum Server gesendet. Als Antwort werden die reinen Daten zurückgeben ohne jeglichen Müll rund herum.

Media Typen (MIME Types)

Es gibt bereits hunderte von definierten Media Typen. Beispiele sind text/plain oder image/png. Einen bestehenden Typen zu wählen oder einen neuen Typen zu entwerfen ist eine wichtige Designentscheidung bei der Erstellung von http API's. (Client und Server müssen es verstehen).

Netzwerkklassen im JDK

```
import java.net.*;
import java.io.*;

public class URLConnectionReader {
  public static void main(String[] args) throws Exception {
    URL oracle = new URL("http://www.oracle.com/");
    URLConnection yc = oracle.openConnection();
    BufferedReader in = new BufferedReader(new InputStreamReader(
      yc.getInputStream()));
    String inputLine;
    while ((inputLine = in.readLine()) != null)
      System.out.println(inputLine);
    in.close();
  }
}
```

Architektonische und Designtechnische Aspekte für verteilte Software

- **Kommunikations Topologie**
 - o Ein Client, ein Server? Mehrere Clients für einen Server? Dynamisches Setup?
- **Location Autonomy (Transparenz)**
 - o Wirkliche Adressen vs. Virtuelle Adressen (Was wenn ein Server verschoben wird?)
- **Invocation and Message Semantics**
 - o Bytestream vs. Document vs. Procedure vs. Remote Object
- **Timeout Management**
 - o 1ms? 30 Sekunden? Unendlich?
- **Error Handling**
 - o Erneute Requests
 - o Server idempotent machen?

Idempotent

Definition gemäss Internet: "*Idempotence is a funky word that often hooks people. Idempotence is sometimes a confusing concept, at least from the academic definition. From a RESTful service standpoint, for an operation (or service call) to be idempotent, clients can make that same call repeatedly while producing the same result. In other words, making multiple identical requests has the same effect as making a single request. Note that while idempotent operations produce the same result on the server (no side effects), the response itself may not be the same (e.g. a resource's state may change between requests). The PUT and DELETE methods are defined to be idempotent. However, there is a caveat on DELETE. The problem with DELETE, which if successful would normally return a 200 (OK) or 204 (No Content), will often return a 404 (Not Found) on subsequent calls, unless the service is configured to "mark" resources for deletion without actually deleting them. However, when the service actually deletes the resource, the next call will not find the resource to delete it and return a 404. However, the state on the server is the same after each DELETE call, but the response is different. GET, HEAD, OPTIONS and TRACE methods are defined as safe, meaning they are only intended for retrieving data. This makes them idempotent as well since multiple, identical requests will behave the same.*"

A server is idempotent if all the operations it supports are idempotent. It is useful if a server declares that it is idempotent (e.g., its operations are all queries). The RPC runtime system learns that fact when it creates a binding to the server. In this case, if the client RPC runtime sends a call but does not receive a reply, it can try to call again and hope that the second call gets through. If the server is not idempotent, however, it's not safe to retry the call. In this case, the client could send a control message that says "Are you there?" or "Have you processed my previous message?" but it can't actually send the call a second time, since it might end up executing the call twice.

Even if it resends calls (to an idempotent server) or it sends many "Are you there?" messages (to a non-idempotent server), the caller might never receive a reply. Eventually, the RPC runtime will give up waiting and return an exception to the caller. The caller cannot tell whether the call executed or not. It just knows that it didn't receive a reply from the server. It's possible that a server will reply later, after the RPC runtime returns an exception. At this point, it's too late to do anything useful with the reply message, so the RPC runtime simply discards it.

Looking at the issue a bit more abstractly, the goal is to execute an idempotent operation *at least once* and to execute a non-idempotent operation *at most once*. Often, the goal is to execute the operation *exactly once*. Transactions can help. A call executes exactly once if the server is declared non-idempotent and the RPC executes within a transaction that ultimately commits. We will explore exactly-once beha

Beispiel zur Wichtig von Time-Outs und Error Handling

Eine Exception, welche eine Airline gegrounded hat.

Eine JDBC Exception wurde auf einem Applikationsserver nicht gefangen (not caught). Dies führte dazu, dass alle Self Check-In Kioske der Airline offline gingen. Die Kioske waren dann offline für mehrere Stunden. Zur Normalität kam es erst als alle Applikations- und Datenbankserver neugestartet wurden.

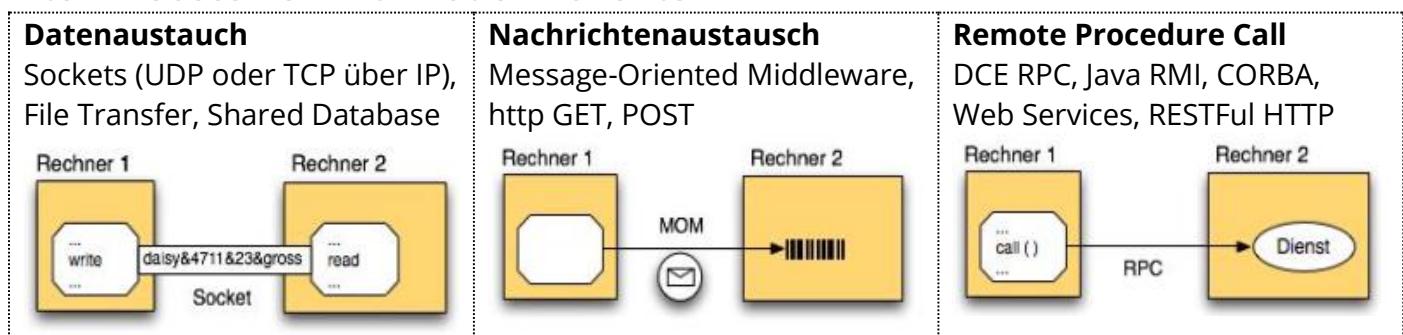
Hauptgründe, wieso es dazu kam

- Unangemessenes Wiederholungs-/Failover-Management auf den Applikationsservern
- Kein Timeout Management auf den Applikationsservern
- Unvollständiges Monitoring der Applikationsserver (Sie wurden immer noch als Grün angezeigt, obwohl Sie geblockt waren).

Architekturstile im Vergleich

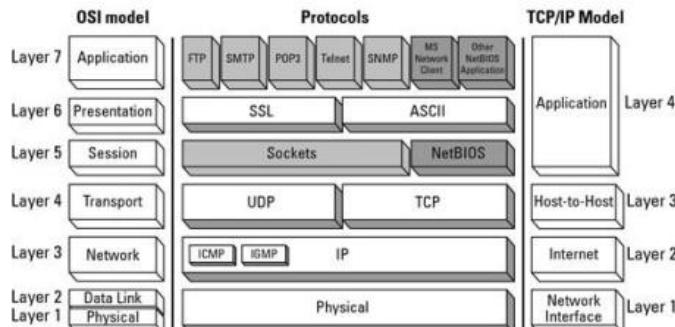
Stil	Topologie-Muster	Verbreitung	Known Uses
Client-Server	n:1 (2-Tier, 3-Tier, n-Tier)	Mainstream	WWW (HTTP) Web Services Microservices DCE RPC
Distributed Objects	n:m	Nach Boom jetzt gering	CORBA, RMI
Hub-and-Spoke with Messaging	Stern, Bus (also n:1:m)	Mainstream	ESB
Event-Driven Architecture (EDA)	Wie Hub-and-Spoke	Forschungsthema Prototypen	IoT
Shared Data	n:1	Forschungsthema Prototypen	Data Spaces
Peer-to-Peer (P2P)	n:m, oft mit Client-Server als unterliegender Kommunikationsinfrastruktur	Hoch in Nischen-Einsatzgebieten	File Sharing Music Sharing

Inter-Process Kommunikation Varianten



TCP/IP Sockets

Verbindung zwischen dem OSI und dem TCP/IP Model



- Network interface (layer 1):** Deals with all physical components of network connectivity between the network and the IP protocol
- Internet (layer 2):** Contains all functionality that manages the movement of data between two network devices over a routed network
- Host-to-host (layer 3):** Manages the flow of traffic between two hosts or devices, ensuring that data arrives at the application on the host for which it is targeted
- Application (layer 4):** Acts as final endpoints at either end of a communication session between two network hosts

Sockets im Überblick

Sockets sind ein Basis-Mechanismus für alle komplexeren Verfahren wie http oder RMI. Es dient zum Austausch von Bytestörmern auf der Programmierebene. Das Sockets ist eine Verbindung zwischen zwei Kommunikations-Endpunkten (IP + Port). Beim Konzept gibt es die zwei Rollen: Client und Server.

Nachteile von Sockets

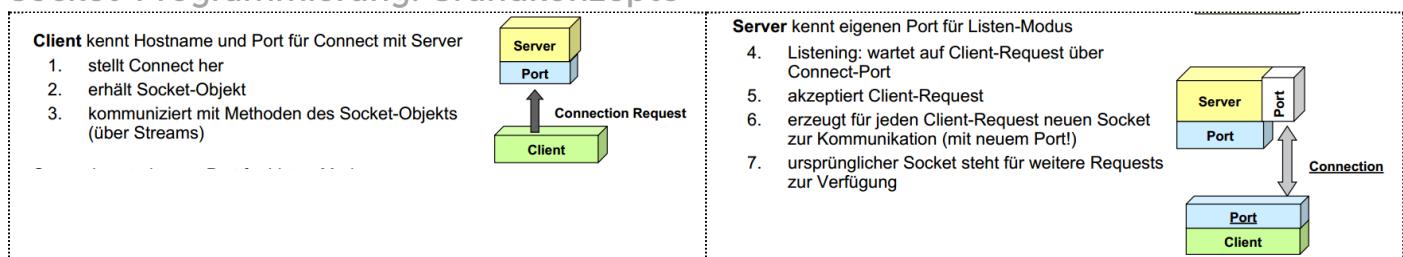
Es wird ein Konstruieren und Parsen der Byteströme erforderlich (keine Objekte und daher auch keine Typsicherheit). Zudem muss das Message Exchange Pattern (MEP) selbst spezifiziert, implementiert und überprüft werden. Das MEP definiert die Reihenfolge der send/receive Calls durch Client und Server.

Algorithmus für den TCP Client

Finden der IP und der Portnummer des Servers, Erstellen eines TCP Sockets, Verbinden des Sockets mit dem Server (Server muss laufen und auf Request hören), Senden/Empfangen von Daten durch das Socket, Verbindung schliessen.

Algorithmus für den TCP Server

Ein TCP Server Socket erstellen, Binden des Server Sockets auf die Server IP und Port, Akzeptieren von neuen Verbindungen von den Clients, Senden / Empfangen von Daten über das Server Socket, Schliessen der Verbindung mit dem Client.

Socket-Programmierung: Grundkonzepte**Java Socket API****Beispiel****Socket Server**

Ein Socket (engl. Steckdose) ist ein Verbindungsendpoint, der vom Programm wie eine gewöhnliche Datei beschrieben und gelesen werden kann. Der Serverprozess muss von außen eindeutig angesprochen werden können. Dazu bindet er sich an einen festen Port, den sogenannten "well known port", über den er erreichbar ist

```
*import java.net.ServerSocket;*
public class TimeServer {
    public static void main(String args[]) throws Exception {
        int port = 2342;
        // erzeuge neuen Server-Socket für speziellen Port
        ServerSocket server = new ServerSocket(port);
        while (true) {
            System.out.println("server > Waiting for client...");
            // Server akzeptiert Anfragen von Client und erzeugt dazu Client-Socket
            // Vorsicht: die Methode blockiert bis der Client eine Connection aufmacht
            Socket client = server.accept();
            System.out.println("server > Client from " + client.getInetAddress() + " connected.");
            // Stream zum Schreiben an den Client
            PrintWriter out = new PrintWriter(client.getOutputStream(), true);
            Date date = new Date();
            out.println(date);
        }
    }
}
```

Socket Client

Der Client braucht keinen festen Port. Er holt sich einen normalen Socket, dem vom System eine freie Nummer zugeteilt wird. Der Server erfährt die Nummer des Clients aus der Anfrage und kann ihm unter diesem Port Antworten. Im nächsten Schritt ruft der Client connect() auf, um eine Verbindung mit dem Server aufzunehmen, der in den Parametern beschrieben wird. Sobald die Verbindung da ist, sendet der Client seine Anfrage per write() oder alternativ send() und wartet per read() oder recv() auf die Antwort des Servers.

```
*import java.net.Socket;*
public class TimeClient {
    public static void main(String args[]) throws IOException {
        String host = "localhost";
        int port = 2342;
        // Socket zur Verbindung zum Server
        Socket server = new Socket(host, port);
        System.out.println("client > Connected to " + server.getInetAddress());
        // Stream zum Lesen von Server
        BufferedReader in =
            new BufferedReader(new InputStreamReader(server.getInputStream()));
        String date = in.readLine();
        System.out.println("client > Server said: " + date);
    }
}
```

Berkeley Sockets und das Message Passing Interface (MPI)

Gleiches Protokoll auf allen Platformen und in allen Sprachen. Der API Komfort ist je nach Sprache und Bibliothek unterschiedlich. Dabei müssen die System Resourcen gemanaged werden.

Primitive	Meaning
Socket	Create a new communication end point
Bind	Attach a local address to a socket
Listen	Announce willingness to accept connections
Accept	Block caller until a connection request arrives
Connect	Actively attempt to establish a connection
Send	Send some data over the connection
Receive	Receive some data over the connection
Close	Release the connection

Primitive	Meaning
MPI_bsend	Append outgoing message to a local send buffer
MPI_send	Send a message and wait until copied to local or remote buffer
MPI_ssend	Send a message and wait until receipt starts
MPI_sendrecv	Send a message and wait for reply
MPI_isend	Pass reference to outgoing message, and continue
MPI_issend	Pass reference to outgoing message, and wait until receipt starts
MPI_recv	Receive a message; block if there is none
MPI_irecv	Check if there is an incoming message, but do not block

IPv4 vs. IPv6

Es gibt einige Änderungen für die Socket-Programme (auch Java). Das Socket API und das primitive Interface bleiben das Gleiche. Die Adressierung hat geändert und daher auch die Formate der Antworten. Zudem unterscheidet sich das Quality-of-Service (Performance, Timeout Management).

Vorteile und Nachteile von Java Sockets

Vorteile

Flexibel, Mächtig, Generiert nur wenig Netzwerk Traffic (wenn es effizient genutzt wird).

Nachteile

Können nur Raw Daten senden, Client und Server müssen einen Mechanismus haben um die Daten zu interpretieren, Zudem müssen beide die State Informationen halten.

WebSockets

Das Protokoll

Eine auf TCP basierendes Netzwerkprotokoll, das entworfen wurde, um eine bidirektionale Verbindung zwischen einer Webanwendung und einem WebSocket-Server bzw. einem Webserver herzustellen. Während bei einer reinen HTTP-Verbindung jede Aktion des Servers eine vorhergehende Anfrage des Clients erfordert, reicht es beim WebSocket-Protokoll, wenn der Client die Verbindung öffnet. Der Server kann dann diese offene Verbindung aktiv verwenden und kann neue Informationen an den Client ausliefern, ohne auf eine neue Verbindung des Clients zu warten. Bei WebSockets entfallen die durch den HTTP-Header verursachten zusätzlichen Daten, die bei jeder Anfrage einige Hundert Bytes umfassen können

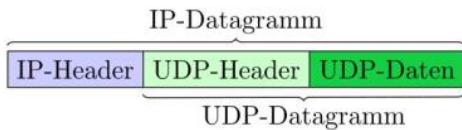
Client API und Support

Web Sockets ist eine state-of-the-art Kommunikationstechnologie für Webapplikationen. Es arbeitet über Sockets. Ist sichtbar für ein JavaScript Interface oder kompatibler HTML5 Browser. Sobald man eine WebSocket Verbindung hat, ist es möglich Daten mit send() an den Server zu senden.

Server Support

- C++: libwebsockets
- Erlang: Shirasu.ws
- Java: Jetty (and others)
- Node.JS: ws
- Ruby: em-websocket
- Python: Tornado, pywebsocket
- PHP: Ratchet, phpws

Eine Demo zu Websockets gibt es unter <http://www.websocket.org/echo.html>

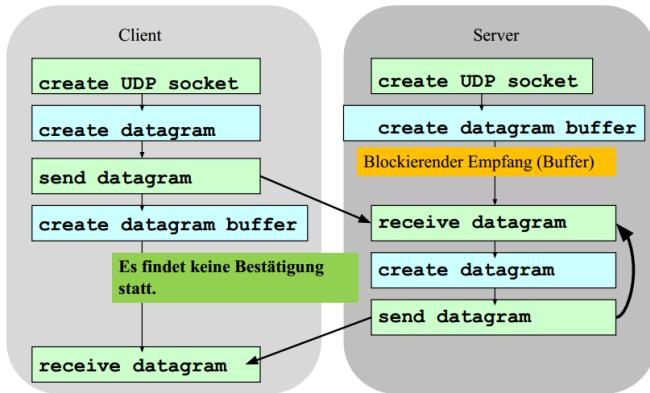


Verbindungsloses Netzwerkprotokoll für den Versand von Datagrammen in IP-basierten Netzen Die Entwicklung von UDP begann 1977, als man für die Übertragung von Sprache ein einfacheres Protokoll benötigte als das bisherige verbindungsorientierte TCP.

- Keine Verzögerungen bei der Sprachübertragung
- Dafür keine Garantie, dass ein Datagramm ankommt
- Keine Garantie, dass Datagramme in der gleichen Reihenfolge ankommen, in der sie gesendet wurden
- Keine Garantie, dass ein Datagramm nur einmal beim Empfänger eintrifft.

Eine Anwendung, die UDP nutzt, muss daher gegenüber verlorengegangenen und unsortierten Paketen unempfindlich sein oder selbst entsprechende Korrekturmaßnahmen und je nachdem auch Sicherungsmaßnahmen vorsehen.

Struktur eines UDP Programms



Beispiel - Sender und Empfänger

Sender

```

public class UDPSender {
    private final String TARGET_HOST = "localhost";
    private final int TARGET_PORT = 9191;
    DatagramSocket socket;
    public UDPSender(String[] args) {
        try {
            socket = new DatagramSocket();
            String message;
            DatagramPacket packet;
            BufferedReader in = new BufferedReader(new InputStreamReader(
                System.in));
            message = in.readLine();
            byte[] data = new byte[message.length()];
            data = message.getBytes();

            packet = new DatagramPacket(data, data.length,
                InetAddress.getByName(TARGET_HOST), TARGET_PORT);
            try {
                socket.send(packet);
            } catch (IOException e) {
                e.printStackTrace();
            }
            socket.close();
        } catch (Exception e) {
            System.err.println(e);
        }
    }

    public static void main(String[] args) {
        new UDPSender(args);
    }
}

```

Empfänger

```

public class UDPReceiver {
    private final int TARGET_PORT = 9191;
    private final int BUFFER_LENGTH=256;

    public UDPReceiver(String[] args) {
        System.out.println("[UDPReceiver]main()");
        try {
            byte[] inbuf;
            DatagramSocket socket = new DatagramSocket(TARGET_PORT);
            String msg=null;
            do {
                inbuf= new byte[BUFFER_LENGTH];
                DatagramPacket packet = new DatagramPacket(inbuf, inbuf.length);
                socket.receive(packet);
                msg=new String(inbuf.trim());
                System.out.println("[UDPReceiver]Message: " +
                    + msg);
            } while (!msg.equals("."));
        } catch (SocketException e) {
        } catch (IOException e) {
        }
    }

    public static void main(String[] args) {
        new UDPReceiver(args);
    }
}

```

Da mit UDP vor Übertragungsbeginn nicht erst eine Verbindung aufgebaut werden muss, kann der Datenaustausch schneller beginnen. *Das fällt vor allem bei Anwendungen ins Gewicht, bei denen nur kleine Datenmengen ausgetauscht werden müssen.*

Einfache Frage-Antwort-Protokolle wie DNS (Domain Name System) verwenden UDP, um die Netzwerkbelastung gering zu halten und damit den Datendurchsatz zu erhöhen. *Ein Drei-Wege-Handschlag wie bei TCP (dem Transmission Control Protocol) für den Aufbau der Verbindung würde unnötigen Overhead erzeugen.*

Daneben bietet die ungesicherte Übertragung auch den Vorteil von geringen Übertragungsverzögerungsschwankungen: Geht bei einer TCP-Verbindung ein Paket verloren, wird es automatisch neu angefordert. Das braucht Zeit, die Übertragungsdauer kann daher schwanken, was für Multimediaanwendungen schlecht ist. Bei VoIP z. B. käme es zu plötzlichen Aussetzern, bzw. die Wiedergabepuffer müssten größer angelegt werden. *Bei UDP bringen verlorengegangene Pakete nicht die gesamte Übertragung ins Stocken, sondern vermindern lediglich die Qualität.*

Bei Übertragungsfehlern oder bei Überlast löscht IP Pakete. *Datagramme können daher fehlen. UDP bietet hierfür keine Erkennungs- oder Korrekturmechanismen, wie etwa TCP.*

Im Falle von mehreren möglichen Routen zum Ziel kann IP bei Bedarf neue Wege wählen. Dadurch ist es in seltenen Fällen möglich, dass später gesendete Daten früher gesendete überholen. Außerdem kann ein einmal abgesendetes Datenpaket mehrmals beim Empfänger eintreffen.

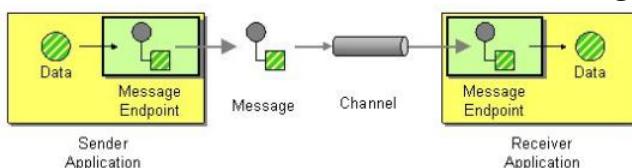
TCP/IP vs. UDP/IP

- Verbindungsorientiert vs. Nicht-verbindungsorientiert
- IP-Level Addressing und Routing via DNS, ARP, etc.
- Unterschiede in den Quality of Service (QoS)
 - o Sequencing
 - o How to deal with packet loss?
- Broadcast und Multicast möglich

Message Exchange Patterns

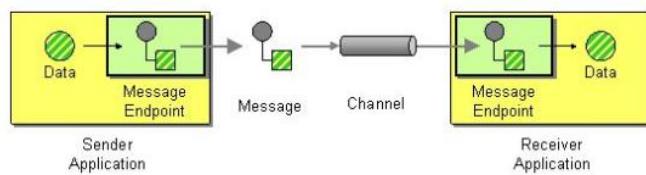
Basic Messaging Pattern

Applikationen kommunizieren indem sie Nachrichten senden über Message Channels. Die Applikation weiss über das Message Format, Channels, oder andere Details zur Verbindung.



Blocking Receiver Message Pattern

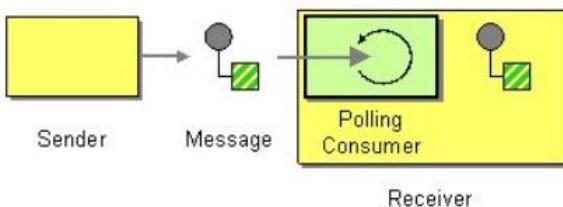
Synchrone Nachrichtenendpunkte, welche solange blockieren bis eine Nachricht empfangen wird.



Polling (Non-Blocking) Receiver Message Pattern

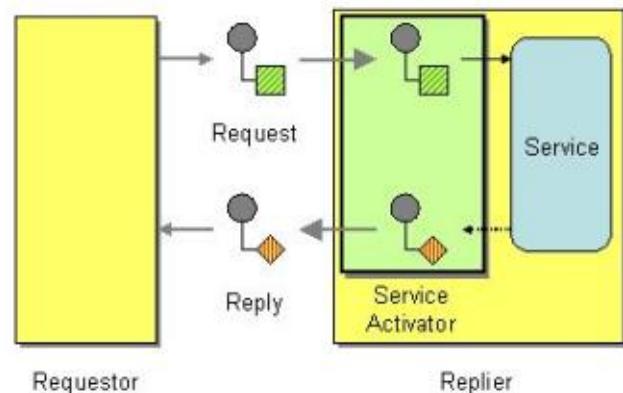
Pattern

Polling Consumer, einer der explizit einen Aufruf macht, wenn er eine Nachricht empfangen möchte.



Service Activator Message Pattern

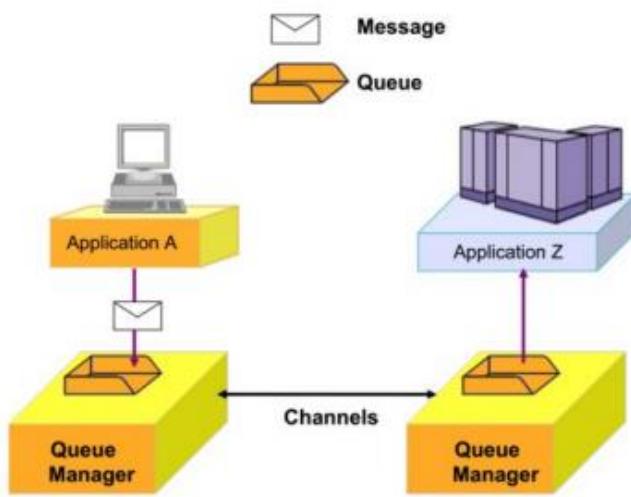
Design und Implementation eines Service Activator, welcher die Nachrichten zu den Channels verbinden (die durch die Services genutzt werden). Dieser kann one-way oder Two-Way sein.



Asynchronous Messaging

Einführung in Messaging

Nachrichtenkonzept und die Komponenten



Message Descriptor

Identifiziert die Nachricht und enthält Steuerinformationen (z. B. Nachrichtentyp und Priorität), die der Nachricht durch die sendende Anwendung zugewiesen wurde.

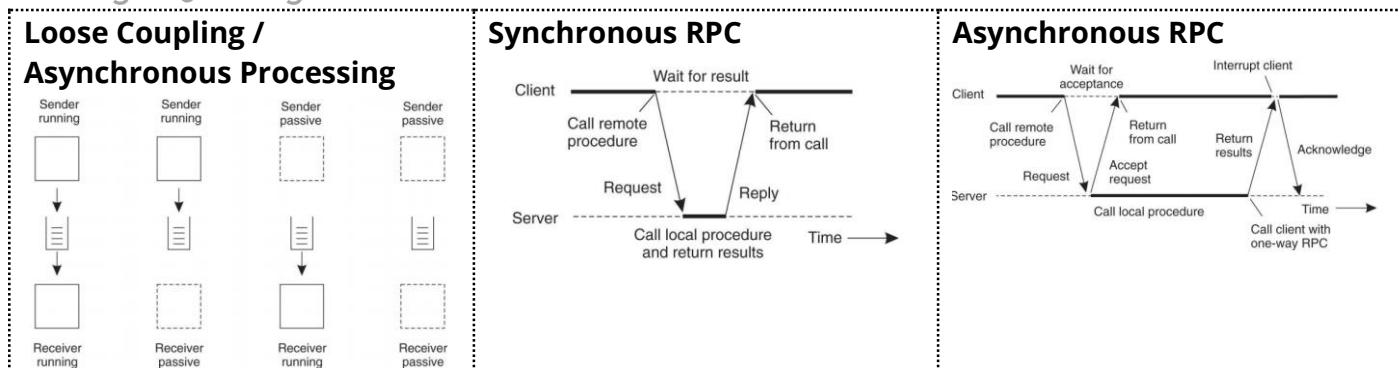
Message data

Enthält die Anwendungsdaten. Die Struktur der Daten wird durch die Anwendungsprogramme definiert, die sie verwenden, und der Warteschlangenmanager ist weitgehend unbekümmert mit seinem Format oder Inhalt.

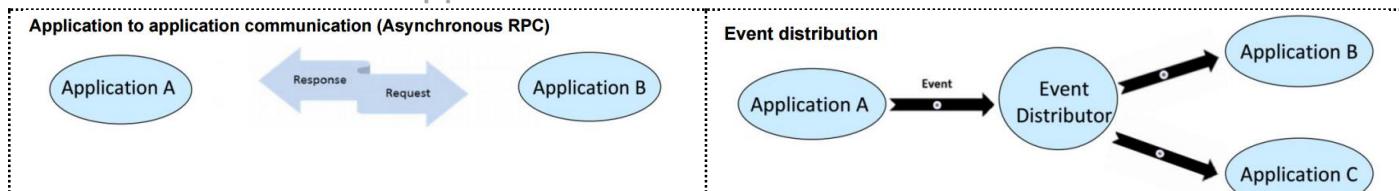
Queue Manager

Ist verantwortlich für das Akzeptieren und Zustellen der Nachrichten. Er verwaltet die Queues aller Nachrichten, welche warten. Warteschlangenmanager sind über ein Kommunikationsnetzwerk über logische Kanäle verbunden. Nachrichten fliessen automatisch über diese Kanäle vom Produzenten einer Nachricht an den Verbraucher dieser Nachricht, basierend auf der Konfiguration der Warteschlangenmanager in der Infrastruktur.

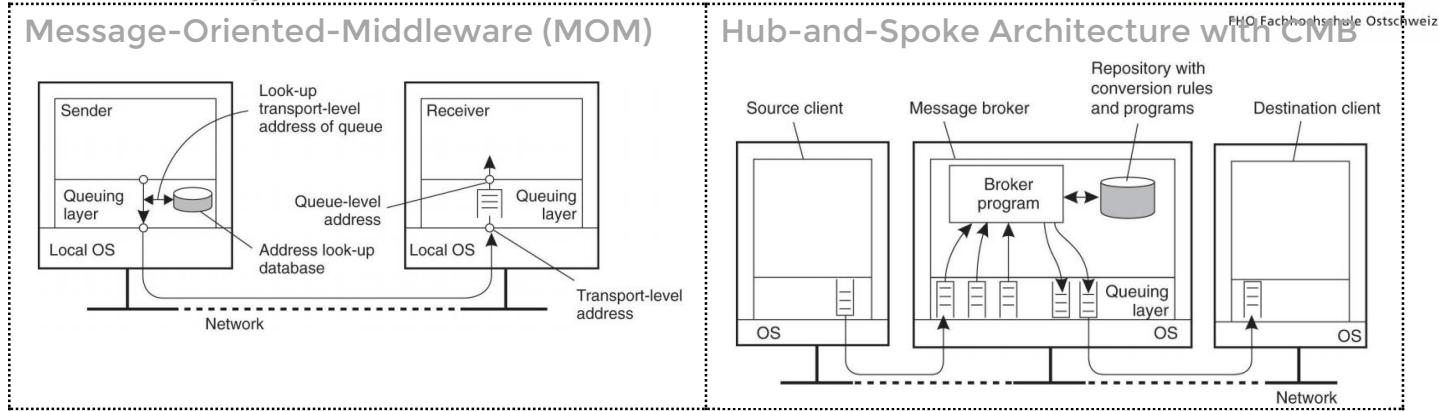
Message-Queuing Model und RPC's



Nachrichten nutzen um Applikationen zu verbinden und Daten zu verteilen



Asynchrone Verarbeitung kann über verschiedene Wege genutzt werden. Warteschlange große Mengen an Arbeit, die dann einmal pro Tag durch einen Batch-Prozess eingereicht wird. Die Arbeit kann auch sofort eingereicht und dann verarbeitet werden, wenn die empfangende Bewerbung den Umgang mit einer vorherigen Anfrage beendet hat. Asynchrone Verarbeitung impliziert nicht unbedingt alle Antwortzeiten.



Java Message Service (JMS) API

Point-to-Point Channel über die Queue API, Publish-Subscribe Channel über die Topic API.

Attribute	Description
Transport type	Determines the transport protocol to be used
FIFO delivery	Indicates that messages are to be delivered in the order they are sent
Message length	Maximum length of a single message
Setup retry count	Specifies maximum number of retries to start up the remote MCA
Delivery retries	Maximum times MCA will try to put received message into queue

Reliability levels

Best effort nonpersistent

Nachrichten werden verworfen, wenn eine Messaging-Engine stoppt oder fehlschlägt. Nachrichten können auch verworfen werden, wenn eine Verbindung, die verwendet wird, um sie zu senden, nicht verfügbar ist oder als Ergebnis von eingeschränkten Systemressourcen.

Express nonpersistent

Nachrichten werden verworfen, wenn eine Messaging-Engine stoppt oder fehlschlägt. Nachrichten können auch verworfen werden, wenn eine Verbindung, die verwendet wird, um sie zu senden, nicht verfügbar ist.

Reliable nonpersistent

Nachrichten werden verworfen, wenn eine Messaging-Engine stoppt oder fehlschlägt.

Reliable persistent

Nachrichten können verworfen werden, wenn eine Messaging-Engine ausfällt.

Assured persistent

Nachrichten werden nicht verworfen.

JMS Message-Struktur

Eine JMS-Message besteht aus a) Header, b) Properties und c) Body. Ein Header wird immer gesendet. Er enthält Informationen für das Routing und die Identifikation. Ein MOM-Provider normiert seine Message Header. Properties sind optional – sie enthalten zum Beispiel Informationen, mit deren Hilfe ein Customer Nachrichten filtern oder weiterrouten kann (Erweiterung des Standard-Headers). Der Body ist auch zwingend erforderlich – er enthält die auszutauschenden Nutzdaten (Text, Objekte, Binärdaten).

```
ActiveMQObjectMessage
{
    commandId = 5, responseRequired = true, messageId = ID:BB-PT-115631-50798-1393663530103-1:1:1:1,
    originalDestination = null, originalTransactionId = null,
    producerId = ID:BB-PT-115631-50798-1393663530103-1:1:1,
    deliveryCount = 0, durable = false, expiration = null, timestamp = 1393663530476,
    arrivalTime = 1393663530480, brokerInTime = 1393663538000, correlationId = null,
    replyTo = null, persistent = true, type = null, priority = 4, groupID = null, groupSequence = 0, targetConsumerId = null,
    redeliveryCounter = 0, size = 0,
    properties = null, userProperties = null, dropable = false,
    content = org.apache.activemq.util.ByteSequence@6218741, marshalledProperties = null, dataStructure = null,
    deliveryCount = 0, size = 0,
    properties = null, userProperties = null, dropable = false,
    payload = {"customerName": "Hans Mustermann", "customerAddress1": "Hirschenstrasse 1", "customerAddress2": "CH-00000
    Kanton", "contractNumber": "501", "additionalInformation": "Hans Mustermann", "insuranceType": "L"}
}
```

Queue Sender

```

Connection connection;
connection = connectionFactory.createConnection();
connection.start();
Session session =
  connection.createSession(mp.transacted,
    Session.AUTO_ACKNOWLEDGE);
Destination destination =
  session.createQueue("testQueue");
MessageProducer producer =
  session.createProducer(destination);
producer.setDeliveryMode(DeliveryMode.PERSISTENT);
TextMessage message =
  session.createTextMessage("hello");

producer.send(message);
producer.close();
session.close();
connection.close();
  
```

Queue Receiver

```

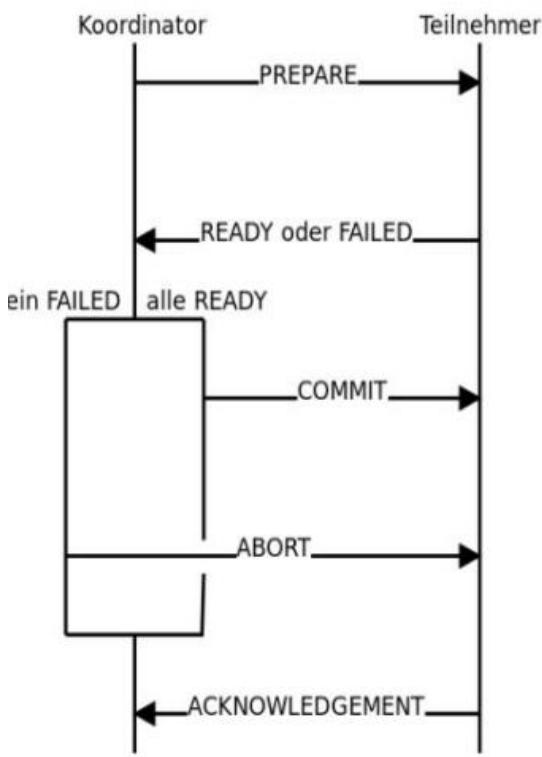
Connection connection;
connection = connectionFactory.createConnection();
connection.start();
Session session =
  connection.createSession(mc.transacted,
    Session.AUTO_ACKNOWLEDGE);
Destination destination =
  session.createQueue("testQueue");
MessageConsumer consumer =
  session.createConsumer(destination);
TextMessage text=(TextMessage) consumer.receive();
consumer.close(); session.close(); connection.close();
  
```

Andere APIs oder Provider

- Provider, welche die Java Messaging Service (JMS) Spezifikation beinhalten
 - o Apache ActiveMQ
 - o Any Java Enterprise Edition (JEE) Application Server: IBM, JBoss, Oracle
- RabbitMQ (eigene API, nicht kompatibel zu JMS)
 - o Open Source, Implementiert das AMQP Protokoll.
- Cloud Provider z.B. Amazon Simple Queueing Server (SQS)
- Apache Camel oder Sprint Integration
- ZeroMQ
 - o Alternativer Weg zum Queue basierten Nachrichten.

Messaging Middleware Provider**IBM MQ**

Ermöglicht es, Anwendungen asynchron zu kommunizieren, d.h. Nachrichten können zu unterschiedlichen Zeiten gesendet, empfangen und verarbeitet werden. Kann jede Art von Daten als Nachrichten transportieren. Arbeitet mit einer breiten Palette von Computing-Plattformen, Anwendungen, Web-Services und Kommunikationsprotokolle.



Erste Phase

Der Koordinator sendet ein prepare an alle Teilnehmer. Die Teilnehmer verarbeiten die Transaktion bis zu dem Punkt, wo die Transaktion entweder mit commit oder rollback abgeschlossen wird. Dabei schreiben sie Einträge in ihr undo log und in ihr redo log. Die Teilnehmer antworten mit ready, wenn die Transaktion erfolgreich war - oder sie antworten mit failed, wenn die Transaktion fehlgeschlagen ist.

Zweite Phase

Wenn der Koordinator von allen Teilnehmern eine ready Meldung bekommen hat: Der Koordinator sendet commit an alle Teilnehmer. Die Teilnehmer schließen die Transaktion mit commit ab und geben alle Locks und Ressourcen frei. Die Teilnehmer senden ein acknowledgment zurück. Der Koordinator beendet die Transaktion, wenn er von allen Teilnehmern die Bestätigung erhalten hat.

Wenn zumindest einer der Teilnehmer ein failed schickt: Der Koordinator sendet abort an alle Teilnehmer. Die Teilnehmer schließen die Transaktion mit rollback ab (mittels des undo logs) und geben alle Locks und Ressourcen frei. Die Teilnehmer senden ein acknowledgment zurück. Der Koordinator beendet die Transaktion ebenso mit rollback, wenn er von allen Teilnehmern die Bestätigung erhalten hat.

Persistente und nicht persistente Nachrichten

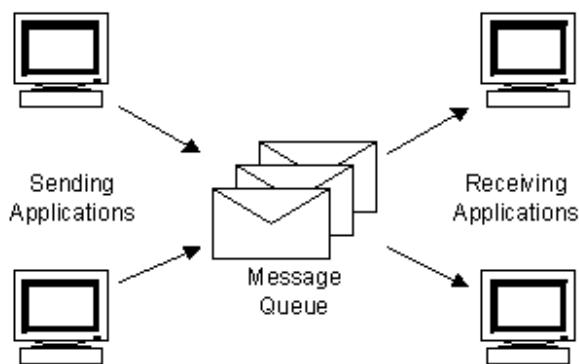
Meldungen, die kritische Geschäftsdaten enthalten, wie z. B. Zahlungseingang für eine Bestellung, sollten zuverlässig aufrechterhalten werden und dürfen im Falle eines Fehlers nicht verloren gehen. Einige Nachrichten können nur Abfragedaten enthalten, wobei der Verlust der Daten nicht entscheidend ist, da die Abfrage wiederholt werden kann. In diesem Fall gilt die Leistung als wichtiger als die Datenintegrität.

Persistente Nachrichten

IBM MQ verliert keine anhaltende Nachricht durch Netzwerkfehler, Lieferfehler oder Neustart des Warteschlangenmanagers. Jeder Warteschlangenmanager hält ein fehlertolerantes Wiederherstellungsprotokoll aller Aktionen, die bei persistenten Nachrichten ausgeführt werden. Dieses Protokoll wird manchmal als Zeitschrift bezeichnet

Nicht-persistierende Nachrichten

IBM MQ optimiert die Aktionen, die auf nicht persistenten Nachrichten für die Leistung durchgeführt werden. Die nicht-persistenten Nachrichtenspeicherung basiert im Systemspeicher, so dass es möglich ist, dass sie in Situationen wie Netzwerkfehlern, Betriebssystemfehlern, Hardwarefehlern, Warteschlangenmanager-Neustart und internem Softwarefehler verloren gehen können.



und von mehreren empfangenden Anwendungen gelesen werden. Es ist Bestandteil von Windows aber nicht installiert/aktiviert im Default Setup.

Active MQ

MOM API Primitives (Platform-Independent Level)

Write (Put), Consuming Read/Receive (Get) – Synch and asynchronous (event-or callback-based), Non-Consuming Read/Receive (Browse) – Like top() in stack.

Primitive	Meaning
Put	Append a message to a specified queue
Get	Block until the specified queue is nonempty, and remove the first message
Poll	Check a specified queue for messages, and remove the first. Never block
Notify	Install a handler to be called when a message is put into the specified queue

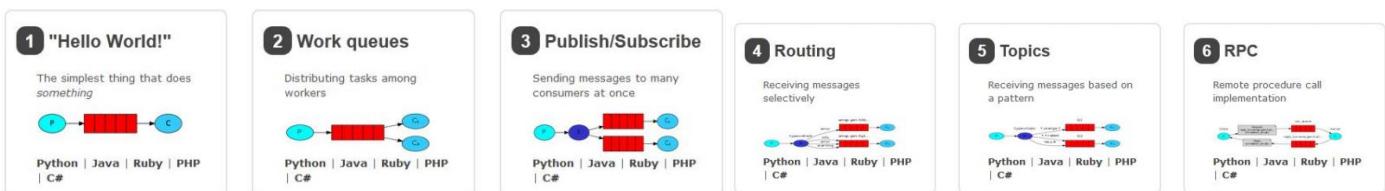
Enterprise Integration Patterns

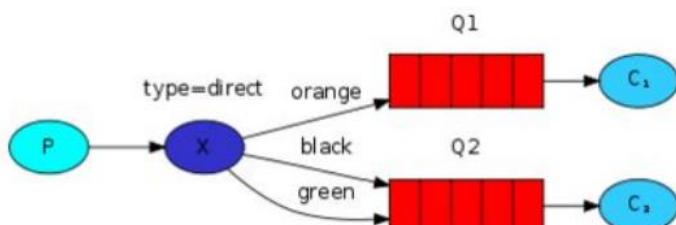
Siehe Vorlesungfolien.

RabbitMQ (based on AMQP)

Patterns

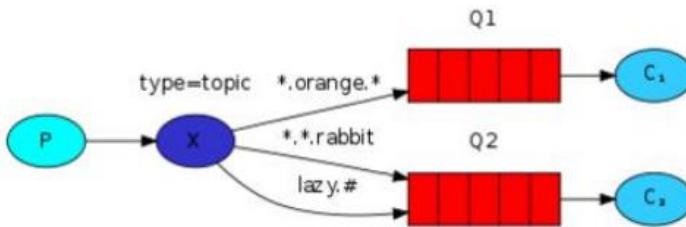
Basierend auf dem AMQP (Advanced Message Queueing Protocol)





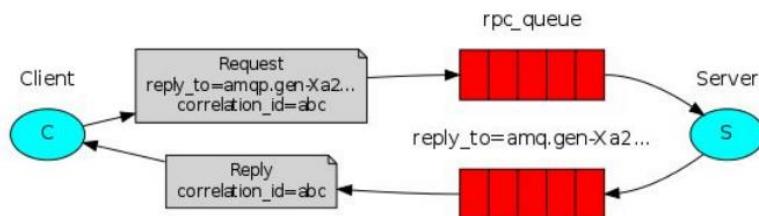
Beispiel von einem direkten Austausch mit zwei gebundenen Warteschlangen. Die erste Warteschlange ist mit dem Key «Orange» gebunden, die zweite Warteschlange hat zwei Bindings von schwarz und green. Die Zustellung erfolgt entsprechend den Bindings. Alle anderen Nachrichten werden verworfen.

Topics



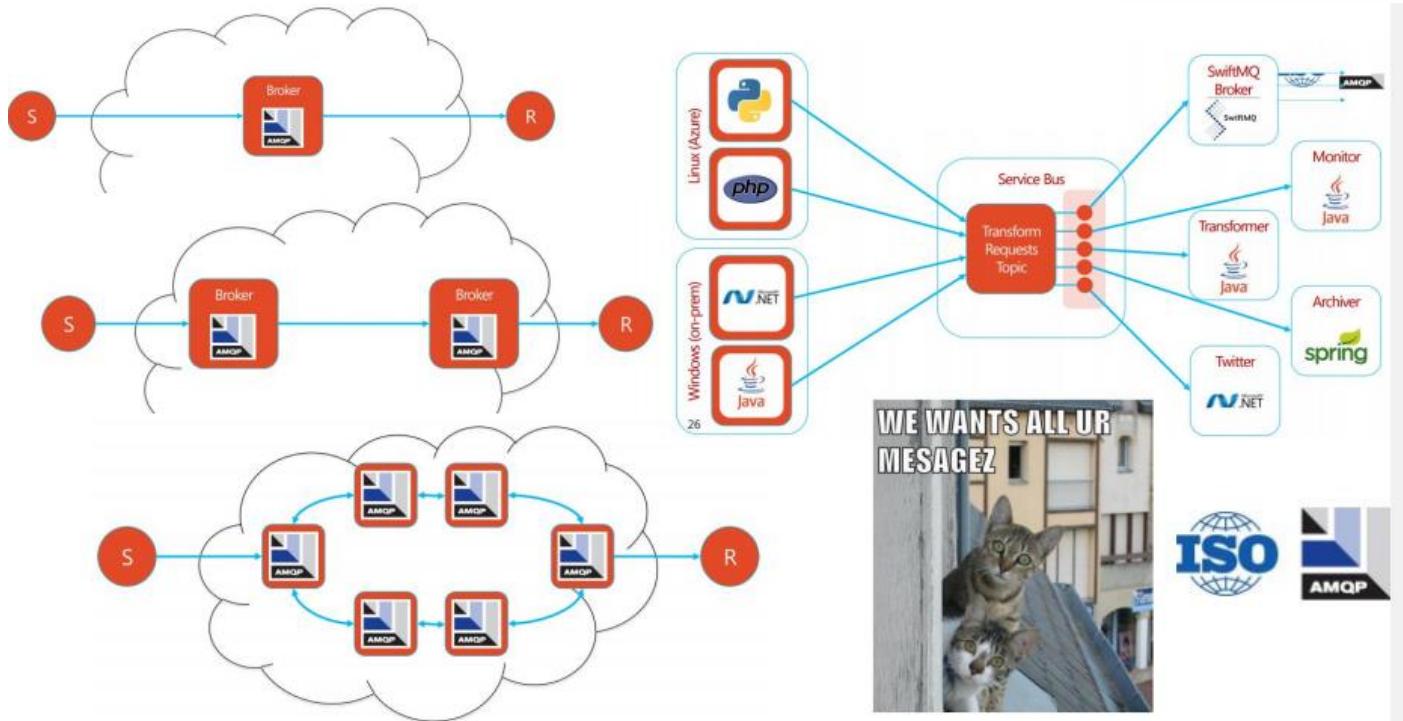
In this example, messages describe animals. Messages have a routing key that consists of three words .. Q1 is bound "`*.*.orange.*`" and Q2 with "`*.*.rabbit`" and "`lazy.#`". Q1 is interested in all the orange animals. Q2 wants to hear everything about rabbits, and everything about lazy animals. Messages with "`quick.orange.rabbit`" will be delivered to both queues. Messages "`lazy.orange.elephant`" also will go to both of them. "`quick.orange.fox`" will only go to the first queue, and "`lazy.brown.fox`" only to the second. "`lazy.pink.rabbit`" will be delivered to the second queue only once, even though it matches two bindings. "`quick.brown.fox`" doesn't match any binding so it will be discarded.

RPC



Wenn der Client startet, erstellt er eine anonyme, exklusive Callback-Warteschlange. Für eine RPC-Anforderung sendet der Client eine Nachricht mit zwei Eigenschaften: `reply_to`, die auf die Callback-Warteschlange und `correlation_id` gesetzt ist, die für jede Anfrage auf einen eindeutigen Wert gesetzt wird. Die Anforderung wird an eine `rpc_queue`-Warteschlange gesendet. Der RPC-Arbeiter (aka: server) wartet auf Anfragen auf diese Warteschlange. Wenn eine Anforderung erscheint, führt sie den Job aus und sendet eine Nachricht mit dem Ergebnis an den Client zurück, wobei die Warteschlange aus dem Feld `reply_to` verwendet wird. Der Client wartet auf Daten in der Rückrufwarteschlange. Wenn eine Meldung erscheint, prüft sie die Eigenschaft `correlation_id`. Wenn es dem Wert aus der Anfrage entspricht, gibt er die Antwort an die Anwendung zurück.

Wenn der Client startet, erstellt er eine anonyme, exklusive Callback-Warteschlange. Für eine RPC-Anforderung sendet der Client eine Nachricht mit zwei Eigenschaften: `reply_to`, die auf die Callback-Warteschlange und `correlation_id` gesetzt ist, die für jede Anfrage auf einen eindeutigen Wert gesetzt wird. Die Anforderung wird an eine `rpc_queue`-Warteschlange gesendet. Der RPC-Arbeiter (aka: server) wartet auf Anfragen auf diese Warteschlange. Wenn eine Anforderung erscheint, führt sie den Job aus und sendet eine Nachricht mit dem Ergebnis an den Client zurück, wobei die Warteschlange aus dem Feld `reply_to` verwendet wird. Der Client wartet auf Daten in der Rückrufwarteschlange. Wenn eine Meldung erscheint, prüft sie die Eigenschaft `correlation_id`. Wenn es dem Wert aus der Anfrage entspricht, gibt er die Antwort an die Anwendung zurück.



RMI und Web Services

Remote Procedure Calls (RPC)

Da spricht man von einer Technik zum Aufbau von verteilten Client-Server Applikationen. Es erweitert die Notation der lokalen Aufrufe um die Remote Procedure Aufrufe (Calls) und macht damit das Client-Server Model einfacher zu programmieren.

Es verdeckt somit die Details des Netzwerks und isoliert die Applikation von den physischen und logischen Elementen der Datenkommunikation. Ein RPC ist wie ein Funktionsaufruf, welcher man mit Parametern aufrufen kann.

Events während einem RPC

- Der Client ruft den client stub auf. Dieser Aufruf ist ein lokaler Call, welcher die Parameter auf den Stack legt.
- Der Client stub packt die Parameter in eine Nachricht und macht einen Systemaufruf um die Nachricht zu versenden. Die Packetierung heisst «Marshalling».
- Das lokale OS sendet die Nachricht von der Client Maschine zur Server Maschine.
- Das lokale Server OS gibt die Packete an den Sever Stub weiter.
- Der Server Stub entpackt die Parameter aus der Nachricht.
- Der Server Stub ruft abschliessend die Server Prozedur auf. Die Antwort macht den gleichen Weg, einfach umgekehrt.

Potentielle Fehler

Der Client sendet eine Request-Nachricht zu einem entfernten Server um eine bestimmte Prozedur mit den angegeben Parametern auszuführen. Der Remote Server sendet eine Antwort an den Client, welcher dann sein Programm weiterfährt. Während der Server den Aufruf verarbeitet ist der Client typischerweise geblockt und wartet auf die Antwort. RPC können aufgrund von Netzwerkproblemen fehlschlagen. Der Aufrufer muss also mit solchen Problemen umgehen können ohne zu wissen, dass der RPC nicht angekommen ist..

RMI (Java Remote Method Invocation)

RMI ist ein Teil des OO-Ansatzes von Java. Er kann komplexe Objekte als Argumente und Return-Werte mitgeben und nicht nur vordefinierte Datentypen. So können Sie auch ganze Hash-Tables übermitteln.

Somit ist es möglich die volle Power der OO-Technologie auf in verteilten Systemen zu verwenden. Denn RMI macht es einfach Java Servers und Java Client zu schreiben, welche auf die Server zugreifen. Das Remote Interface ist ein Java Interface. Der Server hat nur ein paar Zeilen Code um den Server zu beschreiben, alles andere wie Java-Objekte.

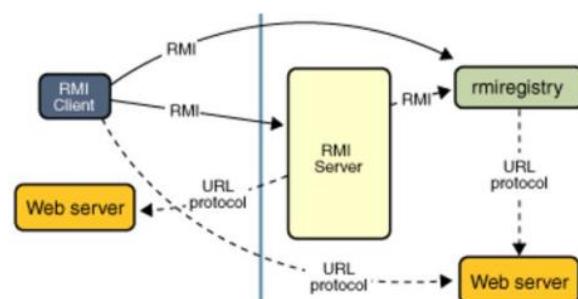
Zwei Programme – Client und Server

Ein typisches Server-Programm erstellt einige entfernte Objekte, macht Verweise auf diese Objekte zugänglich und wartet darauf, dass Clients Methoden auf diesen Objekten aufrufen. Ein typisches Client-Programm erhält eine Remote-Referenz auf ein oder mehrere entfernte Objekte auf einem Server und ruft dann Methoden auf sie auf. RMI stellt den Mechanismus zur Verfügung, mit dem der Server und der Client kommunizieren und Informationen hin und her versenden.

Eine verteilte Objekt-Applikation braucht folgendes:

- Lokalisieren von Remote Objekten
- Kommunikation mit Remote Objekten
- Laden der Klassendefinitionen, von den Objekten welche mitgegeben werden

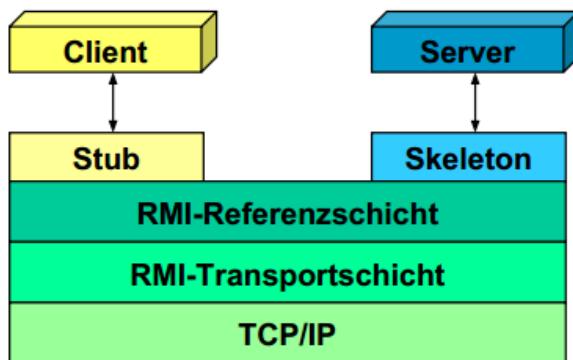
RMI Registry benutzen um eine Referenz eines Java-Objekts zu erhalten



Der Server ruft die Registry auf um den Namen mit einem Remote-Objekt zu binden. Der Client schaut dann nach dem Remote Objekt mit dem Namen in der Server Registry und ruft eine Methode darauf auf. Das RMI System nutzt Webserver um die Klassendefinitionen zu laden (Client-Server und Server-Client).

Businesslogik nur auf dem Server

RMI kann die Klassenimplementierungen vom Client auf den Server verschieben und umgekehrt. Beispielsweise können Sie eine Schnittstelle für die Prüfung von Mitarbeiter-Expense-Reports definieren, um zu sehen, ob sie mit der aktuellen Firmenpolitik übereinstimmen. Wenn ein Expense Report erstellt wird, kann ein Objekt, das diese Schnittstelle implementiert, durch den Client der Server abgerufen werden. Wenn die Richtlinien geändert werden, beginnt der Server, eine andere Implementierung dieser Schnittstelle zurückzugeben, die die neuen Richtlinien verwendet. Die Einschränkungen werden daher auf der Client-Seite überprüft und sorgen für eine schnellere Rückmeldung an den Benutzer und weniger Belastung des Servers - ohne eine neue Software auf dem System des Benutzers zu installieren. Dies gibt Ihnen maximale Flexibilität, da die Änderung von Richtlinien erfordert, dass Sie nur eine neue Java-Klasse schreiben und sie einmal auf dem Server-Host installieren.

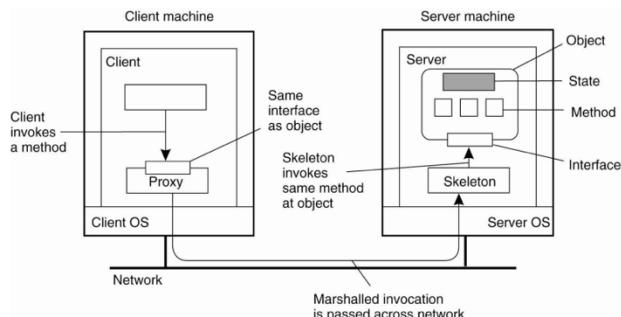


Ein Stub ist ein Stellvertreterobjekt (Remote Proxy), das die Clientaufrufe an den Server weiterreicht.

Ein Skeleton nimmt Aufrufe des Stubs entgegen und leitet sie an ein Serverobjekt weiter.

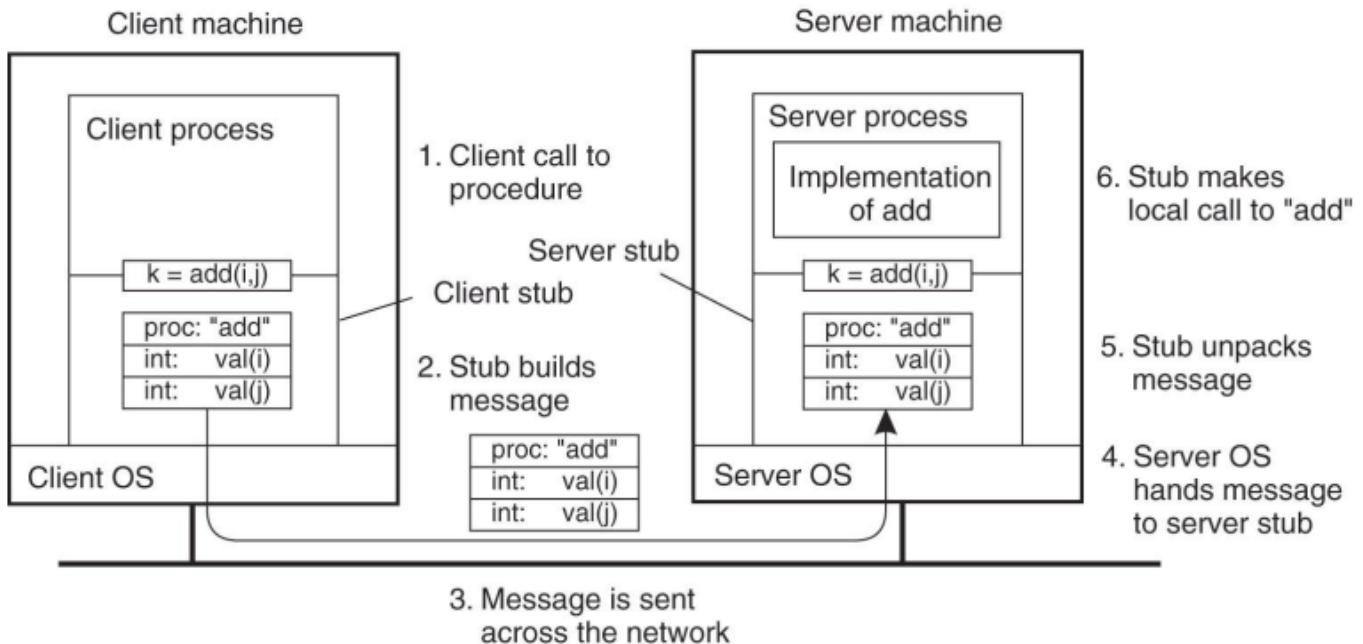
Die RMI-Referenzschicht stellt den Namensdienst (=Registry) zur Verfügung. Die RMI-Transportschicht verwaltet Verbindungen und wickelt die Kommunikation ab.

Klassen und Interfaces



Die Interfaces definieren die Methoden. Die Klassen implementieren die Methode der Interfaces. Ein Objekt wird zu einem Remote-Objekt wenn es das Remote Interface implementiert (java.rmi.Remote).

Innvoltierte Schritte



Schritte vor einem RMI Aufruf

1. Start der RMI-Registry	3. Binden des Server-Objektes	5. Look-up
2. Start des Servers	4. Start des Clients	6. Methoden des Server-Objekts aufrufen

Stub-Klasse in RMI (Remote Proxy)

Die Stub-Klasse baut eine Socket-Verbindung zum Server auf (Connect-Call). Sie schickt Namen der Methode und Parameter und holt das Ergebnis ab.

```
public class HelloImpl_Stub implements Hello {
    Socket socket = new Socket("[hostname]",4711);
    ObjectOutputStream outStream =
        new ObjectOutputStream(socket.getOutputStream());

    // für remote Methode sayHello():
    public String sayHello( ) throws RemoteException{
        outStream.writeObject("sayHello"); // simplified example
        ObjectInputStream in =
            new ObjectInputStream(socket.getInputStream());

        return (String)in.readObject();
    }
}
```

Skeleton-Klasse in RMI

Erzeugt das Server-Socket, der den Port aus dem Stub bindet (bind/listen/accept). Wartet dann auf den Methodenaufruf vom Client und delegiert diesen an ein Objekt. Zu letzt liefert es einen Rückgabewert über die Socketverbindung an den Client zurück.

```
public class HelloImpl_Skeleton extends Thread{
    HelloImpl myHello;
    ObjectInputStream inStream =
        new ObjectInputStream(socket.getInputStream());

    public void run(){
        ServerSocket serverSocket = new ServerSocket(4711);
        Socket socket = serverSocket.accept();

        String method = (String) inStream.readObject(); // simplified ex.
        if(method.equals("sayHello")){
            String returnVal = myHello.sayHello();
            ObjectOutputStream outStream =
                new ObjectOutputStream(socket.getOutputStream());
            outStream.writeInt(returnVal);
        }
    }
}
```

RMI-Programmierung auf der Serverseite

Interface für Remote-Methoden

Wird von den beiden Rollen (Client und Server) benötigt. Es muss von Remote aus dem java.rmi Package abgeleitet sein. Alle Methoden müssen eine RemoteException werden. Die Remote-Interfaces haben fachliche Namen ohne Suffix.

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

public interface Hello extends Remote {
    String sayHello() throws RemoteException;
}
```

Klasse zur Implementierung des Remote-Interfaces

Es implementiert das Interface (in diesem Fall «Hello». Jedes entfernte Objekt ist eine Unterklasse von UnicastRemoteObject. Es wird ein Konstruktor benötigt, welcher die RemoteException wirft.

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

public class HelloImpl extends UnicastRemoteObject implements Hello {
    public HelloImpl() throws RemoteException { super(); }

    public String sayHello() throws RemoteException{
        return "Hello World!";
    }
}
```

Remote-Objekte erzeugen und in RMI-

Registry anmelden

Auf dem Server-Rechner wird ein Objekt der Implementierungsklasse erzeugt. Es wird unter dem Namen remoteHello beim Rechner registriert. Standard-Port-Nummer für den rmiRegistry-Prozess auf dem Server ist 1099.

```
import java.rmi.Naming;

public class HelloServer {
    public static void main(String args[]) {
        try {
            HelloImpl obj = new HelloImpl();
            Naming.rebind("rmi://[hn]/remoteHello", obj);
        } catch (Exception e) { ... }
    }
}
```

RMI-Programmierung auf dem Client

Remote-Interface definieren

```
public interface Hello extends Remote {analog zum Server}
```

Lookup des Remote Objekts remoteHello in der Registry

- Es wird das Remote-Interface verwendet (Packagename muss passen!)
- Registrieren des **RMISecurityManager**-Objekts erforderlich, wenn der Client den Stub-Code dynamisch vom Server laden will

```
import java.rmi.*;
public class RmiClient {
    public static void main(String[] args) {
        try {
            //System.setSecurityManager(new RMISecurityManager());
            Hello obj = (Hello)Naming.lookup("rmi://[hn]/remoteHello");
            String message = obj.sayHello();
            System.out.println(message);
        } catch (Exception e) { ... }
    }
}
```

Methode des Remote-Objects aufrufen

Schwächen von RMI

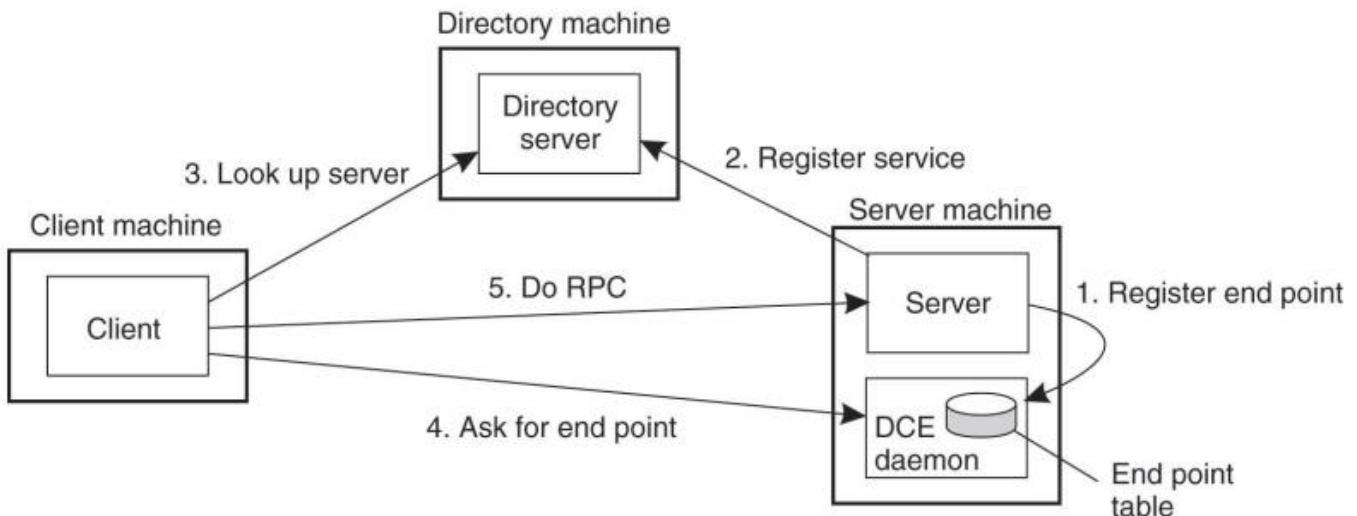
- «The exception that grounded an airline» (Kein Timeout Management.)
- Java versions differ in the way they implement RMI
- All CORBA problems when RMI is used with Java Enterprise Edition
- Practical project experience → nur wenige nutzen es in purer Form
 - MOM oder http werden heute bevorzugt.

Interface Description Languages (IDLs)

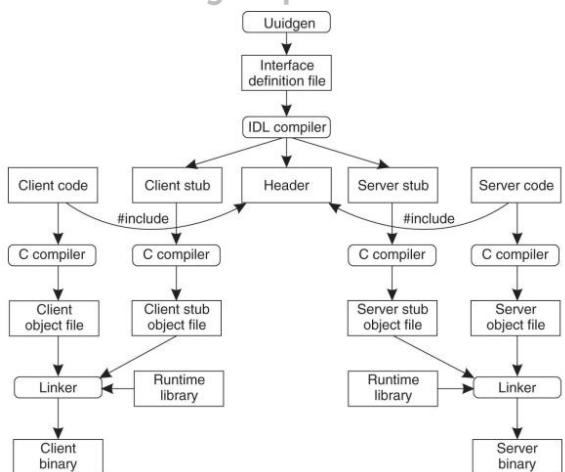
Einführung

Ein IDL ist eine Spezifikationssprache, mit der Programmierschnittstellen sprachunabhängig beschrieben werden können. IDLs werden verwendet, um die Kommunikation zwischen Clients und Servern in Remote Procedure Calls (RPC) aufzubauen.

DCE (Distributed Computing Environment) from the 1990s



IDL Processing Steps



IDL Verwendung

OMG IDL basiert auf DCE IDL von den 1990s. Es benutzt CORBA und unterstützt den RMI Compiler und die .NET Tools. Es hat ein eigenes Typensystem welches die Programmierspezifikationen abstrahiert.

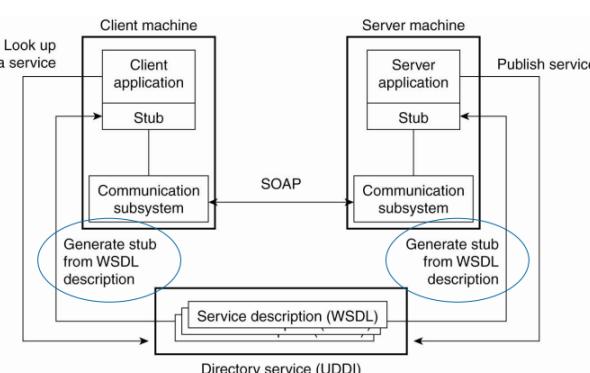
Web Services

Einführung

WSDL ist eine XML-basierte Schnittstellendefinitionssprache, die verwendet wird, um die Funktionalität eines Webdienstes zu beschreiben. Maschinell lesbare Beschreibung, wie der Service

aufgerufen werden kann, welche Parameter es erwartet und welche Datenstrukturen es zurückgibt. Standard, maschinenlesbares Format, aus dem Clients und Skelett eines Service automatisch erstellt werden können. Entwickelt wurde es von IBM, Microsoft und Ariba.

Architektur



RPC Style SOAP Nachrichten

Es ist um die Annahme gebaut, dass du den Web-Service anrufen möchtest, genau wie du eine normale Funktion oder Methode nehmen würdest, die Teil deines Anwendungscodes ist. Der Nachrichtentext enthält für jeden "Parameter" der Methode ein XML-Element. Diese Parameter-Elemente werden in ein XML-Element eingehüllt, das den Namen der aufgerufenen Methode enthält.

Document Style Web Service

Es erlaubt Ihnen, alle Daten, die Sie wollen, in XML einzuschließen und auch ein Schema für dieses XML zu enthalten. Dies bedeutet, dass der Client- und Server-Anwendungscode die Marshalling- und Unmarshalling-Arbeit durchführen müssen. Dies steht im Gegensatz zu RPC, in dem der Marshalling / Unmarshalling-Prozess von der SOAP-Bibliothek abgewickelt wird.

Example of an RPC Style SOAP request message:

```
<soap:envelope>
  <soap:body>
    <multiply>  <!-- web method name -->
      <a>2.0</a> <!-- first parameter -->
      <b>7</b>   <!-- second parameter -->
    </multiply>
  </soap:body>
</soap:envelope>
```

Example of a Document Style SOAP request message:

```
<soap:envelope>
  <soap:body>
    <!-- arbitrary XML -->
    <movies
      xmlns="http://www.myfavoritemovies.com">
      <movie>
        <title>2001: A Space Odyssey</title>
        <released>1968</released>
      </movie>
      <movie>
        <title>Donnie Darko</title>
        <released>2001</released>
      </movie>
    </movies>
  </soap:body>
</soap:envelope>
```

Vor- und Nachteile von RPC und Document WSDL Styles

RPC WSDL-Stil schafft eine enge Kopplung zwischen Dienstanbieter und Client. Das Ändern der Reihenfolge der Parameter oder der Typen der Parameter ändert die Definition des Web Service selbst (so wie es die Definition einer normalen Funktion oder Methode beeinflusst).

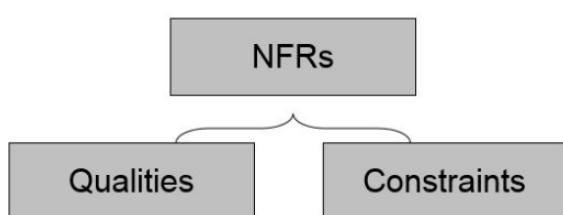
Dokument WSDL-Stil-Services implementieren eine lose gekoppelte Art der Interaktion zwischen Dienstanbieter und Client. Der Hauptunterschied des Dokumentenstils auf den RPC-Stil besteht darin, dass der Client die Serviceparameter in einem normalen XML-Dokument an den Server sendet, anstatt einen diskreten Satz von Parameterwerten von Methoden. Es gibt keine direkte Zuordnung zwischen den Serverobjekten (Parameter, Methodenaufrufe usw.) und den Werten in XML-Dokumenten. Der Hauptnachteil des Dokumentenstils besteht darin, dass es keine Standardmethode gibt, welche Methode des Webdienstes die Anfrage beantragt.

Schwächen von Web Services

- Ausführlichkeit von XML
 - o SOAP envelope size (normalerweise 4-20 mal grösser als der Payload)
- WSDL Sprache Design
 - o Es gibt zu viele redundante Elemente in WSDL 1.1
 - o Von WSDL 2.0 gibt es nur wenige Implementationen
- Notwendigkeit von Spezifikationen und Generatoren
 - o Die REST Community sagt das es die formalen Service Contracts und die Code Generation nicht braucht.

Operationale Modelle, Design for Performance

Non-Functional Requirements (NFRs)



Nicht-funktionale Anforderungen (oder NFRs) definieren die wünschenswerten Qualitäten eines Systems und die Einschränkungen, innerhalb derer das System aufgebaut werden muss

Qualitäten definieren die Eigenschaften und Merkmale, die das gelieferte System nachweisen sollte.

Einschränkungen sind die Einschränkungen, Standards und Umweltfaktoren, die in der Lösung berücksichtigt werden müssen.

Constraints



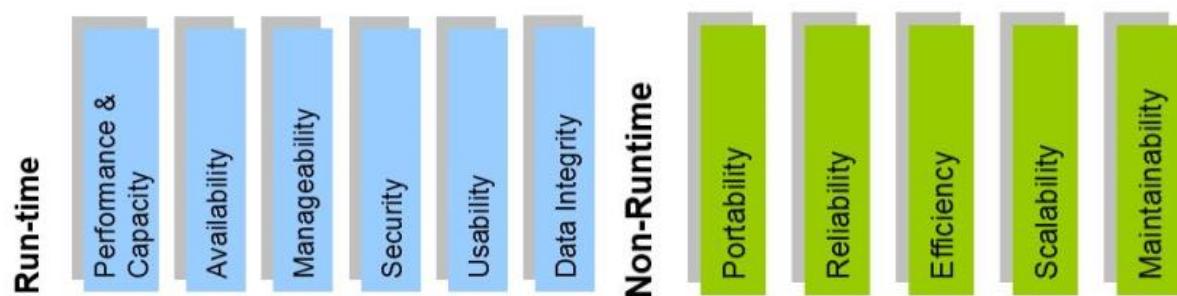
Die Geschäftsaspekte des Projekts, die Geschäftsumgebung des Kunden oder die IT-Organisation, die die Architektur beeinflussen. Das technische Umfeld und die vorherrschenden Standards, die das System und das Projekt in Betrieb nehmen müssen.

Qualities

Runtime-Qualitäten sind "messbare" Eigenschaften, die oft als "Service Level Requirements" ausgedrückt werden.

Qualitäten könnten auch mit den Entwicklungs-, Wartungs- oder Betriebsanliegen zusammenhängen, die zur Laufzeit nicht

ausgedrückt werden.



SEI Software Engineering Institute

Quality Attributes des Technical Reports

Der Report stammt zwar aus dem Jahre 1995, es ist aber zu grossen Teilen heute immer noch aktuell. Zu den Attributen zählen folgende Begriffe: Availability, Compliance, Maintainability, Performance, Privacy, Recovery, Resilience, Scalability und Usability.

Relevante Standards

- NFR nach ISO 9126
- ISO 25010 (Mehrheitlich eine Aktualisierung des ISO 9126)

Trotz fortgesetzter technologischer Fortschritte verbringen IT-Architekten erheblichen Zeitaufwand um den Anforderungen an Servicequalität gerecht zu werden. Die aufgrund der signifikanten Einschränkungen. Software sowie Infrastruktur Design soll iterativ geplant werden um dieses Ziel zu erreichen.

Nicht-funktionale Anforderungen und Service-Levels können vertraglich bindend sein. Eine Nicht-Einhaltung kann in der IT einen Penalty bedeuten und resultiert dann in einer Strafzahlung.

Modellierungstheorie, Techniken und Werkzeuge stehen zur Verfügung, um bei der Bewertung von Designalternativen zu helfen. Unabhängig von der Qualität des Designs muss die Qualität der Umsetzung durch Testing überprüft werden.

Der Aufwand sollte immer proportional zum Risiko sein.

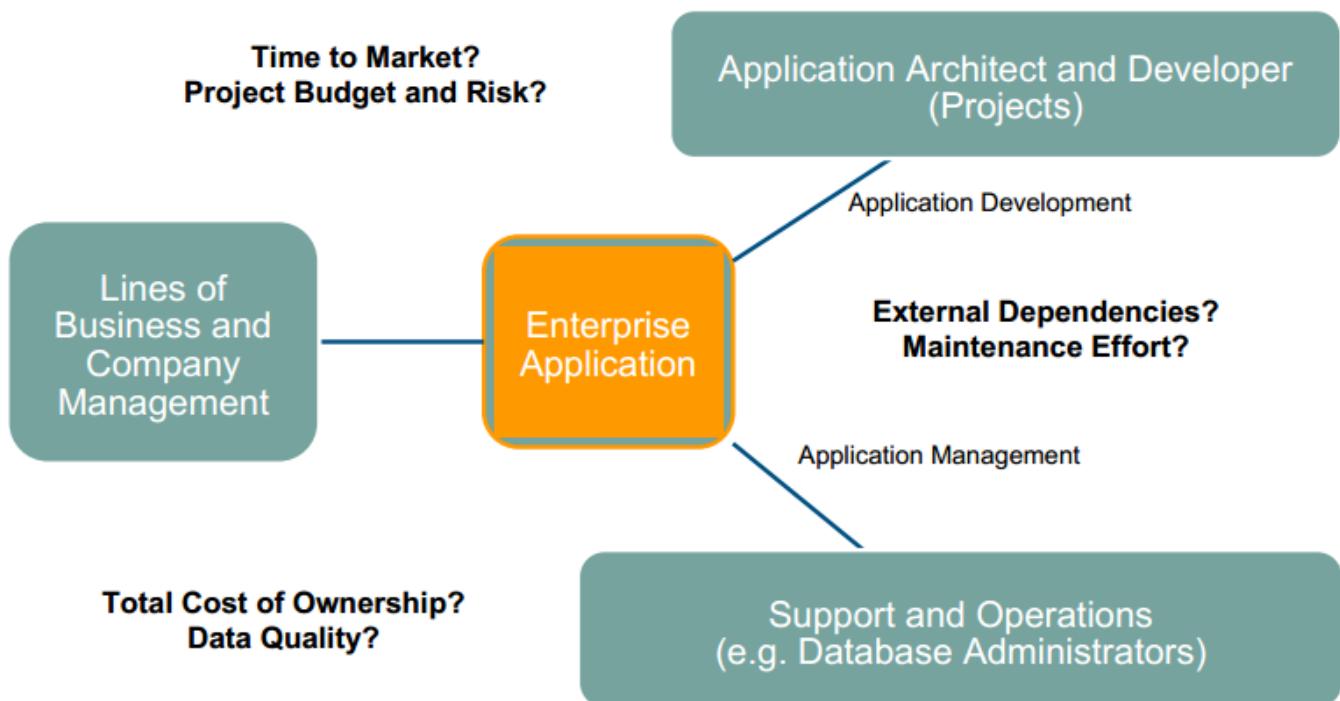
Beispiel für Design Tradeoff – Security vs. Performance

Ein Kompromiss ist eine Situation, bei der es darum geht, eine Qualität oder einen Aspekt von etwas im Gegenzug zu verlieren, um eine andere Qualität oder einen Aspekt zu gewinnen.

Herausforderungen für Architekten und Designer

- Verschiedene NFRs können miteinander in Konflikt stehen
- Verschiedene Stakeholder können unterschiedliche Prioritäten haben
- Unzureichende Informationen stehen zur Verfügung
- Informationen können sich ändern

Stakeholders und ihre Bedenken



Die beste Technik zur Verringerung des Risikos einer schlechten Dienstqualität ist es, die Qualitäten von Anfang an zu betrachten. Fixen Sie es früh und sparen sich damit Geld und Probleme, welche später auftauchen.

Definition

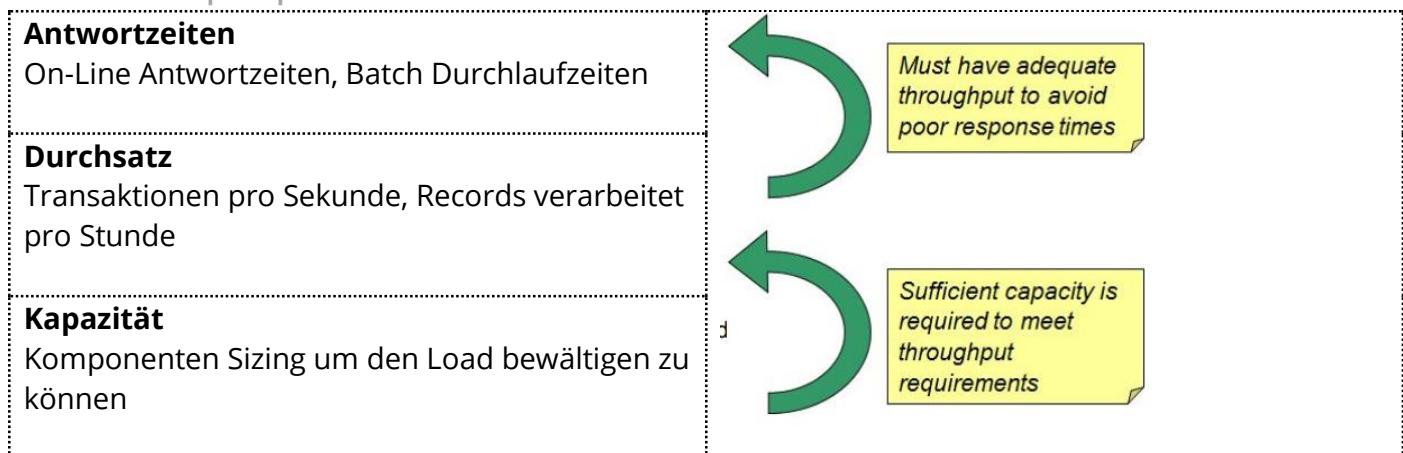
"Performance. Der Grad, zu dem ein System oder eine Komponente seine beabsichtigten Funktionen innerhalb vorgegebener Einschränkungen, wie Geschwindigkeit, Genauigkeit oder Speicherverbrauch, erfüllt."

Im Allgemeinen

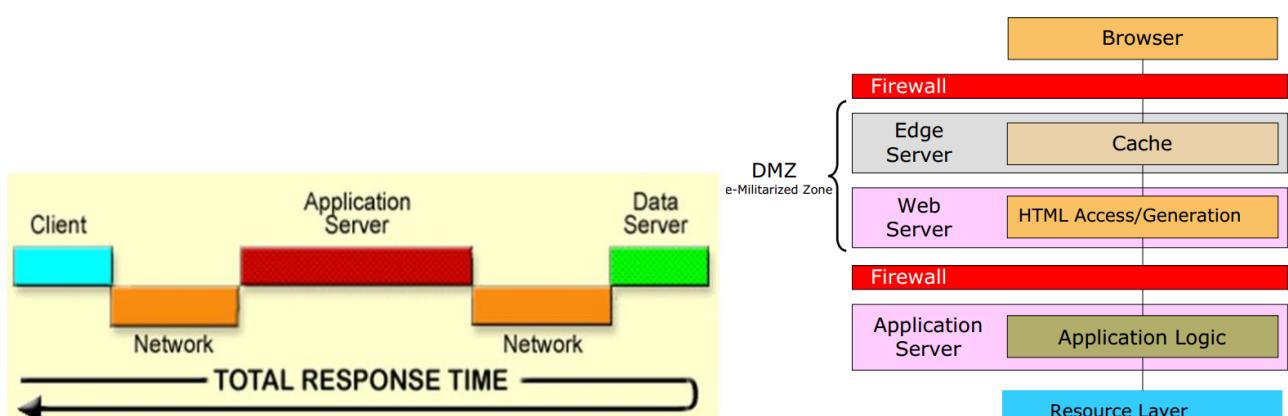
Rechtzeitigkeit der Reaktion und Vorhersagbarkeit sind die beiden Hauptziele. "Schneller" ist nicht immer genug, wie zum Beispiel ein Echtzeit-System eine äußerst gleichbleibende Leistung erfordert.

Performance Mythen

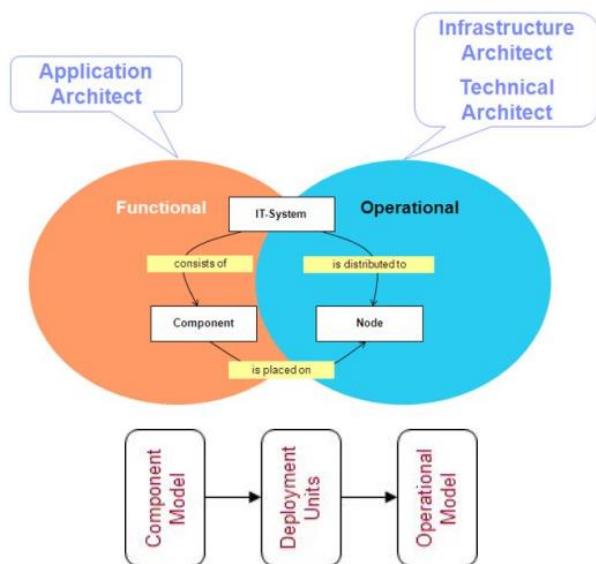
- CPU ist billig
 - o Aufsummierung der Ineffizienz (auf der Serverseite)
- Concurrency verbessert immer die Performance
 - o Ist nicht so. Kommunikation und Synchronisation haben auch Kosten
- Caching verbessert die Performance
 - o «In memory caching speeds up applications until it slows them down.»

Die drei Hauptaspekte der Performance**End-zu-End Antwort Zeit**

Ist die Summe der Antwortzeit aller Komponenten.



Die Ansprechzeiten verschlechtern bei hoher Auslastung.

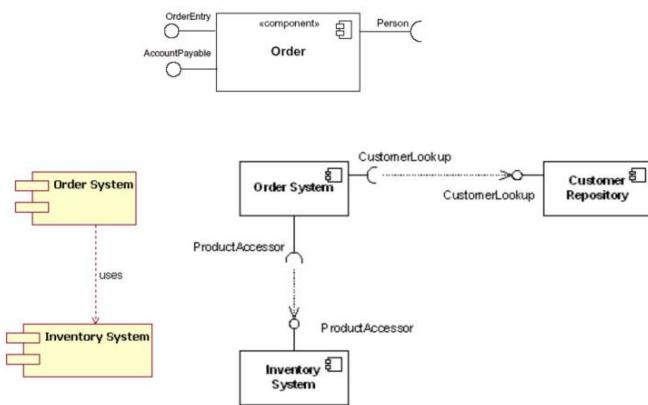


Deployment Units = Eine Kollection von Komponenten, welche auf Nodes plaziert sind.

UML Komponenten Modelle

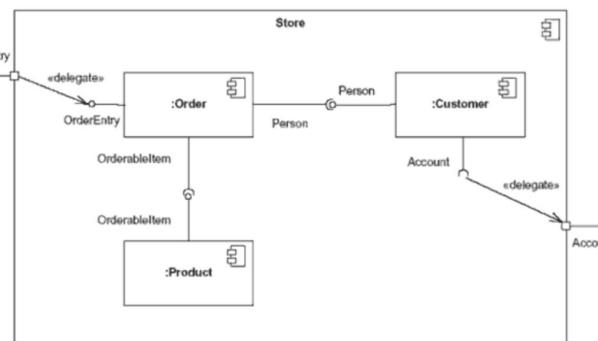
Componenten Relationships

Server provided, Services required (Dependencies)



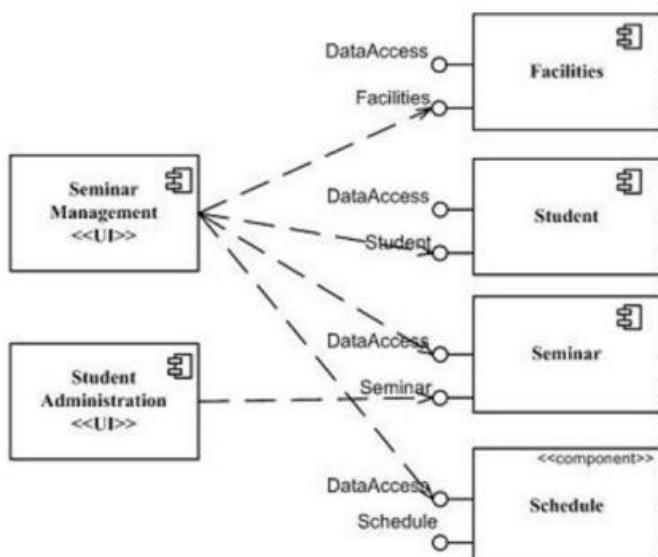
Hierarchies

"contains" – Relationship between components



Schließen der Lücke zwischen den Anforderungen (das "Was") und der Lösung (das "Wie")

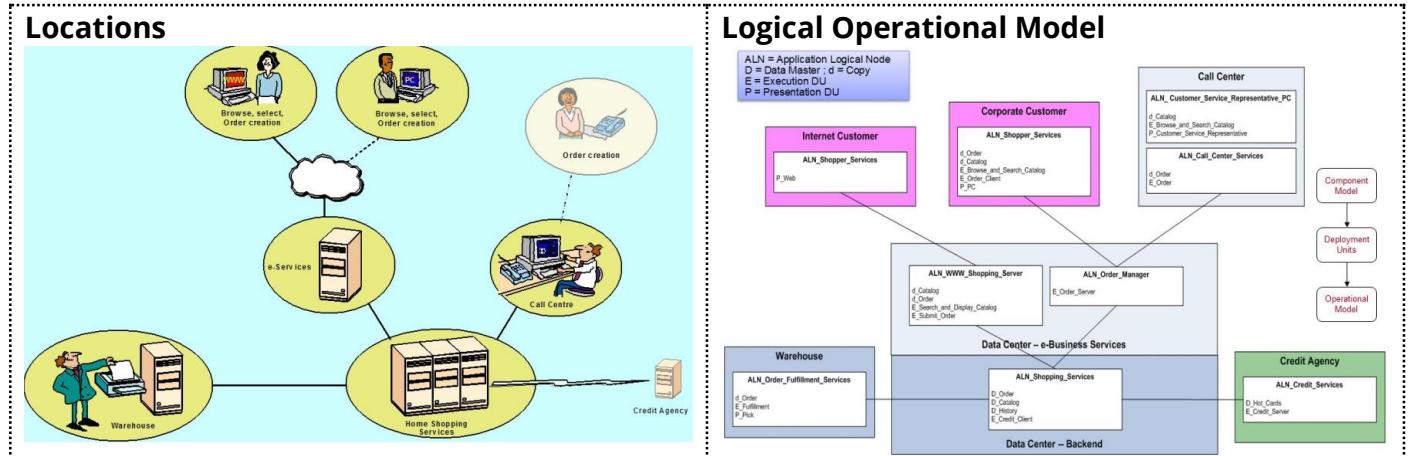
Leitfaden und stellen Sie den Kontext für das Design ein, damit er die funktionalen und nicht funktionalen Anforderungen erfüllt (z. B. Erweiterbarkeit oder Wartbarkeit, etc.). Es visualisiert das System und hilft es zu begreifen.



Verteilte Software-Systeme

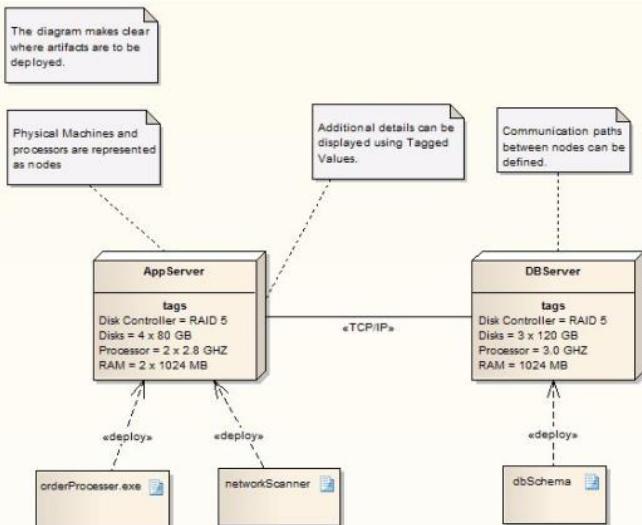
Operational Model

Es repräsentiert die System Infrastruktur Architektur. Die geografische Struktur der Standorte und deren Grenzen, über die das IT-System eingesetzt und betrieben wird. Die Platzierung der Knoten des Systems in diese Orte. Die Verbindungen zwischen den Knoten. Die Organisation der Systemelemente in Zonen. Sowie Sizing und andere Hardware-Spezifikationen für alle Computer, Speichergeräte und Netzwerk-Technologien.



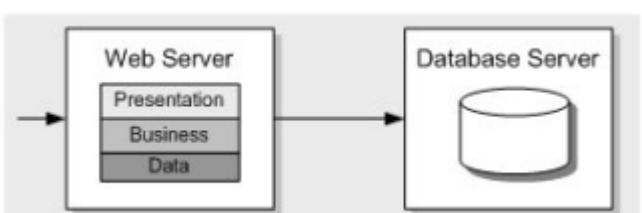
Deployment Units = Klebstoff zwischen logischer Sicht (Komponentenmodell) und Infrastruktursicht (Einsatz / Betriebsmodell). Denn das Component Modeling und das Operational Modelling beeinflussen sich gegenseitig.

UML Deployment Diagramm

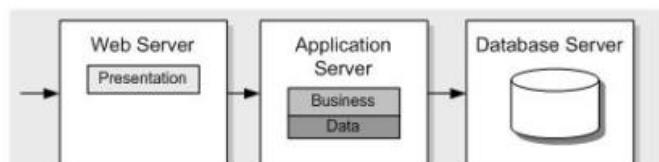


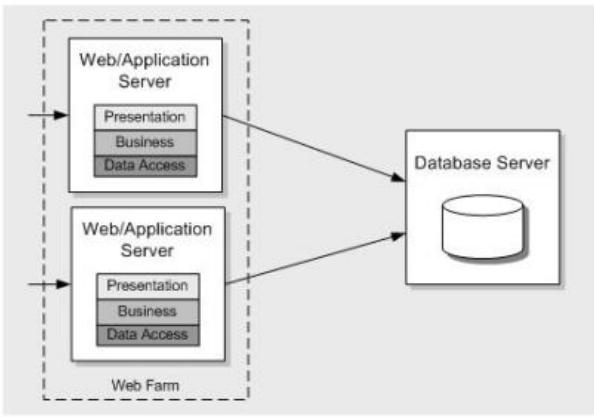
MSDN Deployment patterns

Nicht-verteiltes Deployment

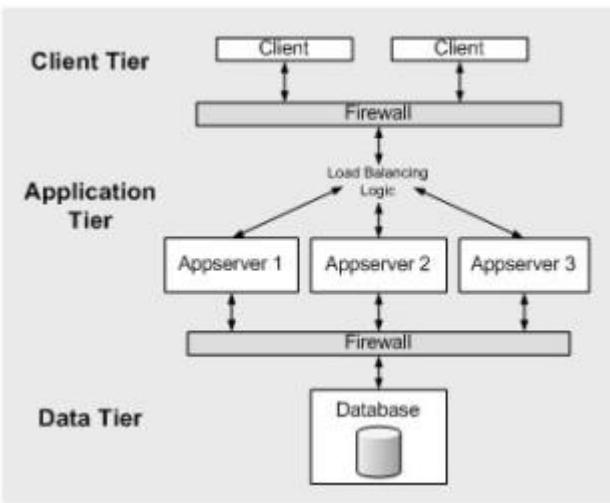


Verteiltes Deployment

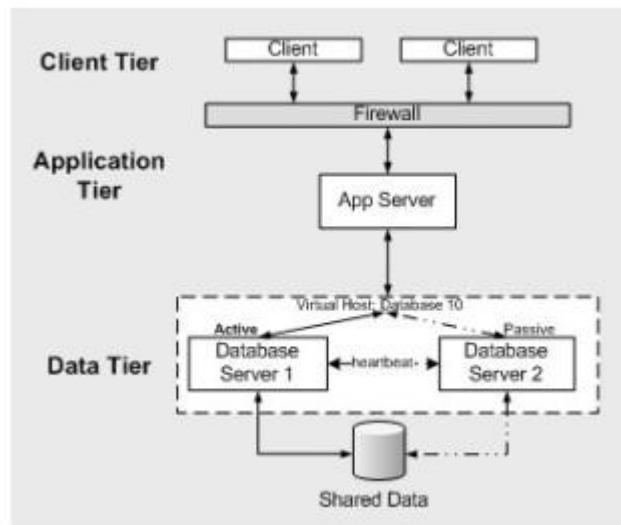




Load-balancing cluster



Failover Cluster



Design for Scalability and Availability

Scalability NFR

Scalability

Die Skalierbarkeit ist die Fähigkeit des Aushaltens von sich erhöhenden Workloads, ohne die SLA zu brechen. Natürlich unter der Voraussetzung, dass die Ressourcen auch erhöht werden.

Linear scalability

Dies ist der meiste Fall. Der Mögliche Workload ist abhängig von den zur Verfügung stehenden Ressourcen.

Dafür gibt es drei verschiedene Wege. Entweder arbeitet man härter / schnell (durch Erhöhung des Speeds), man arbeitet smarter (Besser Algorithmen) oder man holt sich «Hilfe» indem man Parallelität einführt.

Scalability Types

Generation Scalability (Technology Evolution)

Bessere, schnellere, grossere Komponenten, welche durch neue Technologien ermöglicht werden.

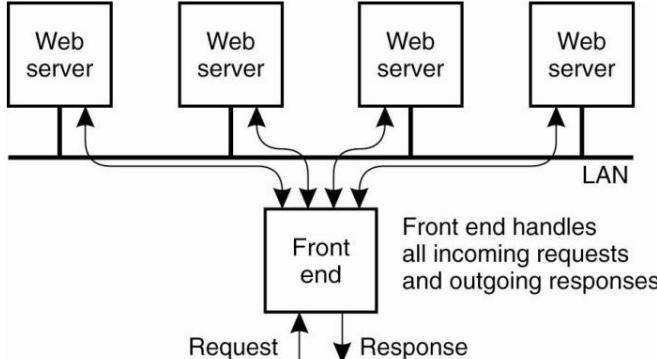
Vertical Scalability (Scale Up)

Das Erhöhen der Performance der individuellen Servers (CPU, Storage, bandwidth).

Horizontal Scalability (Scale Out)

Das Erhöhen der Server bzw. Links.

Horizontal Scaling (Scale Out)



Load Balancer

Ein Load Balancer wird zwischen dem Internet und dem Unternehmens Backend-Server installiert.

Er fungiert als der einzige Point-of-Presence-Knoten des Unternehmens im Internet, auch wenn das Unternehmen mehrere Back-End-Server aufgrund der hohen Nachfrage oder einer großen Menge an Inhalt verwendet. Zudem fängt Datenanforderungen von Clients ab und leitet jede Anfrage an den Server weiter, der derzeit die Anfrage füllen kann.

HA (High availability)

Kann durch die Installation eines Backups Load Balancer erreicht werden, um zu übernehmen, wenn die primäre vorübergehend ausfällt. Die Verfügbarkeit erfolgt durch Parallelität, Lastverteilung und Failover-Unterstützung.

Load Balancer verbessert die Verfügbarkeit und Skalierbarkeit einer Website durch transparente Clustering von Content-Servern. Wenn ein Server ausfällt, wird dieser nicht belastet.

Edge Servers – IP Spraying

Ein Edge Server (Load Balancer) hat eine eigene virtuelle IP Adresse. Die Applikationsserver haben alle ihre eigenen physikalischen IP Adressen.

IP Spraying

Verteilt die Last auf Server in einem Cluster (Satz von Servern, die dieselbe Anwendung ausführen, und können den Clients denselben Inhalt zur Verfügung stellen).

Load Balancer sendet Anfragen für virtuelle IP-Adressen an eine oder mehrere oder die physikalischen IP-Adressen. (Nach Round Robin, Least-Recently-Used, Workload-Based).

Beispiele

- HAProxy
- Nginx
- WebSphere Application Server Edge Componentes

Session Affinity / Sticky Sessions

Session Affinität überschreibt den Load-Balancing-Algorithmus, indem er alle Anfragen in einer Session an einen bestimmten Applikationsserver weiterleitet. Session Affinität verwendet Cookies, um Sitzungsinformationen zu verfolgen und potenziell Anmeldeinformationen zu pflegen. Session-Affinität füllt diese Cookies mit einer Session-ID, die die folgenden Informationen enthält:

- Ein **Bezeichner** für die Wiederherstellung von Sitzungsdaten
- **Routing-Informationen**, um sicherzustellen, dass alle Anfragen in dieser Sitzung immer auf denselben Anwendungsserver weitergeleitet werden

Availability NFR

Ein System gilt als «available», wenn es läuft und richtige Ergebnisse liefert und ndere NFRs zu treffen, zum Beispiel Reaktionszeit. Die Verfügbarkeit eines Systems ist der Bruchteil der Zeit, die es Verfügbar ist. Je mehr Fehler das ein System hat, desto weniger ist es verfügbar. Gleichzeitig gilt, je länger es dauert ein System zu reparieren nach einem Fehler, je weniger verfügbar ist es.

Availability – Gründe für Fehler

Aspekt 1 – Architektur und Umwelt

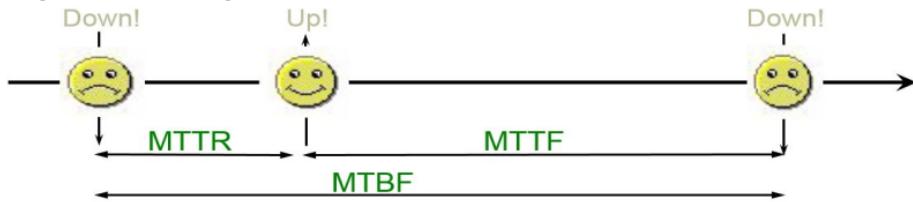
- Kommunikationssysteme
 - o Mehr Zahlen für unterbruchsfreiere Leitungen
 - o Mehr als eine Leitung im Einsatz haben (am besten von verschiedenen Anbietern)
- Power
 - o Battery Backup, damit der Main Memory noch erhalten bleibt
 - o UPS
- Air Conditioning
- Höhere Gewalt
 - o Bunker, Backup in einer anderen Region

Aspekt 2 - System Management

Auch der beste Operator wird das System irgendwann zum Fallen bringen. Daher lassen sich dort Vorkehrung treffen. Zum einen die Automatisierung, redundante Maintenance Procedures, sodass der Operator mindestens zwei Sachen falsch machen muss, weiteres Training und nicht zuletzt Konfigurationstools, welche das Ausmass einer Änderung aufzeigen. Bei Software Installation sind dies Prozeduren, welche keinen Neustart benötigen.

Aspekt 3 - Transiente Softwarefehler

Key Availability Terms



Mean Time to Recover (MTTR)

- Average time to recover (includes repair) a component, sub-system or a system.

Mean Time to Failure (MTTF)

- Average time between successive failures of a given component, sub-system or system.

Mean Time between Failure (MTBF)

- Average time between successive failures of a given component, sub-system or system

Recovery Time Objective (RTO)

- Time within which the service **must** be restored

Recovery Point Objective (RPO)

- Maximum tolerable period in which data might be lost after a failure

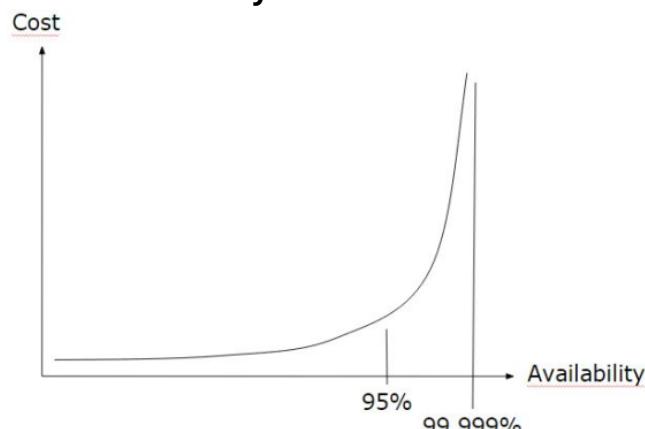
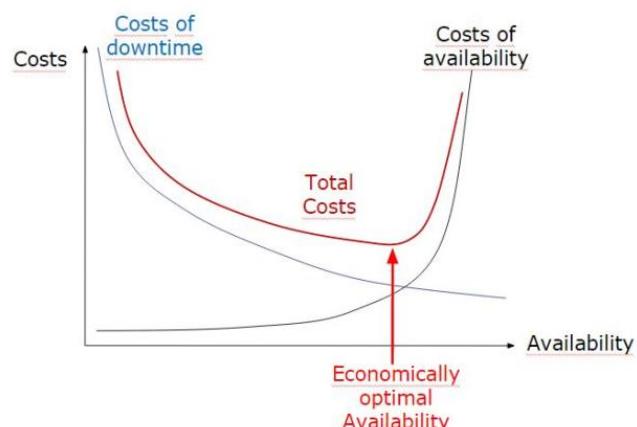
} Qualities

typically specified as NFR (mean / max)

Was heist 99.99x Availability?

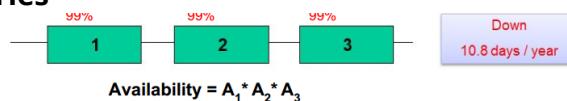
24x7 Verfügbarkeit ist nahe zu unmöglich. Dies geht sehr in die Kosten.

Availability %	Downtime per year	Downtime per month*	Downtime per week
90% ("one nine")	36.5 days	72 hours	16.8 hours
95%	18.25 days	36 hours	8.4 hours
97%	10.96 days	21.6 hours	5.04 hours
98%	7.30 days	14.4 hours	3.36 hours
99% ("two nines")	3.65 days	7.20 hours	1.68 hours
99.5%	1.83 days	3.60 hours	50.4 minutes
99.8%	17.52 hours	86.23 minutes	20.16 minutes
99.9% ("three nines")	8.76 hours	43.8 minutes	10.1 minutes
99.95%	4.38 hours	21.56 minutes	5.04 minutes
99.99% ("four nines")	52.56 minutes	4.32 minutes	1.01 minutes
99.999% ("five nines")	5.26 minutes	25.9 seconds	6.05 seconds
99.9999% ("six nines")	31.5 seconds	2.59 seconds	0.605 seconds
99.99999% ("seven nines")	3.15 seconds	0.259 seconds	0.0605 seconds

Cost of Availability**Cost of downtime vs. cost of availability optimization****Cost of Downtime in Several Domains**

Branch	Cost of Downtime / Hour
Manufacturing	28.000
Logistics	90.000
Retail	90.000
Home Shopping	113.000
Media („pay per view“)	1.100.000
Bank (back office)	2.500.000
Credit Card Processing	2.600.000
Stock Brokerage	6.500.000

Source: Contingency Research / Computer Zeitung 16/2006

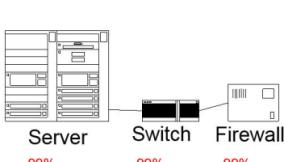
Using components**In Series**

- Components connected form a chain
- Each component relies on the previous component for availability
- The total availability is always lower than the availability of the weakest link

Functional Component Model



Operational Component Model

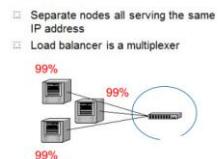
**In Parallel**

- Component redundancy through duplication
- Total availability is higher than the availability of the individual links

Component Model

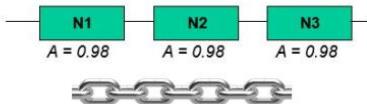


Operational Model

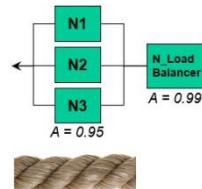




What is the overall availability of this serial structure of nodes?



What is the overall availability of this parallel-serial structure of nodes?



Techniques to improve the availability of a (distributed) system

Avoid Single Points of Failure

Redundancy

Schlagwörter hier sind Clusters, Warm Backups und Hot Backups.

Warm Backup

Wenn der Fehler erkannt wird, wird der Backup-Server sofort zum primären Server und erholt sich nach dem letzten nicht wiederherstellbaren Betrieb des ehemaligen Primärs in den Zustand. MTTR wird reduziert, da die Zeit für die Erstellung des Prozesses für den Backup-Server nicht benötigt wird.

Hot Backup

Die Kopplung zwischen Primär- und Backup ist sehr dicht. Die beiden Server laufen immer synchron, sodass der Backup sofort übernehmen kann. Diese Lösung ist sehr kostenintensiv, da eigentlich jede Komponente doppelt ausgelegt werden muss.

Design Option	Installed Nodes (N configurable)	Running Nodes (Sunny Day)	Active Nodes (Sunny Day)
Cold Standby	N	1	1
Warm Standby	N	N	1
Hot Standby	N	N	N

Detect failures as fast as possible

Die Zeit vom Versagen, den Fehler zu detektieren, trägt zu MTTR bei, d.h. verringert die Verfügbarkeit!
Das Prinzip Fail Fast kann hier eingesetzt werden.

VSS-Management

Wieso und Was?

System Management
Software distribution and upgrading
Version control
Virus protection
User profile management
Backup and recovery
Printer spooling
Job scheduling
Capacity planning
Performance monitoring / Logging
Network management

Configuration Management
Document all components of a system used to build executable programs

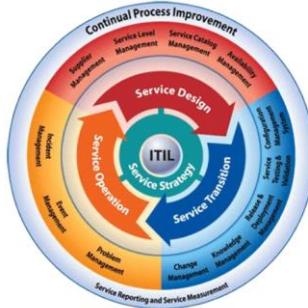
Recreate each build as well as earlier environments in order to maintain previous versions of a product

Prevent unauthorized access to files or to alert the appropriate users when a file has been altered

IT Infrastructure Library (ITIL) → Service Management Framework

IT Infrastructure Library (ITIL): Service Management Framework

- Principles and practices for daily lives of IT operations staff
 - Terminology
 - Functions (organization)
 - Processes
 - Governance
 - Operational
 - Certification
- Service: IT operations
 - e.g. User help desk
 - e.g. Software updates
 - e.g. Backup and recovery



ITIL Edition 2011 Components

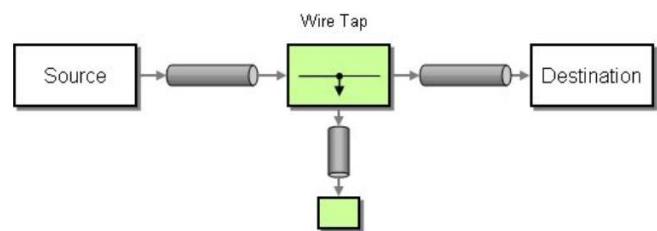


System Management Patterns

Systems Management Patterns zur Überwachung und Steuerung von Message- / Queuing-basierten verteilten Softwaresystemen.

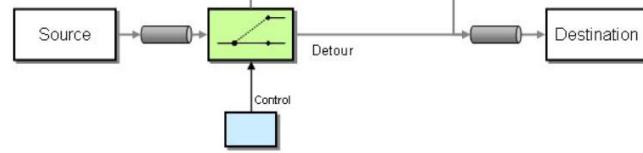
Wire Tap

Überprüfen Sie Nachrichten, die auf einem Punkt-zu-Punkt-Kanal fahren. Verbrauchen Sie Nachrichten aus dem Eingangskanal und veröffentlichen die Nachrichten, die nicht am Zielkanal modifiziert sind auf einem separaten Prüfkanal. Da die Inspektionslogik in einer zweiten Komponente ist, ist der Wire Tap generisch und wiederverwendbar

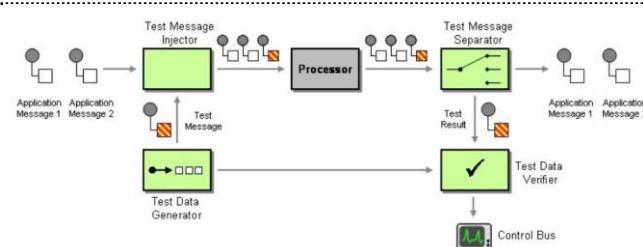


Detour

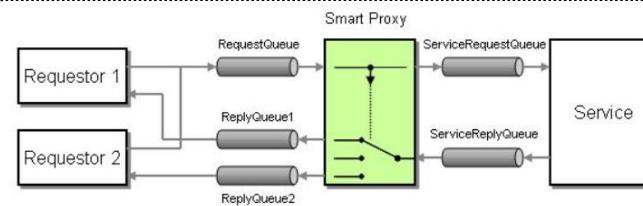
Vermitteln Sie eine Nachricht durch Zwischenschritte, um Validierung, Test oder Debugging durchzuführen. Kontextbasierter Router: In einem Zustand leitet der Router eingehende Nachrichten durch zusätzliche Schritte, während er im anderen die Nachrichten direkt an den Zielkanal leitet. Komponenten im Umweg können die Nachricht überprüfen und / oder ändern. Letztendlich führen diese Komponenten die Nachricht an das ursprüngliche Ziel.

**Test Message**

Sicherstellung der Gesundheit einer Nachrichtenverarbeitungskomponenten. Identifizieren von Nachrichtenverarbeitungskomponenten, die aktiv Nachrichten verarbeiten, aber ausgehende Nachrichten aufgrund eines internen Fehlers verwerfen.

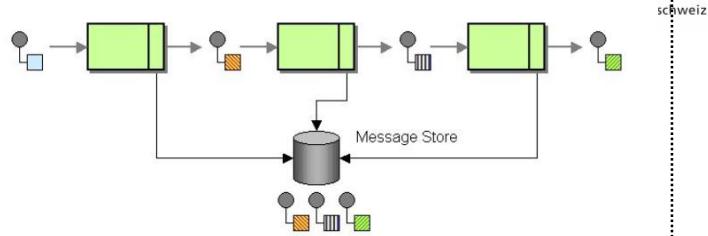
**Smart Proxy**

Verfolgen / Überprüfen von Nachrichten an / von einem Dienst, der Antwortnachrichten an die vom Anforderer angegebene Rücksendeadresse sendet (d.h. die Rücksendeadresse ist in der Anforderungsnachricht enthalten, das einfache Wire-Tap-Muster funktioniert nicht). Smart Proxy fängt Nachrichten ab, die an den Service gesendet werden. Smart Proxy speichert die vom ursprünglichen Absender angegebene Rücksendeadresse. SmartProxy ersetzt die Rücksendeadresse in der Nachricht mit eigener Adresse. Wenn eine Antwortnachricht eingeht, ruft der Smart Proxy die gespeicherte Rücksendeadresse ab und leitet die unveränderte Antwortadresse an den ursprünglichen Anforderer weiter.



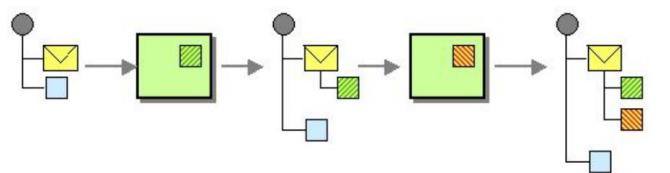
Message Store

Erfassen Sie Informationen über jede Nachricht an zentraler Stelle. Jede Komponente sendet Duplikate an den zentralen Message Store. Um die resultierende Netzwerkverkehrszunahme zu minimieren, können Sie nur Schlüsselfelder duplizieren, die für Systemverwaltungs- und Protokollierungszwecke erforderlich sind, z. B. Nachrichten-ID, Zeitstempel usw.



Message History

Erstellen Sie eine Liste aller Anwendungen, die die Nachricht seit ihrer Entstehung durchlaufen hat, um bei der Analyse und dem Debugging des Nachrichtenflusses zu helfen. Jede Komponente, die die Nachricht verarbeitet (einschließlich des Originators), fügt einen Eintrag zur Liste hinzu. Die Nachrichtenhistorie wird normalerweise im Nachrichtenkopf gespeichert, so dass sie von dem Nachrichtentext getrennt ist, der anwendungsspezifische Daten enthält.

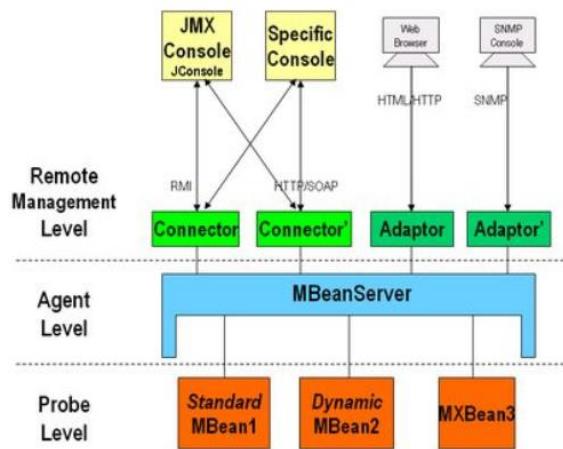


Channel Purger

Eliminieren Sie 'left-over' Nachrichten (von vorherigen, möglicherweise fehlgeschlagenen Tests), so dass sie nicht neue Tests oder das laufende System stören. Entfernen einer einzelnen Nachricht oder eines Satzes von Nachrichten, die auf bestimmten Kriterien basieren, z. B. Nachrichten-ID oder die Werte bestimmter Nachrichtenfelder (insbesondere erforderlich, um Probleme bei der Ausführung von Produktionssystemen zu beheben)



Java Management Extensions (JMX)



Verwalten und Überwachen von Anwendungen, Systemobjekten und Geräten (z. B. Drucken).

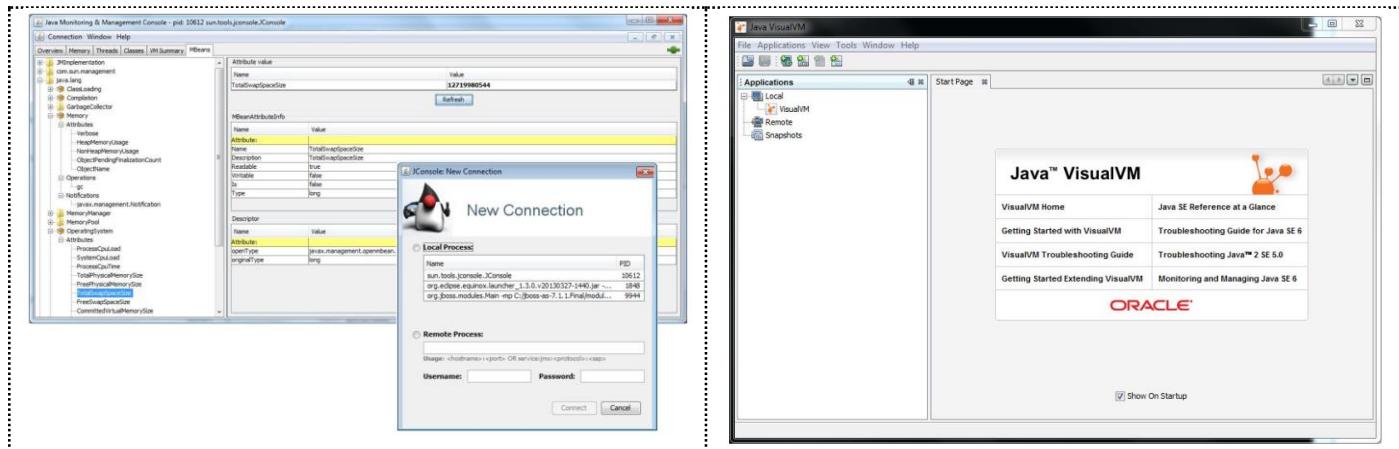
Drei Level Architektur

- **Probe Level** (Instrumentierungsebene): enthält die Sonden (MBeans), die die Ressourcen stützen
- **Agent-Level** (MBeanServer): als Vermittler zwischen den MBeans und den Anwendungen
- **Remote Management Level**: ermöglicht es Anwendungen, über Connectors und Adapter auf den MBeanServer zuzugreifen.

Managed Beans (MBeans)

Implementiert eine Business-Schnittstelle mit Setter und Getter für die Attribute und die Operationen (d.h. Methoden). Stellt eine Ressource dar, die in der virtuellen Java-Maschine ausgeführt wird und über JMX verwaltet und überwacht werden kann. Kann zum Sammeln von Statistiken über Leistung, Ressourcenverbrauch oder Probleme (Pull) verwendet werden. Kann zudem zum Erhalten und Setzen von Anwendungskonfigurationen oder Eigenschaften verwendet werden (Push / Pull). Des Weiteren kann es auch für die Benachrichtigung von Ereignissen wie Fehlern oder Zustandsänderungen (Push) verwendet werden.

JConsole – JMX Client



Logging

Die Herausforderungen beim verteilten Logging

Das Logging in einer verteilten Umgebung stellt viele weitere Probleme dar:

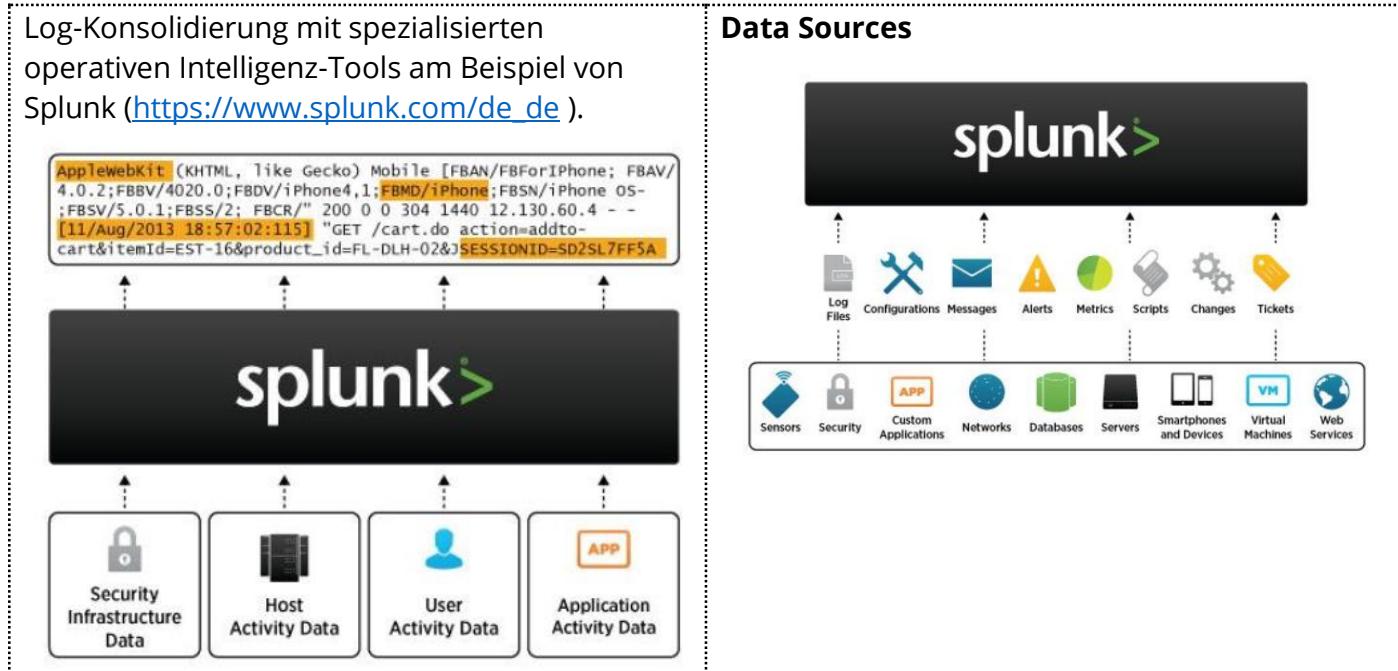
- Die Logging-Lösung sollte für einen Entwickler einfach zu bedienen, über verschiedene Anwendungen wiederverwendbar und erweiterbar auf neue Bedürfnisse zugeschnitten sein.
- Die Protokollierung sollte zu einem einzigen Punkt zentralisiert werden, damit ein Administrator von einem Ort aus darauf zugreifen kann.
- Die Protokollierung sollte wenig Einfluss auf die Anwendungsleistung haben.
- Die Logging-Lösung sollte die Meldungen über die Konfiguration nach Art filtern, um bestimmte Ereignisse auszulösen. Dazu gehören Dinge wie die Aufrechterhaltung der Nachricht und die Durchführung von Alarm Benachrichtigung, wie Paging an einen Administrator.
- Die Protokollierungsumgebung sollte eine zentrale und lokale Protokollierung ermöglichen, indem sie verschiedene Typen identifiziert.

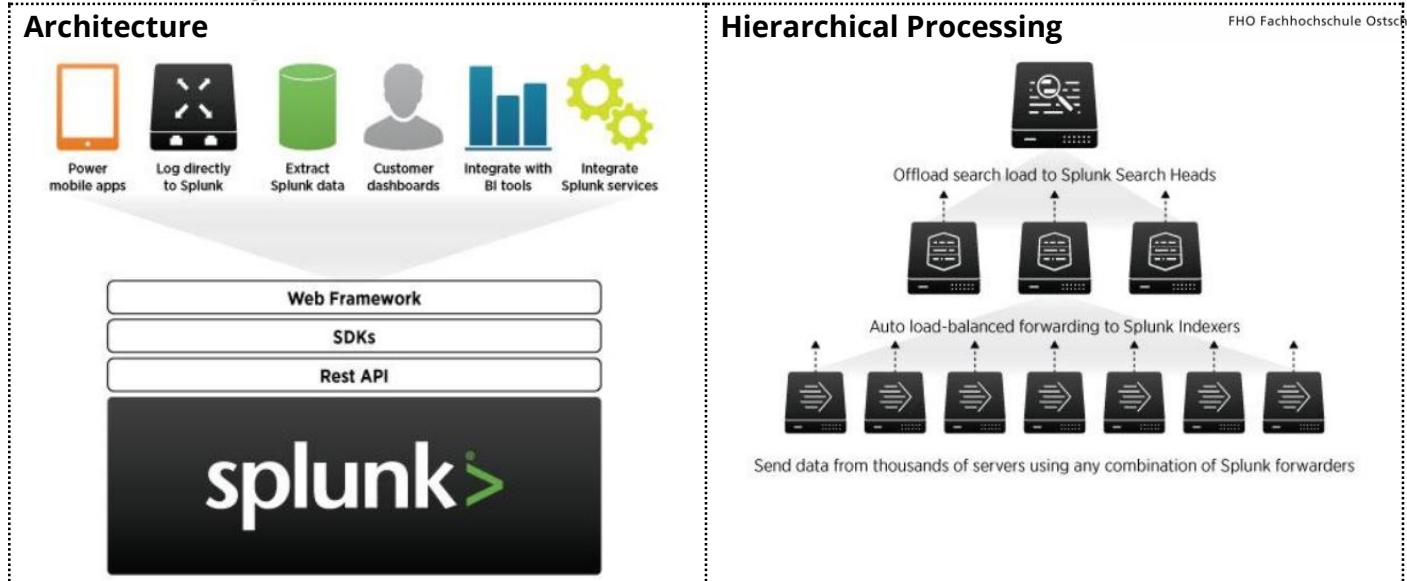
Logging Szenarios und Log Formate

Log-Inhalt sollte Entwickler und Betreiber während der Unterstützung unterstützen bei...

- **Ereignis-Korrelation** über verteilte Knoten
- **Ursachenanalysen** (auf dem Weg von Symptomen zu Korrekturmaßnahmen)
- **Event-Stürmen** (hohe Anzahl von Ereignissen, z. B. während der Peak-Workload oder wenn ein Denial-of-Service-Angriff durchgeführt wird)

Splunk





Tips und Tricks

- Nutzen Sie populäre Logging Frameworks und bauen Sie keine Eigenen (Log4j, Logback)
- Definieren und Verwenden Sie Protokollebenen und die anderen Tags
- Verwenden Sie die Analogie / Metapher eines Notrufs bei der Gestaltung von Log-Formaten

Component Failure Impact Analysis (CFIA)

Das Problem

Ein einziger Komponentenfehler hat dazu geführt, dass das ganze System abgestürzt ist. Zum Reparieren einer einzigen Komponente muss dann das ganze System heruntergefahren werden. Weitere Probleme können bei der System und Netzwerkdokumentation vorhanden sein, weil sie nicht mehr up-to-date oder dafür gar kein Manual vorhanden ist. Dies führt dazu, dass nur eine spezifische Person das ganze System wiederherstellen kann.

Component Failure Impact Analysis (CFIA) ist eine proaktive Verfügbarkeits Management Methode. Sie dient zur Bereitstellung von IT mit der Geschäfts- und Anwenderperspektive und gibt Auskunft darüber, wie sich Defizite in der Infrastruktur und dem Underpinning-Prozess, Verfahren und Service Delivery Fähigkeiten auf den Geschäftsbetrieb auswirken. Anders gesagt eine Methode zur Optimierung der Leistungsfähigkeit der IT-Infrastruktur, der Services und der unterstützenden Organisation, um eine kostengünstige und nachhaltige Verfügbarkeit zu ermöglichen, die es dem Unternehmen ermöglicht, ihre Ziele zu erreichen.

Der Zweck von CFIA und wie die Resultate genutzt werden können

Der Zweck eines CFIA ist es, Informationen zur Verfügung zu stellen, um sicherzustellen, dass die Kriterien für die Verfügbarkeit und die Wiederherstellung von neuen oder bestehenden IT-Services beeinflusst werden, um die Auswirkungen des Ausfalls des Geschäftsbetriebes und der Nutzer zu verhindern oder zu minimieren.

CFIA kann verwendet werden, um die Auswirkungen auf den IT-Service, die sich aus Komponentenfehlern innerhalb der Technologie ergeben, vorherzusagen und zu bewerten. Die Resultate von einem CFIA kann verwendet werden, um zu ermitteln, wo zusätzliche Resilienz betrachtet werden sollte, um die Auswirkung des Komponentenversagens auf den Geschäftsbetrieb und die Benutzer zu verhindern oder zu minimieren, und wo Verbesserungen an Prozeduren oder Prozessen und Fähigkeiten vorgenommen werden sollten. Dies ist besonders wichtig während der Entwurfsphase, wo es notwendig ist, die Auswirkungen auf die IT-Service-Fähigkeit, die sich aus Komponentenfehlern im Rahmen des vorgeschlagenen IT Service Design ergeben, vorherzusagen und zu bewerten.

Hauptziele

- Identifizieren Sie kritische Verfügbarkeiten und Leistungsrisiken für einige oder alle geschäftskritischen Anwendungen.
- Identifizieren Sie Maßnahmen, die getroffen werden können, um die Wahrscheinlichkeit von Ausfällen zu reduzieren.
- Identifizieren Sie Maßnahmen, die getroffen werden können, um die Zeit zu reduzieren, die es braucht, um diesen Fehler zu erkennen und zu reparieren.

Typisches Resultat

Eine auf Phasen aufgeteilte Lösungsroadmap, welche die technischen, die organisatorischen und die prozessnahen Aspekte behandelt.

Phase 1

Verbesserungen, die sofort bzw. innerhalb von wenigen Wochen erreicht werden können und müssen.

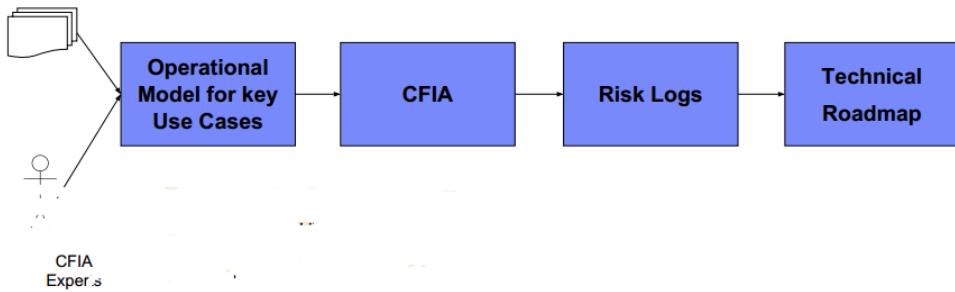
Phase 2

Hochprioritäre Verbesserungen, die mittelfristig, z. B. innerhalb weniger Monate, erreicht werden müssen und können.

Phase 3

Tiefprioritäre Verbesserungen und Verbesserung, welche eine grundlegend architektonische Veränderung erfordern.

Existing Architecture Documentation

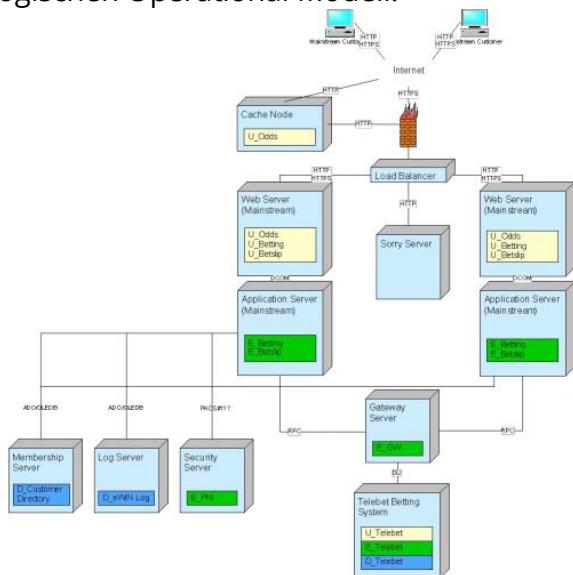


Operational Model

1. Knoten / HW-Plattformen mit den zugeordneten Deployment Units (Funktionale Komponenten)
2. Location Model (Zuordnung von Knoten / Plattformen zu Orten, Gebäuden, Räumen / Zonen innerhalb der Gebäude)
3. Verbindungen / Netzwerk zwischen den Knoten
 - a. WAN (Wide Area Network) → Internet, Private Networks
 - b. LAN (Local Area Network, innerhalb Data Center, Verbindung zwischen Servern und Storage-Systemen)

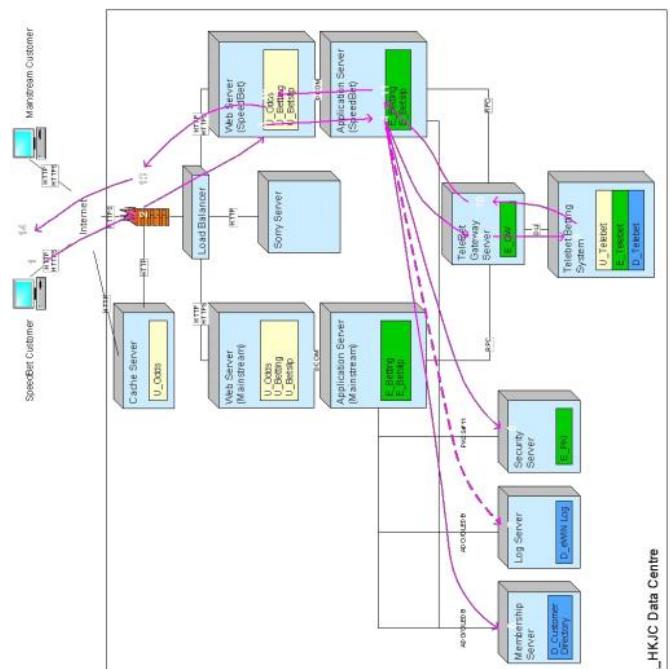
Beispiel

Gestartet wird mit einem statischen Anblick des logischen Operational Modell.



Gehen dann alle «Architecturally Significant» Use Cases durch und identifiziere alle Verbindungen, Nodes und Komponenten welche involviert sind.

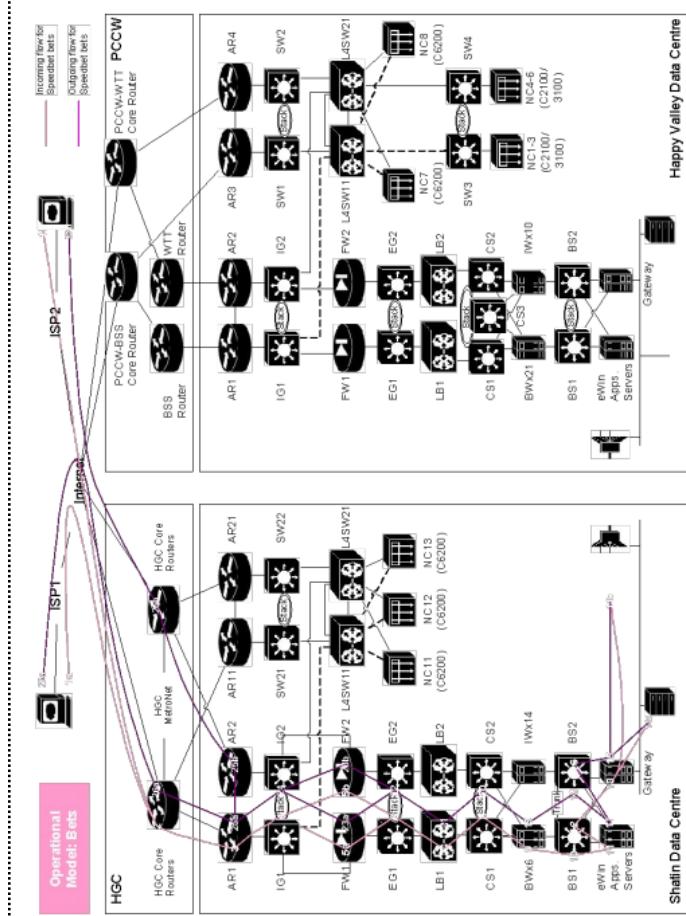
“Architecturally significant use cases” are the minimum set of use cases needed to exercise all the components in the system



Verteilte Software-Systeme

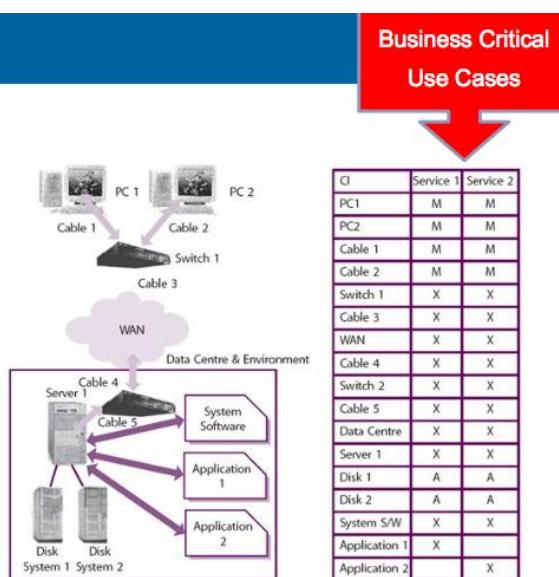
Die Ausarbeitung auf die physikalische Ebene identifiziert alle Komponenten (Software sowie Hardware) und deren Ausfallmodi für die CFIA.

Beispiele für Symbole, die in einem physikalischen Operationsmodell verwendet werden.



Basics CFIA Matrix

Erstellen Sie ein Raster mit Konfigurationselementen (CIs, die Bestandteile, Knoten, Links etc. sein können, je nach den Umständen) auf einer Achse und die IT-Services, die eine Abhängigkeit vom CI haben auf der anderen Achse.



Die Tabelle sollte wie folgt ausgeführt werden.

- Leer lassen, wenn ein Ausfall des CI den Dienst in keiner Weise beeinträchtigt
- Schreiben Sie ein 'X', wenn der Ausfall des CI den IT-Dienst inaktiv macht
- Setzen Sie ein 'A' ein, wenn es einen alternativen CI gibt, um den Service bereitzustellen
- Setzen Sie ein 'M' ein, wenn es einen alternativen CI gibt, aber der Service erfordert manuelle Eingriffe, die wiederhergestellt werden sollen.

Example of Configurations Items (CIs)

Ein CFIA kann in verschiedenen Graden der Granularität durchgeführt werden.

- Hardwarekomponenten wie Server, Netzwerkkarten, SAN Switches, Netzwerk Switches oder Routers
- Betriebssystemkomponenten wie das OS selber, oder wichtige Services wie TCP/IP
- Middleware Software Komponenten wie Datenbanken, Application Servers, Message Queues oder Workflow Managers.
- Applikationssoftware wie SAP....

Gemäss ITIL

Configuration Item (CI) (ITILv3): Any Component that needs to be managed in order to deliver an IT Service. Information about each CI is recorded in a Configuration Record within the Configuration Management System and is maintained throughout its Lifecycle by **Configuration Management**. CIs are under the control of **Change Management**. CIs typically include IT Services, hardware, software, buildings, people and formal documentation such as Process documentation and SLAs.

How to read the CFIA matrix

CIs, die eine große Anzahl von Xs haben, sind für viele Dienste kritisch und können zu einer hohen Auswirkung führen, falls die Komponente ausfällt.

IT-Dienste mit hohen Zählungen von Xs sind komplex und sind anfällig für Misserfolge.

Erweitern Sie dann die CFIA Matrix mit Failure Modes, Failure Effects und Recovery Prozeduren

Für jede Komponente beschreiben Sie:

- Was ist es?
- Wie kann es scheitern?
- Wie kann ein Komponentenfehler detektiert werden (Wie schnell?...)
- Was sind die Effekte wenn diese Komponente ausfällt?
- Was sind die Wiederherstellungs oder Failoverprozeduren?

Logical Node Component	Physical Component Name	Location	Zone	Component Description	Failure Mode	Failure Effects	What is the Recovery/Failover Procedure ?
MTN Active Reverse Proxy	dmzweb1	Newlands	DMZ	MTN Active landing page - active.mtn.co.za, HPBlade BL20p	Hardware failure IIS failure Performance degradation	All services in MTN Active become unavailable All services in MTN Active become unavailable Response times become degraded or MTN Active becomes unusable	No failover Recovery at DR site Manual problem diagnosis
MTN Active Servers	nidwlm01	Newlands	DC	Windows Server 2003, Enterprise Edition 5.2, SP1	Hardware failure	Uncertain - concerns over whether Weblogic 8.1.4 supports will reroute work if one of the servers fails	Restart Weblogic
	nidaIm02	Newlands	DC	Windows Server 2003, Enterprise Edition 5.2, SP1			

For all the components, he CFIA describes key impacts in response to different types of failures ...

... and what the Recovery / Failure process is.

- **Node Description**

- Node Name / Node ID
- Primary Function of the Node
- Business impact if node not available
- Node Owner
- Service Manager
- Technical Lead
- Failover Node(s) and how does it failover
- Failover documents available: Y/N
- Quality of the failover documents: Strong / weak
- Date failover was last tested
- Recovery method for H/W, S/W, and data
- Recovery documents available: Y/N
- Quality of the recovery documents (rate 1-10)
- Data recovery was last tested

- **Clustering**

- Date last verified (e.g., parameter settings)

- **Applications and Data**

- Backup method for failover / recovery: Tape, remote disk copy, sync/async, etc.
- Failover Method
- Failover tested? How? How long did it take?
- Restore tested? How? How long did it take?
- Data Loss **SLA**?
- Application monitoring: How? What is monitored?
- Data monitoring: How? What is monitored?

- **Hardware**

- Hardware model and age in years
- Rack or carcass
- Processor: Single / SMP
- Hardware Monitoring: Y/N; if yes: how
- Heat Management: Y/N; Multiple Fans? Last checked?
- Power Supply: Single / Dual
- Internal Disk: Raid Level
- Network card: Single / Dual

- **Operating System**

- Version and Release
- Supported: Y/N
- Recovery Method: Image copy or re-install
- Operating system recovery procedures, and roles and responsibilities: Strong / Weak
- Patch Management: Strong / Weak
- Privileged User ID Management: Strong / Weak

- **Operations and Admin Processes**

- Strong / weak?
- Experienced Operators
- Experienced System Administrators
- **RCA**s related to this node, or applications running on this node

Risk Log

Dokumentschlüsselbefunde für jeden CI (Knoten, Komponente, Link etc.) zur Unterstützung der Priorisierung von Risiken und Auswahl der Risiken, für die eine Lösung vorgeschlagen wird.

Risiken, die aus der Systemarchitektur vor einer formalen Komplettlösung offensichtlich sind. Risiken, die sich aus der CFIA-Tabelle auf Systemebene ergeben. Risiken, die sich aus der detaillierten "Knoten"-Analyse ergeben. Risiken, die aus früheren Problemsätzen und RCAs entstehen.

- **Resilience risks:** Risks which may cause a service to become unavailable in the first place. They are fundamental weaknesses.
 - Single point of failure in the IT infrastructure
 - No hot failover capability
 - Log files filling up
 - Bugs in code
 - Operator error
 - Old hardware
- **Recovery risks:** Risks which prevent the service from being recovered from an outage in a timely manner.
 - Responsibilities not defined clearly
 - Lack of, or incomplete recovery procedures
 - Lack of skills
 - Over-complex manual tasks with no automation
 - Recovery process is not tested
- **Security Risks:** Risks which can render a service to become useless, for example, through:
 - Security patch management
 - Privileged users who misuse their privileges
 - Denial of service attacks

 RCA = Root Cause Analysis
 (click to learn more)

Resilience Scale Table

Für Resilience Risks, geben Sie ein Risiko-Rating für Auswirkungen, Auftreten und Erkennbarkeit, mit der Rating-Skala in der Resilience Scale Tabelle an.

Resilience Scale Table				
Rating		Business Impact (A node failure would)	Occurrence Time Period	Detect-ability (how easy to detect failure)
H	10	Injure a person.	More than once per day.	Only via a Customer complaint or notice.
	9	Create an illegal situation (e.g regulatory compliance)	Once every 3-4 days	Manual checking of individual nodes infrastructure by layered teams.
	8	Service unusable – performance or outage.	Once per week	Automatic monitoring & alerting on node infrastructure by layered teams.
M	7	Major customer escalation – performance related.	Once per month	Automatic monitoring & alerting on node infrastructure to a central team.
	6	Complaint - major performance degradation.	Once every 3 months	As above plus basic infrastructure correlation.
	5	Complaint –ongoing performance degradation.	Once every 6 months	As above plus basic synthetic transactions to defined points in infrastructure (e.g. data centre).
	4	Visible - minor ongoing loss of performance.	Once per year	As above plus real user transactions added (end to end).
L	3	Visible but can be overcome readily overcome	Once every 1-3 years	As above with extended application logging to assist root cause.
	2	Not visible - minor impact on performance.	Once every 3-6 years	As above with integration to an actively managed CMDB (accuracy).
	1	Not visible– no impact on performance.	Once every 6-100 years	Full business systems impact correlation of above (all nodes, all technical layers).

Assign a value on a 1-10 scale to each of:

- Business Impact: How severe is the failure.
- Occurrence: How likely is the outage to happen.
- Detect-ability: Can the cause be easily detected.

Recovery Scale Table

Für jedes CI (Knoten, Component, Link, etc.), geben Sie eine Risikobewertung für Teaming, Prozedur und Automation mit der Recovery Scale Tabelle an.

Recovery Scale Table			
Rating	Team (The team recovering the service)	Procedure (The procedures used are:)	Automation (The level of automation to assist recovery:)
H	10 Have little experience and no overall leader.	No procedures exist.	There is no automation in any aspect. Long manual recovery.
	9 Inexperienced teams can recover a few critical nodes, limited e-2-e team structure, no RM	Very few procedures exist and are not maintained or tested.	n/a
	8 Inexperienced teams can recover most critical nodes, limited e-2-e team structure, no RM	Some level of procedures exist but they are not maintained or tested	Infrastructure services recovered automatically, long and complex manual recovery of data/apps.
M	7 Inexperienced teams can recover all critical nodes in isolation, limited e-2-e team structure, no RM.	Most nodes have procedures but they are not integrated nor maintained/tested	n/a
	6 Experienced teams can recover all critical nodes, some end to end team structure, no overall RM	All nodes have procedures but they are not integrated or maintained/tested.	Infrastructure services recovered automatically, some well rehearsed manual recovery of applications or data. Intermediate outage.
	5 As below...some e-2-e structure	As above.....Integrated, not maintained/tested.	n/a
	4 Experienced structured team with well defined e-2-e structure inexperienced RM	Integrated system level and node level procedures, Limited maintenance, no testing.	Infra recovered automatically with some application & data recovery automated. Minor outage expected.
L	3 As below....RM has little experience	As below & ...Limited Maintenance	Fully automated recovery with minor service outage.
	2 As below &RM has medium experience...	As below & Limited test	n/a
	1 Experienced and structured team with e-2-e service recovery managed by an experienced Recovery Manager (RM).	Integrated System level and node-level procedures exist and are regularly maintained and fully tested e-2-e	Fully automated recovery and will not be noticed by the Customer (e.g. hot failover).

Assign a value on a 1-10 scale to each of:

- Team (the team recovering the service)
- Procedure (the recovery procedures used)
- Automation (the level of automation to assist recovery)

Cheat Sheet – «Most Popular» Problems

- Identification of Single Points of Failure (SPOF), where loss of a single component would impact the non-functional characteristics of an IT service.
- Application Design, for example singleton processes to read data queues that represent single points of failure, and Application Integration where many systems are tightly coupled together as part of an IT service
- Missing or inadequately documented architecture for the IT service and operational procedures.
- Performance, capacity or scalability concerns for any component, in response to increases in demand.
- Monitoring may be missing or otherwise deficient, resulting a component or service outage not being detected (eg. No monitoring of overall IT service availability).
- Deficiencies in backup and restoration procedures that may impede recovery operations (eg. Data being stored on lots of separate tapes or backup data not being held securely).
- Processes and procedures to recover from a failure (or failover) are missing or deficient. Manual intervention means much more significant delays in recovery.
- Poor change control procedures, where changes are made to an IT service without adequate risk assessment being conducted prior to deployment. This might consider the nature and comprehensiveness of testing (eg. Does it include ,), adequacy of back out procedures and whether changes made in other systems might impact on the IT service being considered as part of this.
- Management of the configuration of hardware, system software, middleware and applications is consistent across development environments (eg. Development, Test, UAT, Pre-production and Production) and across clustered servers in the same environment (eg. Multiple clustered Web servers)
- Deficiencies in Testing procedures or the test environment itself (i.e. the system can't be tested under "production like" conditions)
- Adequacy of backup and restore procedures
- Technical Inadequacies in the solution such as software approach or beyond end of life or use of unsupported software combinations,
- "Key Person" dependencies – where a single individual is responsible for technical support or operations for one or more components essential to the successful operation of the IT service.

Naming

Main Goal

I must be able to find resources that interest me.

Aspects of Naming

Naming: Assign names to entities (Telephone provider assigns phone numbers to customers)

Name Resolution: Access a resource using a name (How can I reach Bob?)

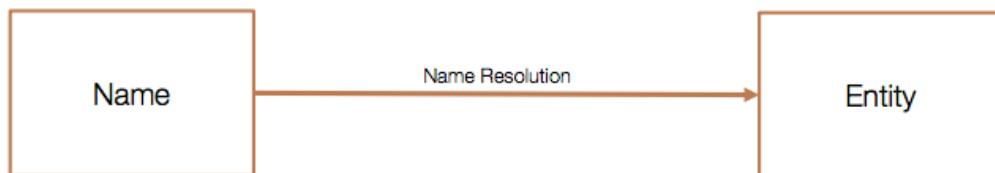
Naming in a Distributed System (e.g. DNS)

Disadvantages of using centralized name resolution in a distributed system

Not robust (Single point of failure), Network latency due to non-locality → Distributed Name Resolution.

Name Resolution

Main Aim:



This is seldom done in one shot.

Q. Why?

"All problems in computer science can be solved by another level of indirection."

David Wheeler



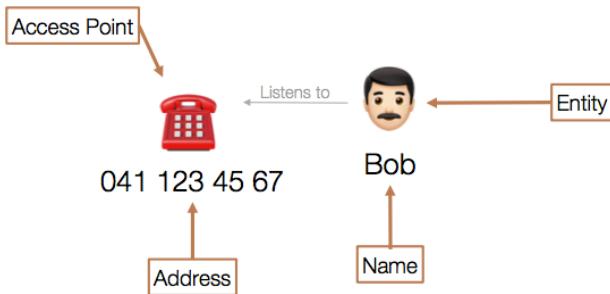
Note: The indirection can have multiple levels.

For example:

DNS Name → IP Address → MAC Address → Network Interface → Host

Terminology

Name	A string of bytes (or characters) that is used to refer to an entity
Entity	Anything that can be operated on
Access Point	Special type of entity that is used to access another entity
Address	Name of an access point

Example Telephone Network

The entity with name "Bob" can be reached over the access point with address 041 123 45 67

From a Name to an Entity**Access Point → Entity**

Often taken care of physically

Address → Access Point

Often taken care of at another level

Name → Address

This is typically the name resolution problem at a given level.

What advantages does this indirection bring?

- Address and access points may change but entity is still accessible through its name
 - o Bob buys a new phone
 - o Bob moves to another country
- Entity can listen to several access points

Other Examples

System	Name	Address	Access Point	Entity
Post	Recipient's name	City + ZIP Code + Street + Number	Mailbox	Recipient (Person)
File System	Path	File Handle	Disk Drive	File (String of bytes)
DNS	Fully Qualified Domain Name	IP	Network Interface	Host
ARP	IP	MAC	Network Interface	Host

More Terminology**Location Independent Name**

Name independent of address, e.g. IP, FQDN

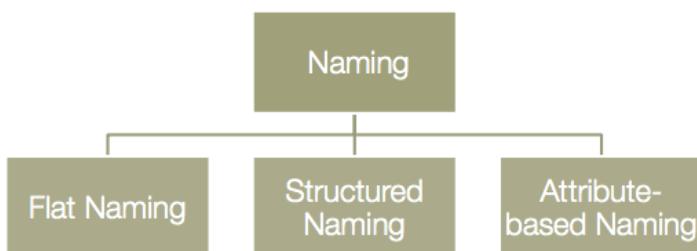
Identifier

A name with the following properties. Refers to at most one entity. Each entity is referred to by at most one identifier. Always refers to the same entity (it is not reused).

Human Friendly Name

Tailored to be used by humans, generally represented as a string of characters.

Flat Naming

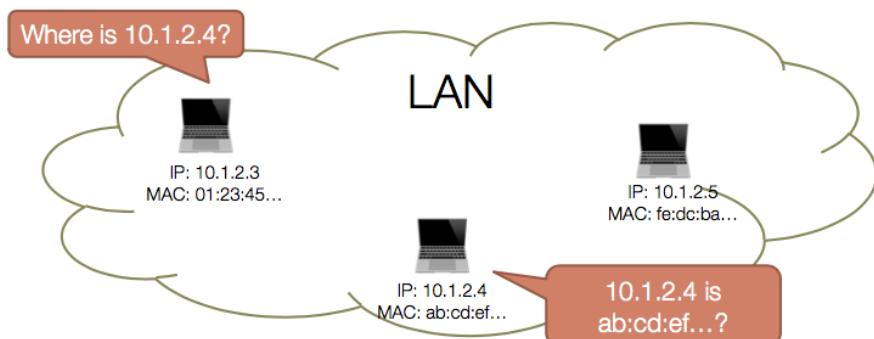


Names are treated as unstructured identifiers. (Note: Names may have an internal structure (e.g. IP address), but this is not used for name resolution). Often, but not always:

- Uniquely represents entity (identifier)
- Contains no information on how to locate entity
- May be a random string

Broadcasting

A simple solution for flat naming, Entities have access to a broadcast medium, each entity knows its name and address. Resolution of name x works as follows: Broadcast „Where is X?“ - Entity X answers with the address.



- The local area network (LAN) is the broadcast medium
- New nodes try to find an IP address unique to the LAN
- Each node has an interface with a MAC address

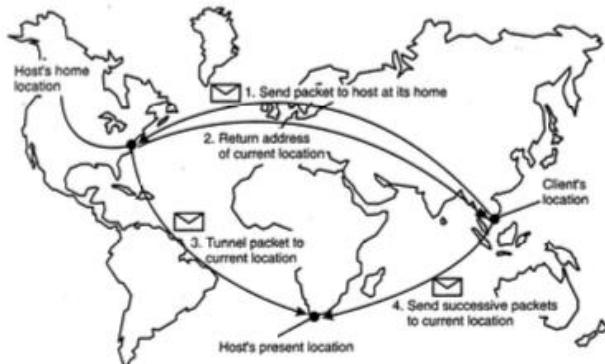
Pro and Cons

- (+) Self-managed naming system: no master node required
- (+) Simple to implement
- (+) Location independent (Unabhängig)
- (-) Does not scale – Communication overhead
- (-) Easy to exploit, an entity may impersonate another

Home-based Approaches

How is it possible to overcome performance problems with broadcasting in large scale networks, especially when hosts are mobile? – Each host has a home location that tracks its current location. Client query home location for the actual location.

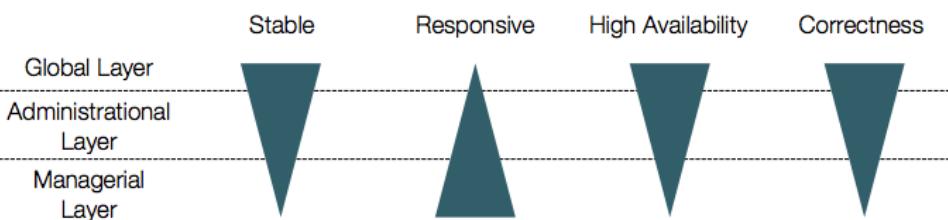
Mobile IP



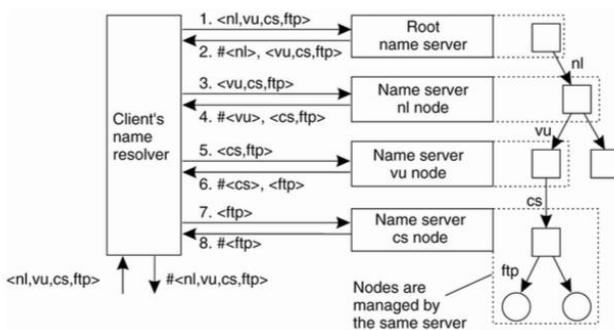
Home Location resolves IP address to another IP address. This results in a high latency for initial request. The home location cannot be moved and the home location must always be reachable.

Structured Naming

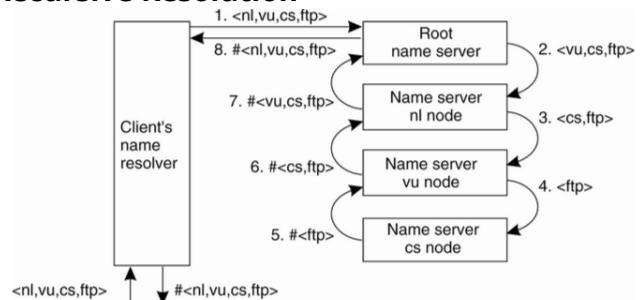
Names are hierarchically structured. For example `unterricht.hsr.ch`. Name resolution can be split into several layers with different responsibilities and requirements. For Example DNS.



Iterative Resolution



Recursive Resolution



This approach results in a high performance demand on each name server. But better caching is possible.

Attributed-based Naming

Aim: The user searches for entities based on a description of what he is looking for in the form of (attribute,value) pairs.

Directory Services == Attribute-Based Naming System. The challenge is here to designing an appropriate set of attributes for a diverse set of people and applications.

LDAP	Lightweight Directory Access Protocol
Mixture	Structured Naming + Attribute based Naming
LDAP Directory Service	= Collection of Directory Entries
Directory Entry	Collection of Attributes
Attribute	(key,value) pair

LDAP Information Model

- Within the information model of LDAP, data is stored in entries, which build up a hierarchical, tree like structure.
- Each entry has a unique name (*DN, Distinguished Name*), which depicts its position within the tree.
- An entry consists of key/value pairs, the *attributes*.
- Some attributes may occur more than once within an entry (single or multi valued, e.g. a person can have more than one telephone number).
- So called *object classes* define, which attributes an entry may have, and which of them are required.
- The classes build up a hierarchy with *top* as root; there is a parallelism to the object oriented world. *top* forces only the attribute *objectclass*, which assigns an entry its object classes.
- A *schema* consists object classes and attribute types, and therefore defines, what kind of entries can be stored within the directory.
- Directory servers ship a schema out-of-the-box, often with elements standardized by RFCs. In addition, most directory solutions allow you to define custom object classes and attributes. But in practice, the pre-defined elements are used. Sometimes they get extended according to special requirements.

Case Study – Bonjour / Zeroconf

An example of a network protocol, that uses various aspects of naming. Allow to add and use devices in a network without manual configuration. Combines various protocols and services:

1. DHCP for assigning IP addresses
2. mDNS (multicast DNS) for hostname resolution
3. Service Discovery Protocol (SDP)

Only works in a local network and requires users to trust the entire network.

Distributed Hash Tables

Motivating Example – P2P Cloud Storage

We want to design a P2P Cloud Storage System (i.e. a distributed P2P “Dropbox”). Each node contributes some of its local storage, and in turn can replicate some of its data on other nodes.

How can I determine the location (address of responsible node) of a replicated file given its name?

A First Attempt

We use the same technique as for a Hash Table. Key = id of file = hashCode(fileURL) and the value = id of responsible node-. The location is then location(fileURL) = hashCode(fileURL) % nrOfNodes.

But there is a problem. Adding/removing locations requires redistribution of almost all entities. For example: Resizing from 3 to 4 locations, 9 out of 12 entries are reassigned to a new location.



Loc. 1	1	4	7	10
Loc. 2	2	5	8	11
Loc. 3	3	6	9	12

Loc. 1	1	5	9	
Loc. 2	2	6	10	
Loc. 3	3	7	11	
Loc. 4	4	8	12	

Solution – Consistent Hashing

A special kind of hashing that allows efficient resizing. Requires, on average, to only move K / N entries (K: number of keys, N: number of locations). Example: Resizing from 3 to 4 locations, only $12/4 = 3$ entries are reassigned to a new location.



Loc. 1	1	4	7	10
Loc. 2	2	5	8	11
Loc. 3	3	6	9	12

Loc. 1	1	4	7	
Loc. 2	2	5	8	
Loc. 3	3	6	9	
Loc. 4	10	11	12	

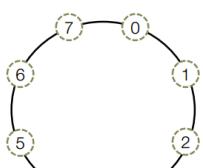
Note: This example is purely illustrative and has no relation to the Chord system presented later

Chord

A Chord is a Scalable Peer-to-peer Lookup Service for Internet Applications. An implementation of consistent hashing in a P2P network. Provides one main operation: Mapping keys to locations in the P2P network.

Namespace

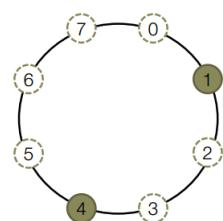
Uses m-Bit identifiers for nodes and keys. Namespace size: 2^m . The value for m is fixed and typically large (e.g. 128, 160). Uses a hash function with the range $[0, 2^m - 1]$. Nodes can be considered to belong to a logical ring. Example: Ring (empty) with $m = 3$.



Nodes

Nodes are identified by an id p. Chord: p is the hash of a node's IP Address: $p = \text{hash}(\text{IP})$. Example: Ring with two

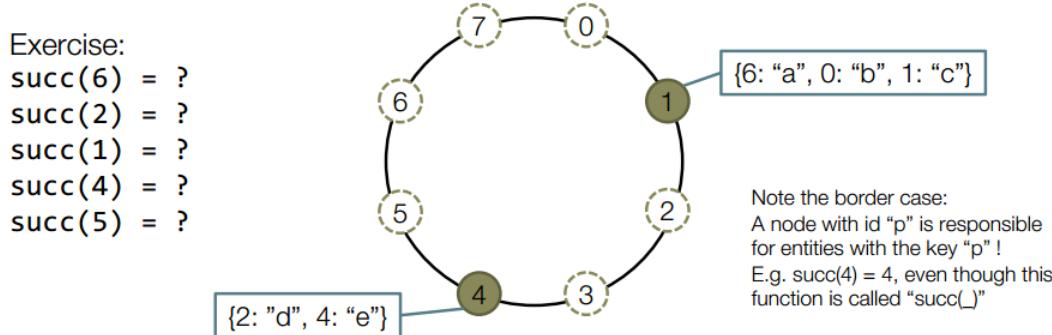
nodes 1 and 4:



Successor

Each node maintains a link to its successor node. $p.\text{successor} = \text{first node clockwise from } p \text{ in the ring}$. Examples: $1.\text{successor} = 4$ and $4.\text{successor} = 1$.

Resources (entities) are identified using keys. An entity with key k falls under the jurisdiction of the node with the smallest identifier p such that $p \geq k$. This "responsible" node for key k is denoted by "succ(k)". Example: 5 string resources distributed over 2 nodes:



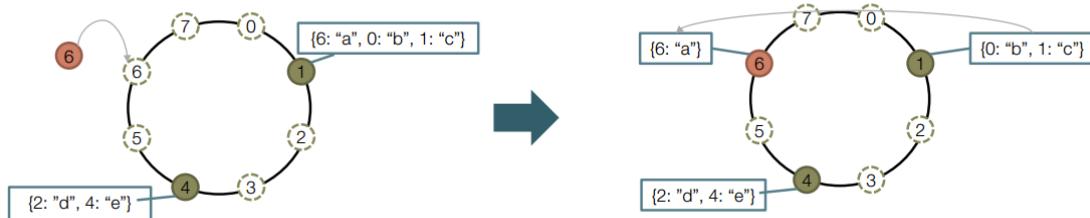
Succ(_) vs. __successor

The name "succ(_)" is not well chosen since it can be confused with the field ".successor". succ(_) is a distributed function, whereas .successor is a local function !! succ(k) should be thought of as lookup(k) or location(k) on the entire ring. For all nodes p within a correctly constructed ring, $p.\text{successor} = \text{succ}(p+1)$.

Joining

A new node n looks up its successor $\text{succ}(n+1)$ and joins the ring. $n.\text{successor} := \text{succ}(n+1)$. Predecessor of $\text{succ}(n+1)$ needs to update its successor to n . All resources at $\text{succ}(n+1)$ with keys k such that $\text{succ}(k)=n$ are moved to n .

Example: Node 6 joins



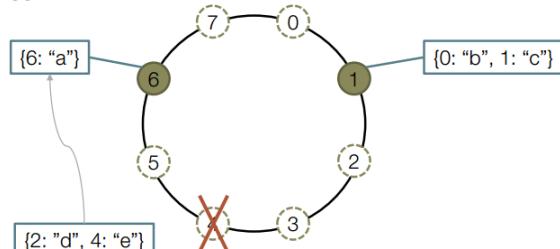
Predecessor

To allow efficient joins, each node keeps track of its predecessor as well. $p.\text{predecessor} = \text{first node counter-clockwise from } p \text{ in the ring}$. Example 1.predecessor = 4 and 4.predecessor = 1.

Leaving

A node n leaves the ring. All resources from n are sent to $n.\text{successor}$. $n.\text{predecessor}$ is advised to change its successor to $n.\text{successor}$.

Example: Node 4 leaves



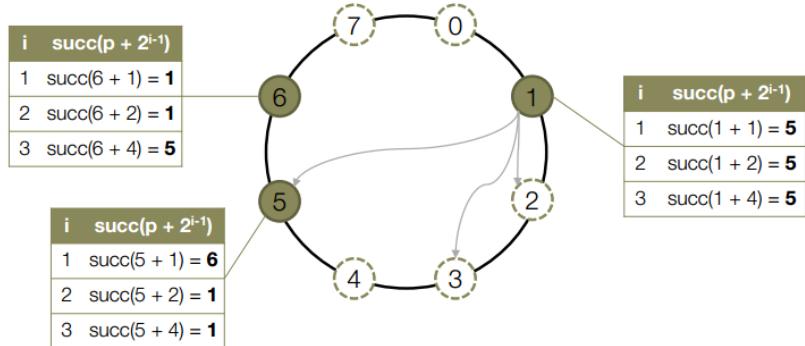
Lookup

How can I perform a lookup? → Naïve approach: Linear search. Each node knows its successor, query is passed clockwise until it reaches the successor node of the query key. Complexity: $O(N)$ (N = number of nodes).

Can I do better? (N can be large). Idea: faster lookup with “shortcuts” in finger table (similar to binary search).

Finger Table

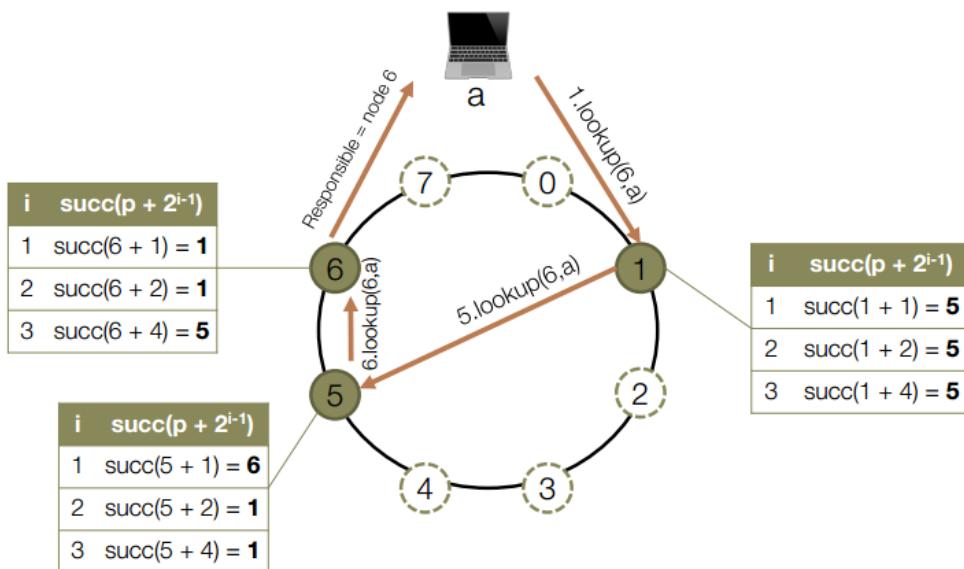
Each node p has a table with index $i \in \{1, \dots, m\}$. $p.\text{finger}[i]$ points to $\text{succ}(p + 2^{i-1})$. $p.\text{finger}[1] = \text{succ}(p + 1) = p.\text{successor}$ (i.e. the first entry in the finger table is the successor of node p).

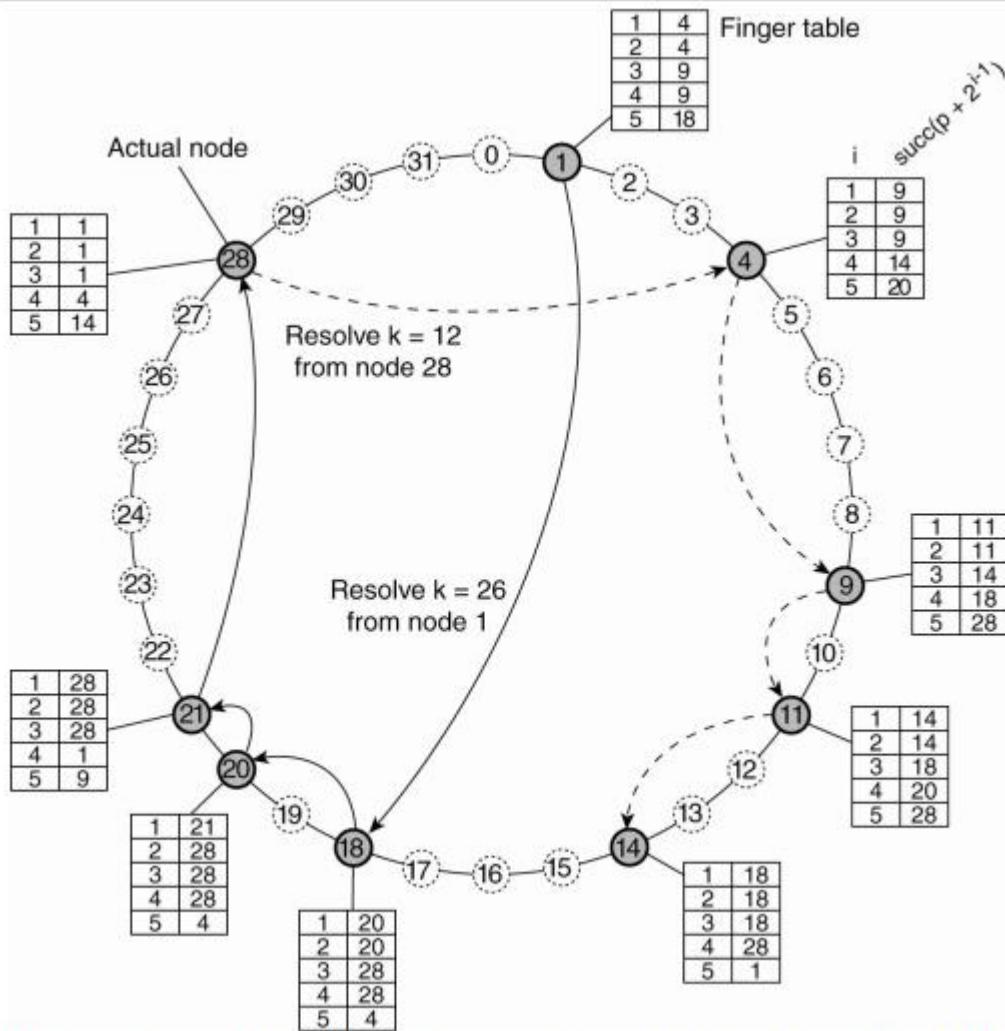


$p.\text{lookup}(k, a)$: Perform a lookup for key k from node p and send the id of the responsible node to address a :

1. If I am responsible for k (i.e. $p.\text{predecessor} < k \leq p$), send my id (and the resource, if applicable) to a .
2. Otherwise, if my successor is responsible (i.e. $p < k \leq p.\text{successor}$), forward the lookup request to my successor ($p.\text{successor}.\text{lookup}(k, a)$).
3. Otherwise, forward the lookup request to the nearest predecessor q of k in my finger table($q.\text{lookup}(k, a)$)

Example: Node with address “a” looking up resource with key 6 from node 1





Chord - Correctness

If finger table is correct, lookup is $O(\log N)$. If successor pointers ($p.\text{finger}[1]$) are correct, lookup will always yield the correct result. But concurrent joins and failing nodes make the information stored at each node (i.e. `_successor`, `_predecessor`, `_finger[_]`) outdated. Each Chord node periodically runs a stabilization procedure to correct outdated information. Due to stabilization, in most practical settings, the network converges to a consistent state.

Chord - Stabilization

Successor

Each node p periodically verifies and updates its immediate successor:

```
x := ( p.successor ).predecessor;
if ( p < x < p.successor )
    p.successor := x
```

} Executed at node 'p'

Example: $p.\text{successor}$ does not point to its immediate successor (Note: $p < x < q$).



Predecessor

Each node o periodically notifies its successor p of its existence. Upon receiving this message, each node p verifies and updates its current predecessor.

```
if ( (p.predecessor is undefined) || (p.predecessor < o < p) )
    p.predecessor := o
```

Case when 'p' has just joined the ring

Executed at node 'p'

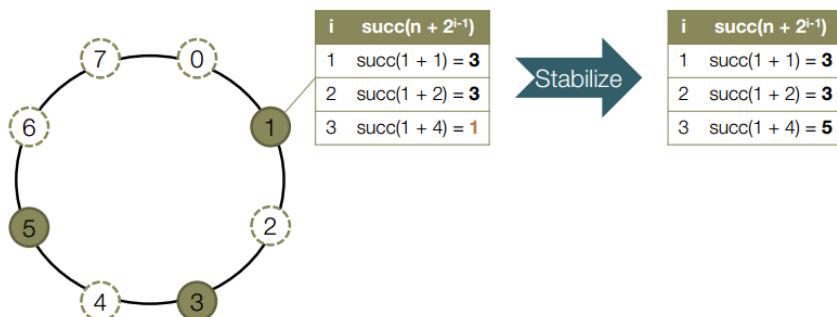
Example: $p.\text{predecessor}$ is outdated, p is notified by o (Note: $n < o < p$).

**Remaining finger table entries**

Each node periodically refreshes random finger table entries:

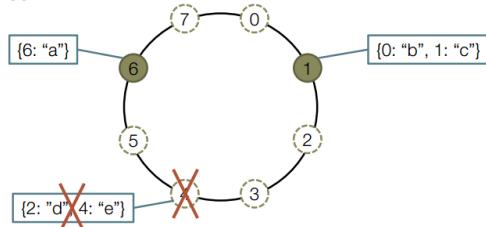
```
i := random integer such that 1 > i ≥ m;
finger[i] = succ(n + 2i-1)
```

Example: Node 5 joined, 1.finger[3] is out of date.

**Leaving (unplanned)**

A node n leaves the ring unexpectedly. All resources from n are lost. The ring is broken & lookups fail.

Example: Node 4 leaves



How could it be possible to keep the ring intact? Possible solution: keep track of the first n successors

Chord - Replication

Possible solution: Replicate files r times by generating r keys for each file. Store replica $i \in \{0, \dots, r-1\}$ at key $(\text{hash(fileURL)} + i * 2^m/r) \% 2^m$. Periodically run background job to verify and make sure that all copies can still be found.

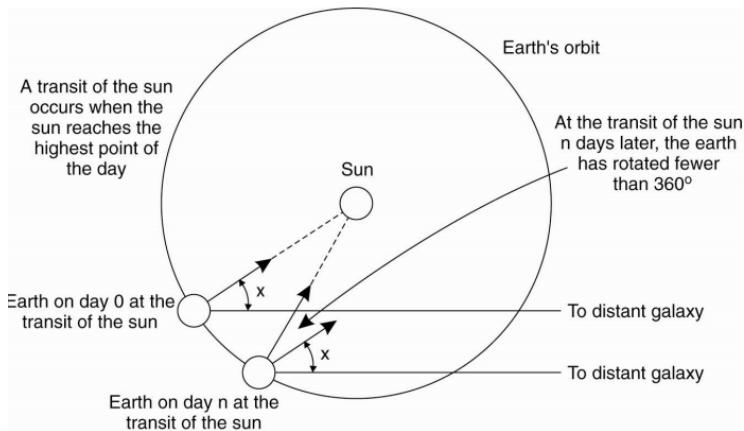
Synchronization

Why is the synchronization of time important in a distributed system? Because distributed means many clocks. The events still needed to coordinate at a global level.

Side Note: According to the special theory of relativity, there is no such thing as "global time". Nevertheless, this is of little consequence though in our day to day lives.

Two types of clocks

Physical clocks (Basis: Astronomy, Mechanics)



Physical clocks have their limits w.r.t accuracy (more on this later). Subject to numerous "bug fixed" over the centuries (e.g. leap year, leap second). The usage is problematic due to superfluous conversions (e.g. time zones, calendar formats). It is still widely used for synchronization in the "human context". People have tried to change this and have failed.

Excursion: Digital Time

Instead of hours and minutes, the mean

solar day is divided up into 1000 parts called ".beats". Each .beat is equal to one decimal minute in the French Revolutionary decimal time system and lasts 1 minute and 26.4 seconds in standard time. Times are notated as a 3-digit number out of 1000 after midnight. So @248 would indicate a time 248 .beats after midnight representing 248/100 of a day, just over 5 hours and 57 minutes. There are no time zones in Swatch Internet Time, instead, the new time scale of Biel Meantime (BMT) is used, based on Swatch's headquarters in Biel, Switzerland.

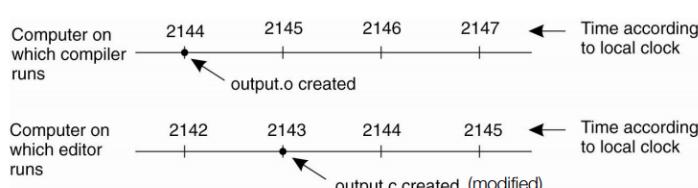


Figure 6-1. (Tannenbaum) When each machine has its own clock, an event that occurred after another event may nevertheless be assigned an earlier time.

Also important for computer systems

When each machine has its own clock, an event that occurred after another event may nevertheless be assigned an earlier time. Plays havoc with systems like "make".

⇒ The compiler will not be run on the modified file.

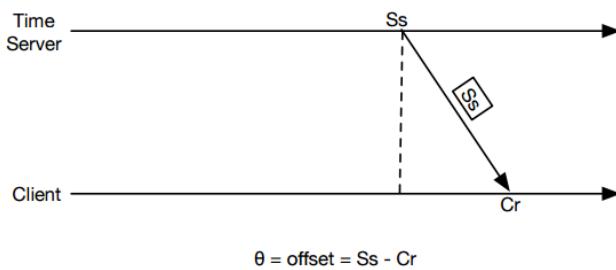
Q) How is the SBB clock synchronised?

Bahnhofsuhr weitergegeben werden. So laufen sie ab 1944 im selben Takt [Abb. 6]. Hans Hilfiker schreibt zu seiner Uhr im Magazin *Schweiz – Suisse – Svizzera*: «In der Sprache des Fachmannes ist eine Bahnhofsuhr eine Nebenuhr – gleichviel, ob ihre Zeiger in Bleistiftgrösse auf einer Bürowand drehen oder ruderlang sich auf der Bahnhofswand bewegen. Sie hat keinen eigenen Antrieb, ist kein selbständiger Zeitmesser; darum heisst sie so. Mehr als 59 Sekunden jeder Minute ist die Nebenuhr stromlos; nichts bewegt sich im Werk, und die Zeiger stehen bockstill. Eine grobe Zeitmessung! Aber ihrer Feinheit, der genauen zeitlichen Lage des Minutensprungs wegen ist sie dennoch für den Bahnbetrieb brauchbar; der Fahrplan kennt ja nur ganze Minuten. Als Kelle geformt kreist der Sekundenzeiger mit seinem weithin sichtbaren, runden Fleck auf dem Zifferblatt, und seine rote Farbe charakterisiert ihn, mindestens bei Tag, zusätzlich. Sein stumpfes Zeigerende ist eine formale Konsequenz der periodischen Ungenauigkeiten, die unvermeidlich, hier aber belanglos sind.»



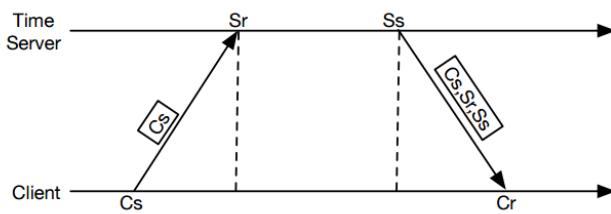
One-sided update

The master clock periodically notifies slaves about the current time. What would be the problem if this technique is used to synchronize time on the internet? => Network latency affects accuracy.



Network Time Protocol (NTP)

Client adjusts its clock by slowing down or speeding up its clocks interrupt interval.



Logical clocks (Basis: Causality)

Motivation

It is often more important that processes agree on the order of events rather than the exact point in time (physical clock) at which an event happens. Physical clocks have limited resolution and accuracy: Two events may be causally related and still have the same timestamp (physical clock). But first we need to define what we mean by “causally related”

Running Example – Updates on a replicated database

We have a replicated database over N peer nodes. Update can be initiated by any node. Goal is that the database must be “consistent”. Assumptions:

- All nodes know all their peers a-priori
- Nodes communicate via a multicast medium
- Messages may be lost, but not reordered

Solution 0

The node that wishes the update, updates its copy of the database and sends the desired update to all its peers.

Problem 0

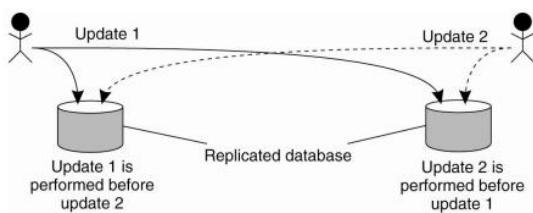
Update messages could be lost,

Solution 1

All nodes must acknowledge the receipt of all updates to all other nodes. Each node maintains a set of pending updates. A node performs an update only if the update has been acknowledged by all its peers.

Problem 1

It can be, that the order in which update are performed is different at different nodes.



- Account A contains 1000\$
- u_1 : Withdraw 100\$ from account A
- u_2 : Increase account A with 1% interest
- Order is crucial:
 - $u_1; u_2: (1000 - 100) * 1.01 = 909$
 - $u_2; u_1: 1000 * 1.01 - 100 = 910$

Solution 2

Each node timestamps all messages it sends with the value of its physical clock. Pending updates are stored in a priority queue (priority = timestamp). A node can only perform updates in the order they are present in the queue.

Problem 2

Physical clocks may not be correctly synchronised. Even with reasonable synchronisation, it can still be the case, that two updates have the same timestamp.

Solution 3

Totally ordered multicasting Use Lamport’s logical clock as the timestamp. (Note: We have not made any assumptions on the update operation. This approach can be generalised to “state machine replication”)

Problem 3

Node failure results in blocking the queue. Updates have a very high latency (need to be acknowledged by all peers). Note: These problems are inherently due to our hard consistency requirements.

Solution 4

Use a vector clock to version updates. Concurrent updates can then be identified and merged. Such as solution is implemented in DynamoDB.

Causality

Causality is defined using the happens before relation " \rightarrow ": $a \rightarrow b$ (event a happens before event b). $a \rightarrow b$ iff at least one of the following conditions are true:

- a and b are in the same process and a occurs before b
- a is the event of sending a message and b is the event of receiving the same message
- $\exists x. a \rightarrow x \wedge x \rightarrow b$ (transitive closure)

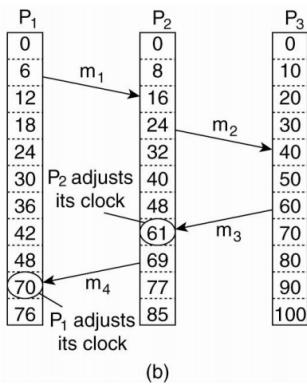
\rightarrow is a strict partial order (think: DAG) (*Note: Asymmetry can be derived from irreflexivity and transitivity using a proof by contradiction.*)

- irreflexive: $a \not\rightarrow a$
- transitive: $(a \rightarrow b \wedge b \rightarrow c) \Rightarrow a \rightarrow c$
- asymmetric: $a \rightarrow b \Rightarrow b \not\rightarrow a$

Definition of the concurrent relation " $| |$ ": $a | | b \Leftrightarrow (a \not\rightarrow b \wedge b \not\rightarrow a)$

Lamport's Logical Clock

Each process P_i maintains an internal counter $C_i(a)$ that assigns a number to all events a occurring in process i . Each process P_i increases $C_i(.)$ between any two successive events. The current value of the counter is sent with each message. If the event a is "sending m from P_i " and the event b is "receiving m on P_j ", then $C_j(b)$ is set to $(\max(C_i(a), C_j(b)) + 1)$.



Note: In this example, the counters at each process are updated at different rates, but this does not affect the validity of properties discussed.

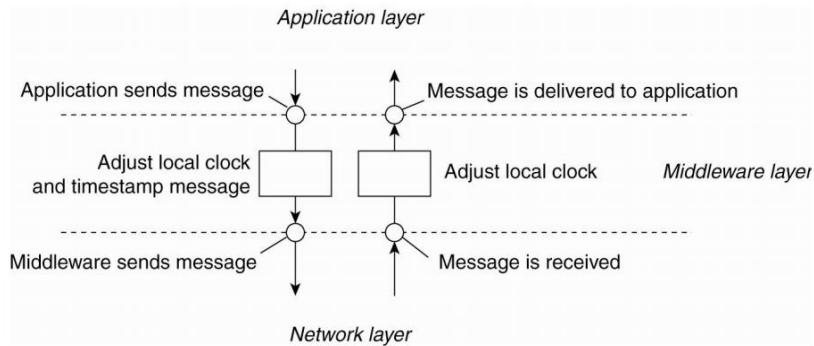
- Note:
- $C(.)$ preserves causality:
 $a \rightarrow b \Rightarrow C_i(a) < C_j(b)$
Examples?
 - Converse not true:
 $C_i(a) < C_j(b) \not\Rightarrow a \rightarrow b$
Examples?
 - Q. Is $C(.)$ totally ordered?
A. Not yet. We need to introduce a fixed, but arbitrary order between events on different processes with the same counter value to make it a strict total order.
Idea: use the process ID to order concurrent events.

Each process P_i maintains an internal counter $C_i(a)$ that assigns a number to all events a occurring in process i . The (distributed) function $C(.)$ assigns the tuple $C(a)$ to any event a , where $C(a) = (C_i(a), i)$ where a is an event in process i . The (overloaded) lexicographic ordering ".

Properties of $C(.)$

- " $C(.) < C(.)$ " is a strict total order (think: linked list) over events:
 - o The relation $C(a) < C(b)$ is a strict partial order over events a & b . (property of the lexicographic ordering "
 - o Additionally comparable: $(C(a) < C(b)) \text{ xor } (C(a) > C(b)) \text{ xor } (a = b)$
- $C(.)$ preserves causality: $a \rightarrow b \Rightarrow C(a) < C(b)$

Result: With $C(.)$ we have a global (distributed) counter (timestamp) that is total and preserves causality!



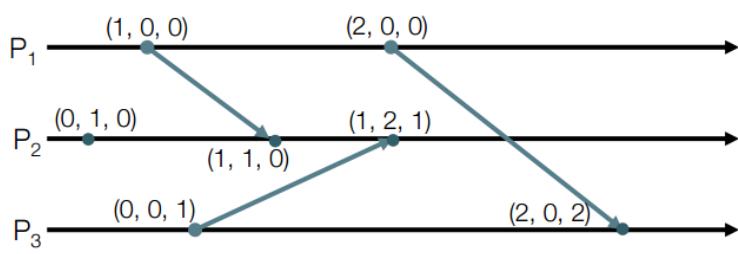
Vektor Clocks

Since Lamport's Clock defines a total order on events, it cannot capture causality exactly:

$$a \rightarrow b \nLeftrightarrow C(a) < C(b)$$

In some cases it is useful to know that two events are not causally related. For example: To detect conflicts amongst distributed updates and scheduling and debugging concurrent computations. Vector clocks track causality between events exactly.

$$a \rightarrow b \Leftrightarrow V(a) < V(b)$$

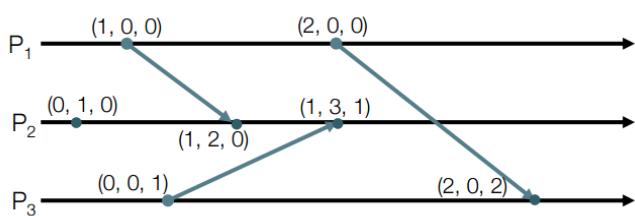


Each process P_i maintains an internal vector V_i with the following properties:

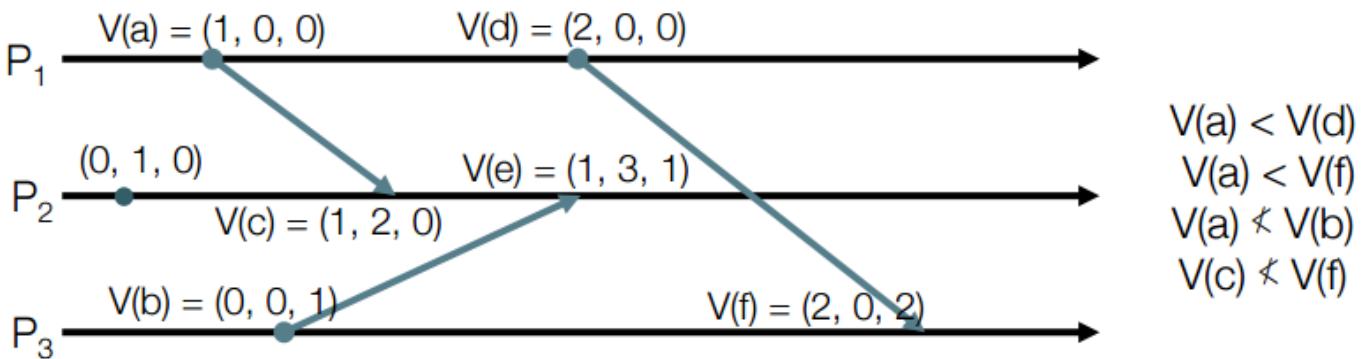
1. $V_i[i]$ is the number of events that have occurred so far at P_i .
2. If $V_i[j] = k$ then P_i knows that k events have occurred at P_j .

Vector clock properties are maintained with the following rules: (image below).

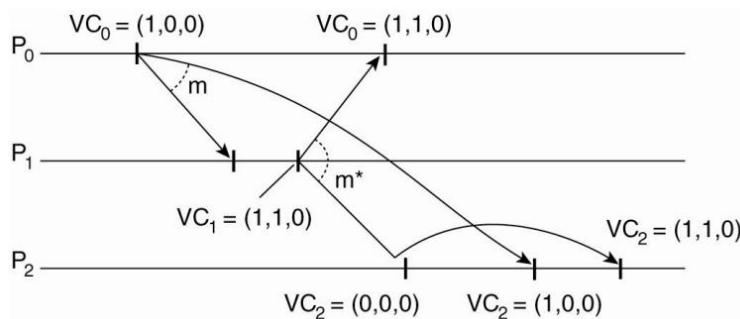
1. Increment $V_i[i]$ with the occurrence of each new event at process P_i . Note: Sending and receiving messages are also considered "new events".
2. Receiving a message at P_j from P_i causes P_j to additionally update each entry $V_j[k]$ to $\max(V_i[k], V_j[k])$.



$V(a)$ is vector $V_i(a)$ of the event a occurring at process P_i . (i.e. Process IDs are not relevant) $V(a) < V(b)$ iff $V(a)[k] \leq V(b)[k]$ for all process indices k and there is one k for which $V(a)[k] < V(b)[k]$. Vector clocks define a causal order between events: $V(a) < V(b) \Leftrightarrow a \rightarrow b$



Example Application



Requirement: A message is delivered to the application only if all messages that causally precede it have also been delivered. *Note: In this setting, only send events are relevant for causality and trigger receive events. Nodes therefore do not update their own index before receiving messages, but only before send events.*

Summary

- Causality: A happens before relation between events
- Lamport's Logical Clock: Assigns a timestamp to events that defines a total order amongst them
 - o $a \rightarrow b \Rightarrow C(a) < C(b)$, but not $C(a) < C(b) \Rightarrow a \rightarrow b$
- Vector Clock: Assigns a timestamp to events that defines a causal order amongst them
 - o $V(a) < V(b) \Leftrightarrow a \rightarrow b$

Modeling causality – Reflection

Does " \rightarrow " always completely capture the concept of causality as we understand it?

Not entirely. Two events on the same process may "actually" be causally independent, but be forced into a causal dependency due to our definition of " \rightarrow ". E.g. Cooking recipes. Two events on different processes may be concurrent w.r.t. " \rightarrow " but still be causally due to externalities. E.g. Leaving a room and entering another. Therefore: The definition of what a relevant event is must be carefully considered when talking about causality.

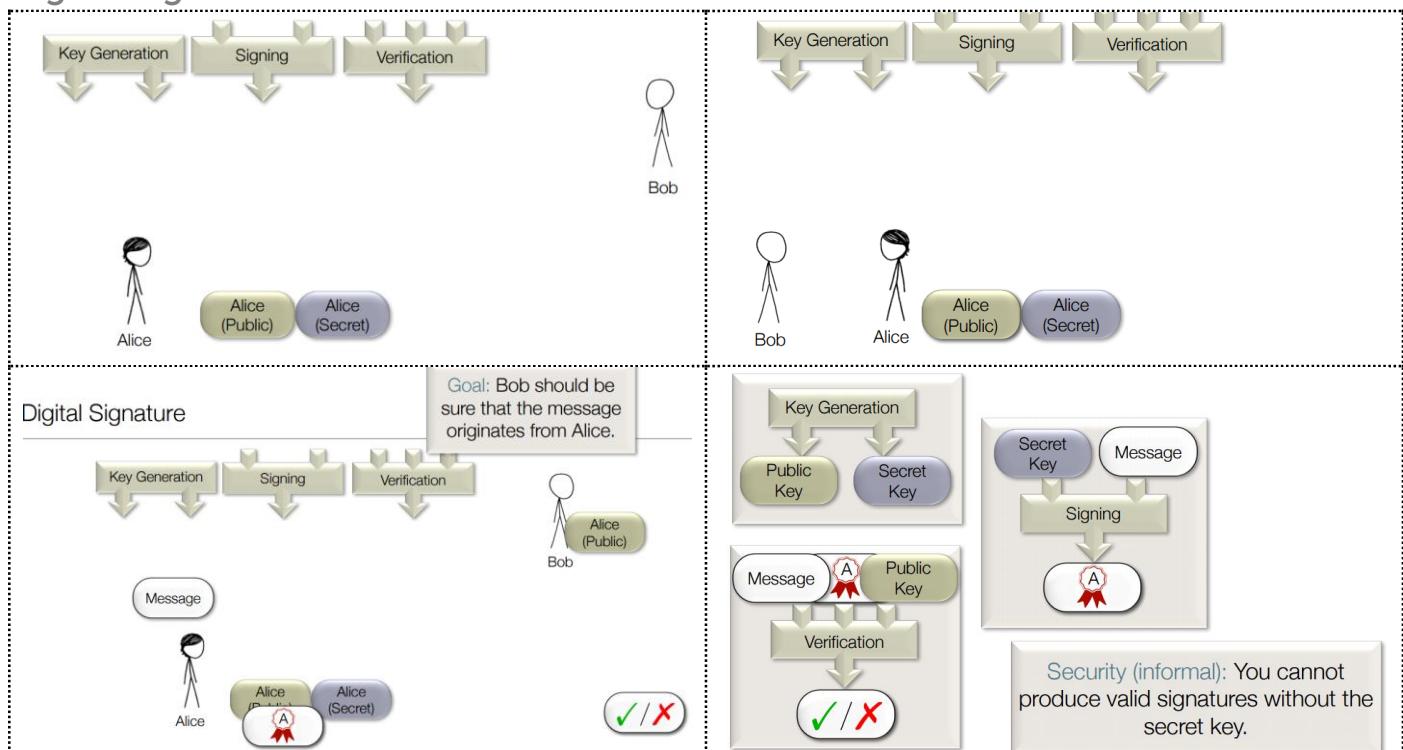
Bitcoin / Digital Money

Goals

We want some kind of “digital money” with the following properties

- Transactions possible
- No double spending
- Everyone can participate
- No central instance or trusted party – no bank.

Digital Signatures



A first attempt at designing a Bitcoin-like system

Setting

A network of computers. Every computer can send messages to some other computers.

Basic Idea

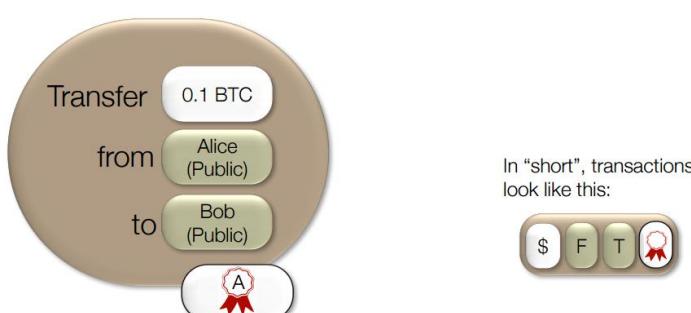
Alice (Public)	10 BTC
Bob (Public)	0.2 BTC
Charlie (Public)	17 BTC
Dora (Public)	0.001 BTC
Eliza (Public)	2 BTC

The basic idea is that every computer maintains a table “who owns what?”. The requirement is, that all computers must have the same table.

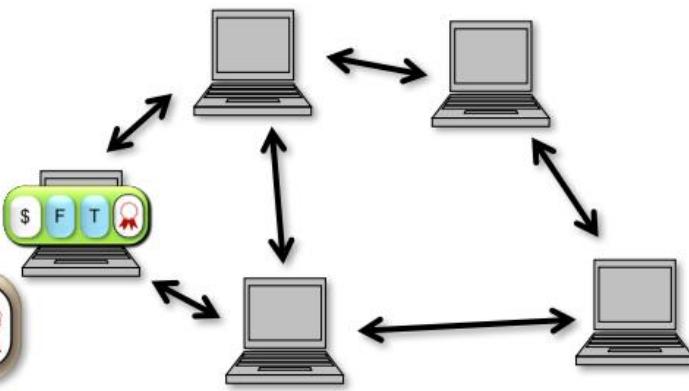
Remark: The public keys are just bit strings.

Sending Bitcoins

To send money, we use transactions. These are messages like this.



I'll send 0.1 Bitcoins to Bob.



Protocol: sending BTC

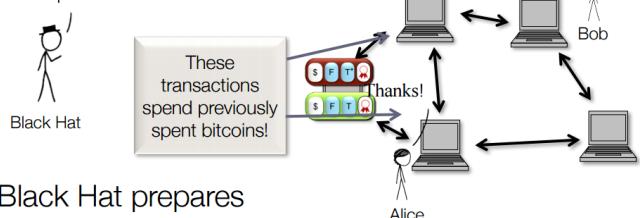
1. Craft a transaction.
2. Give it to your computer.

Protocol: participating

- On valid transactions:
1. Update ledger
 2. Relay transaction

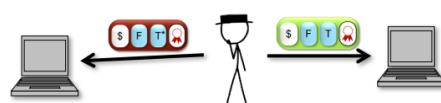
Double Spending

I can exploit this!



Black Hat prepares two transactions:

- (\$ F T R): Give BTC from Black Hat to Alice
- (\$ F T R): Give BTC from Black Hat to Bob

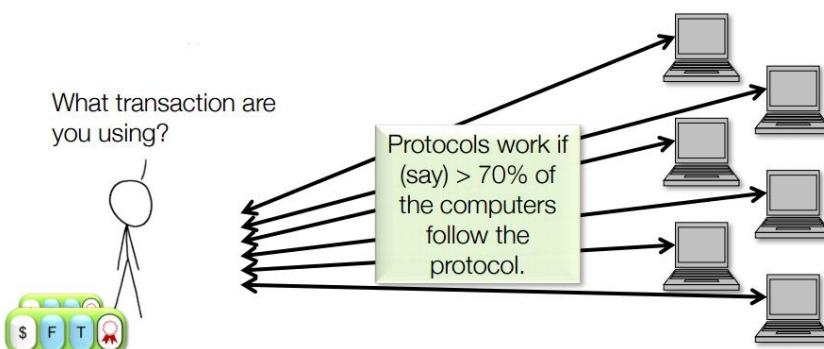


Black Hat spends the same Bitcoins with two different transactions (\$ F T R and \$ F T R).

Computers receiving transaction (\$ F T R) will have a *different* ledger than computers receiving transaction (\$ F T R).

Consensus Protocol

So we need a protocol to agree on a transaction. The “Consensus protocols” are studied since 1980. Starting with Pease, Shostak and Lamport. Has a huge literature!. This is the main idea of the protocol.



This solution does not help us. The design goal, is the everyone can participate. By running a special programm, black hat controls many virtual computers. Like this, he can make different participants believe different things.

A random hash function with a k bit output is

$$\text{RH}: \mathbb{B}^* \rightarrow \mathbb{B}^k \quad \text{or equivalently...}$$

$$\text{RH}: \text{File} \rightarrow \{0, \dots, 2^{k-1}\}$$

where all outputs are chosen uniformly at random, independent of each other. Example:

$$X := \text{RH}(\text{"text"}) \rightarrow x = 44709335$$

$$X := \text{RH}(\text{"next"}) \rightarrow x = 53639915$$

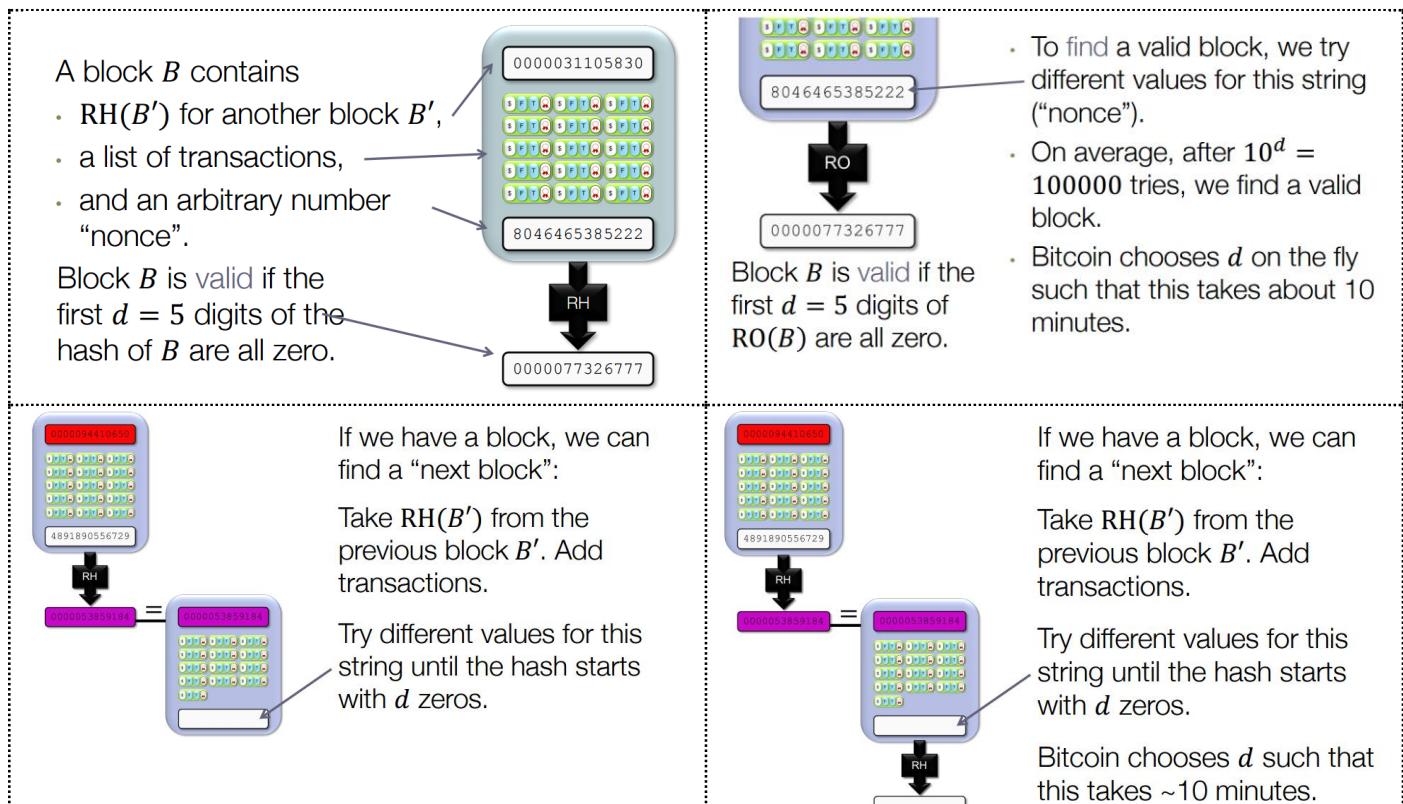
$$X := \text{RH}(\text{"text"}) \rightarrow x = 44709335$$

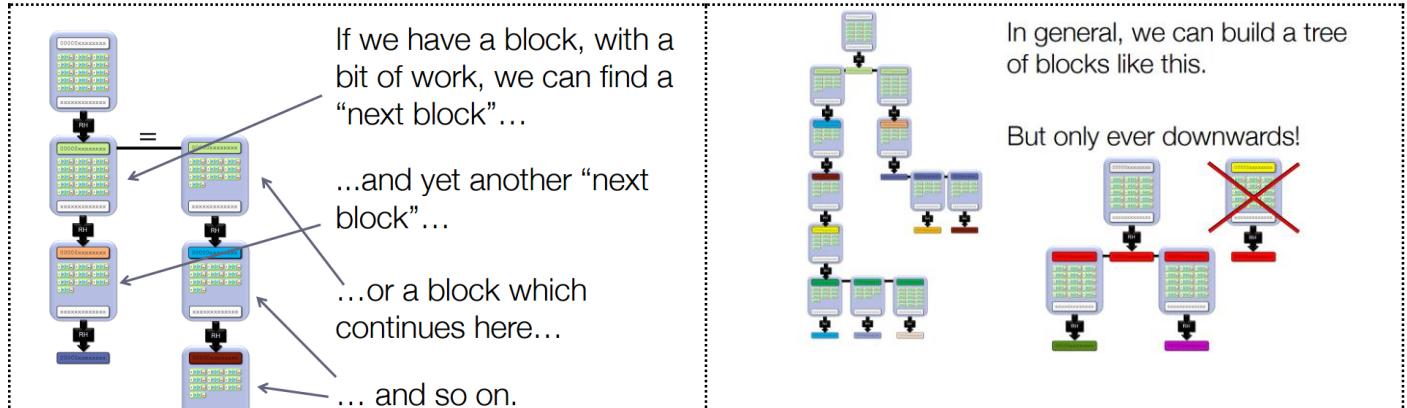
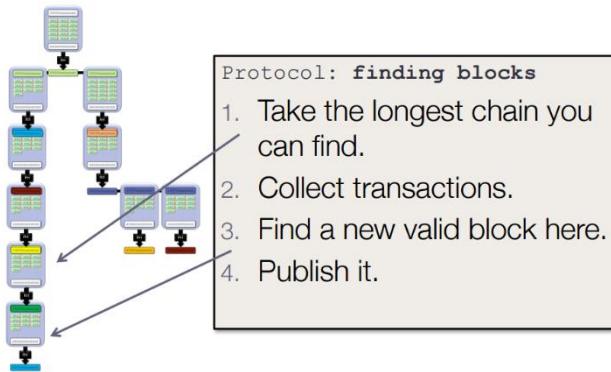
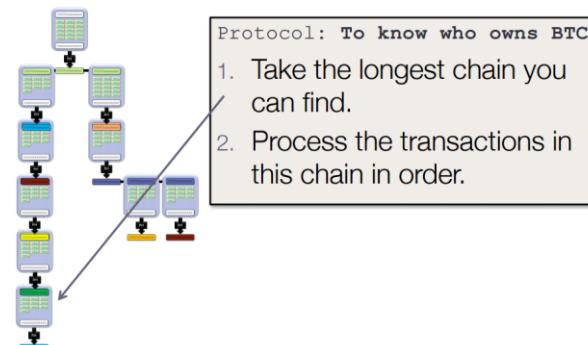
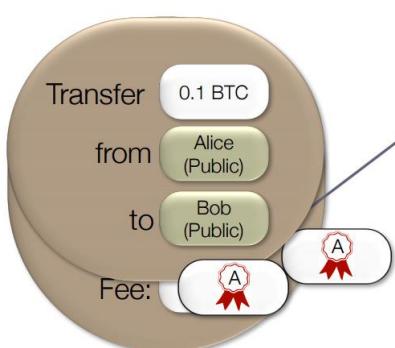
- In practice, we assume that SHA256 is a random hash function. Justification: If we made all computers in the world compute SHA256, it would take $\sim 10^9$ years to find x_1, x_2 such that $x_1 \neq x_2 \wedge \text{SHA256}(x_1) = \text{SHA256}(x_2)$.

Bitcoin's consensus protocol

Step 1 - How does the protocol work?

Blocks



A Tree of Blocks**The Protocol for Finding Blocks****The Protocol for Participants****Why work to find blocks?**

Many people are trying to find blocks, which uses a lot of resources. This is called "mining".

Block reward

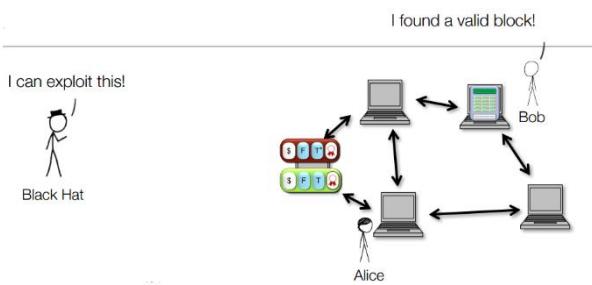
If you find a block, you get bitcoins as a reward. Every transaction specifies a fee. It goes to the person who puts the transaction into a valid block.

Recap – The Bitcoin Protocol**Protocol «participate»**

- Relay valid transactions
- Relay valid blocks in the longest chain
- Work with the longest chain

Protocol «miners»

- Collect valid transactions
- Publish valid blocks which extend the longest chain

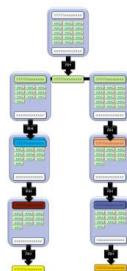
Double Spends

Once a block is found, the double spends vanish.

Occasionally, two people find blocks at around the same time... but typically the problem disappears.

Build an Alternate Chain?

Maybe I should build another chain?



The more RH-calls are devoted to a chain, the faster it grows.

Thus, intuitively: to build a chain as fast as the rest, you need as many RH-calls as the rest.

Denial of Service

If I cannot cheat bitcoin, maybe I can mess it up!



Interesting idea... ...and while Bitcoin incorporates many, many rules to handle this... ...people still try!

Some Bitcoin History

What happened? A company (MtGox) blamed problems on a “bug in the Bitcoin software”.

Can we Exploit this?

On bitfinex.com, some people lent others roughly 15'000 bitcoins (~4 Million CHF).

The others then sell them, to buy them back later.

If I can make people believe that bitcoin is broken...



I can make real money!

**Some Details**

Bitcoin doesn't use SHA256(x), but SHA256(SHA256(x)). Currently, an “initial block reward of 25BTC” is given for every found block besides the fee. Not the length, but the total difficulty of a chain is important, etc... But most of these are not important for the idea.

However, one warning: In real Bitcoin, transactions have many “inputs” and many “outputs”. If you don't specify where a BTC goes, it is a miner fee.

A word of Caution

If you are not careful, misunderstandings can make you lose money... so please apply appropriate care when playing with Bitcoin (or use the “testnet”). The bitcoin system has some serious flaws:

<https://medium.com/@octskyyward/the-resolution-of-the-bitcoin-experimentdabb30201f7#.jmoiyivxh>