Rechnernetze und verteilte Systeme

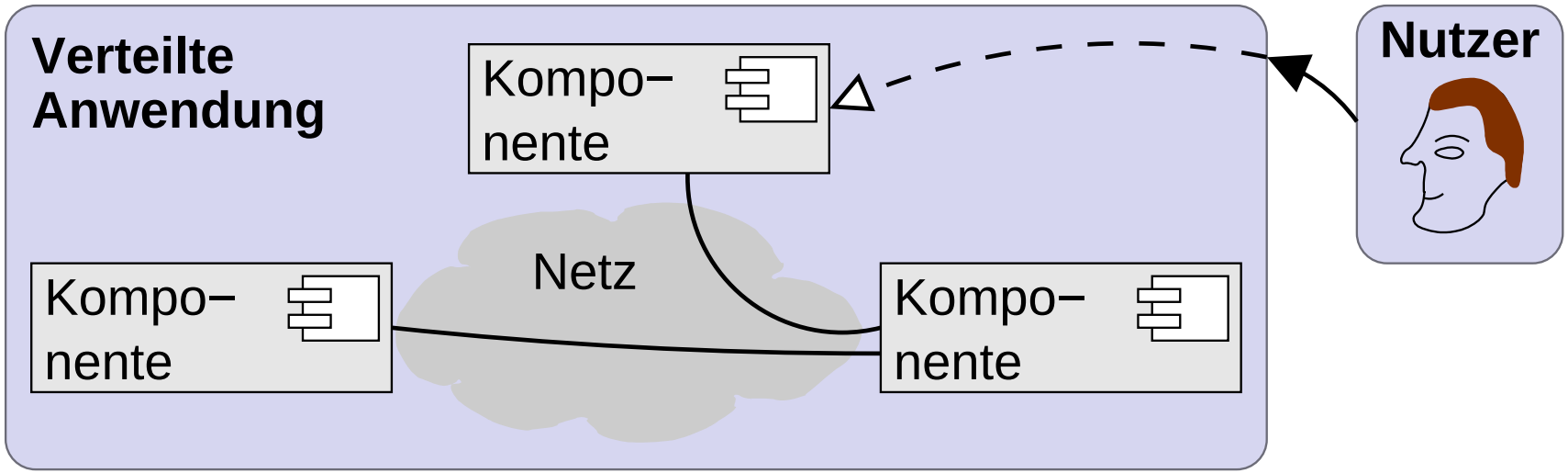
Middleware

Kapitel 9

Rechnernetze und verteilte Systeme
Middleware

Einführung

Kapitel 9.1



Herausforderungen bei verteilten Anwendungen

- Heterogenität bei
  - Datenformate (z.B. Zeichensätze, Bit-Breite von Zahlen)
  - Systemfunktionen/Programmierschnittstellen
  - Programmiersprachen beteiligter Komponenten
- Maßgeschneiderte, anwend.-spezifische Protokolle
  - Entwurf/Umsetzung schwierig/teuer
  - Änderung der Anwendung $\Rightarrow$ neue Anforderungen an Protokoll

Idee

- Einführung einer **Middleware**, die Teile einer Anwendung verbindet
- Verschattung des Zugriffs auf entfernte Komponenten
- Spezifika von Programmiersprachen im Vordergrund, statt denen der Anwendung selbst

Prozeduraufrufe Basisfunktion.

- Aufruf einer Funktion/Methode auf entfernter Komponente.
- Übergabe von Argumenten; Rückgabewerte

Typensystem ⇒Homogenisierung der Datenformate

- Gleiche Typen aus Sicht der Komponenten
- Transparente Umwandlung zwischen Systemen mit verschiedenen Formaten

Beispiele für Kommunikations-Middleware

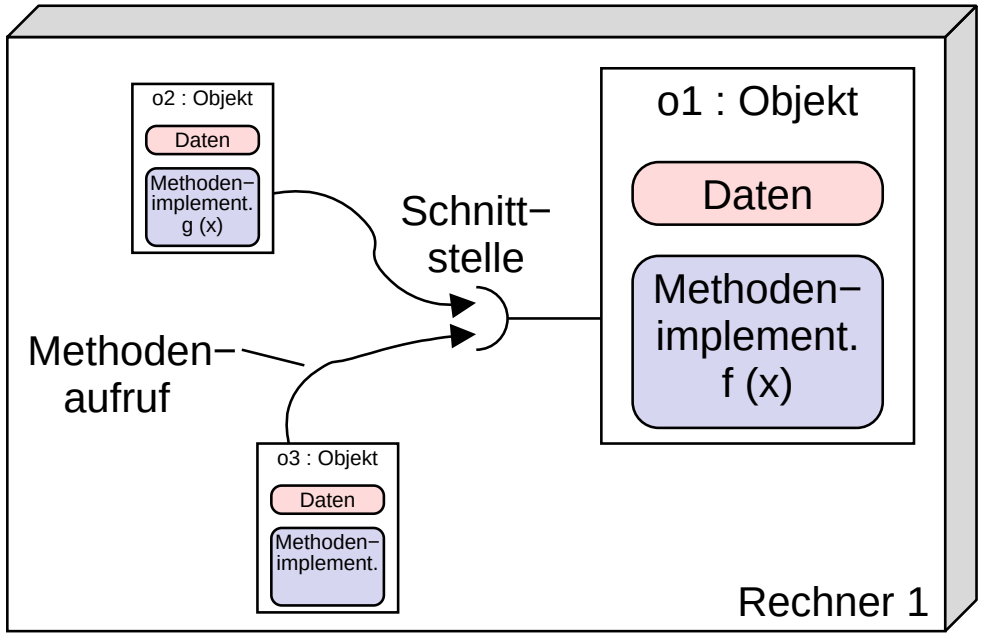
- Unix RPC
- CORBA[a] (Common Object Request Broker Architecture)
- Java Remote Method Invocation (Java RMI[b])
- SOAP[c] (Simple Object Access Protocol)

[a] OMG, Object Management Group
[b] Sun Microsystems
[c] W3C, WWW Consortium

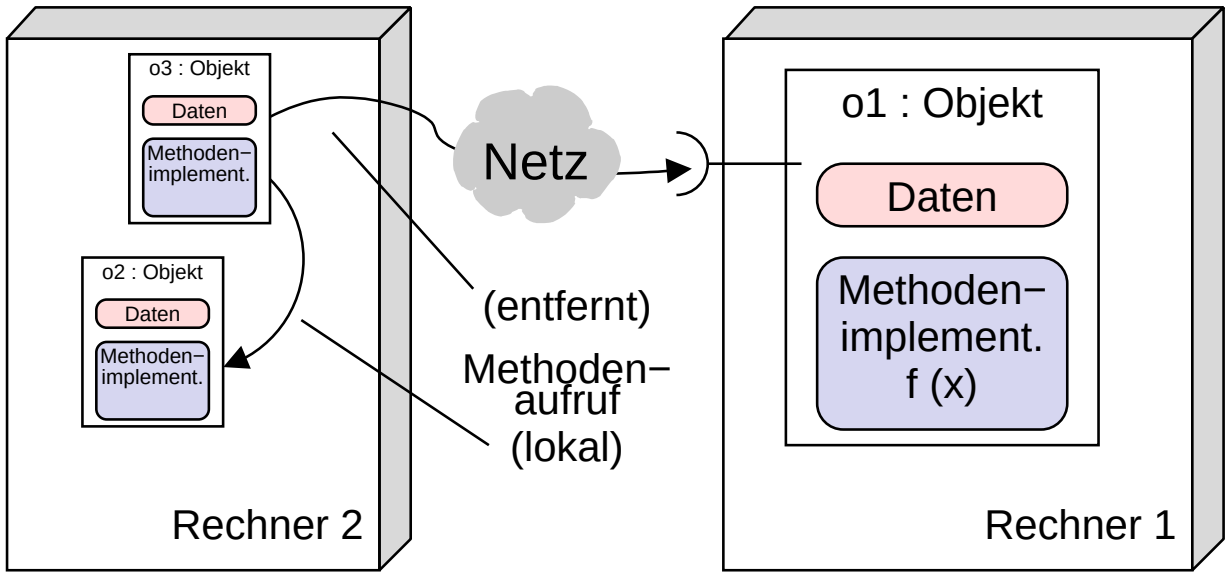
Entfernte Prozeduraufrufe: Grundlegende Fragestellungen

- Beschreibung einer Schnittstelle (z.B. mittels Beschreibungssprache)
  - Binden von Namen an Endpunkte (mittels Verzeichnisdienst)
  - Registrieren von Funktionen/Schnittstellen bzw. Objekten
- Prozeduraufrufe entsprechend
  - Verbindungsmanagement (z.B. Verwaltung von Pools von Verbindungen)
  - Serialisierung von Parametern und Rückgabewerten
  - Fehlerbehandlung (lokale und entfernte Fehler)

- Objekt: Daten (Attribute) + Funktionen (Methoden)
- Objektreferenz erlaubt Zugriff auf Daten und Aufruf von Methoden
- Schnittstelle/Interface: Definition der Methodensignaturen eines Objekts
- Aktion: i.d.R. Aufruf einer Methode eines Objekts
- Auftreten eines Fehlers erzeugt Exception
- Automatische Speicherverwaltung/Garbage collection (GC):
  - automatische Freigabe des für Objekte allozierten Speichers
  - transparent für den Programmierer



- Referenz auf entfernte Objekte (*remote object reference*)
- Referenz auf entfernte Schnittstellen (*remote interfaces*)
- Aktion: Aufruf einer Methode kann sich auf entferntes Objekt beziehen
  → kann zu Aufruf entfernter Prozedur führen
- Exception:
  - Fehlerbehandlung muss uU verteilt geschehen
  - zusätzlich: Fehler, die aufgrund der Verteilung auftreten (z.B. entferntes Objekt in unzulässigem Zustand)



Objektnamen

- Dienste, die von Objekten bereitgestellt werden: mit Namen referenziert
- Anforderung: Verzeichnisdienst, der Namen in Objektreferenzen umsetzt
- z.B. CORBA CosNaming, Java RMI Registry, DNS (bei SOAP)
- Reichweite des Namensraums: innerhalb einer Domäne, Internet-weit etc

Objektreferenzen

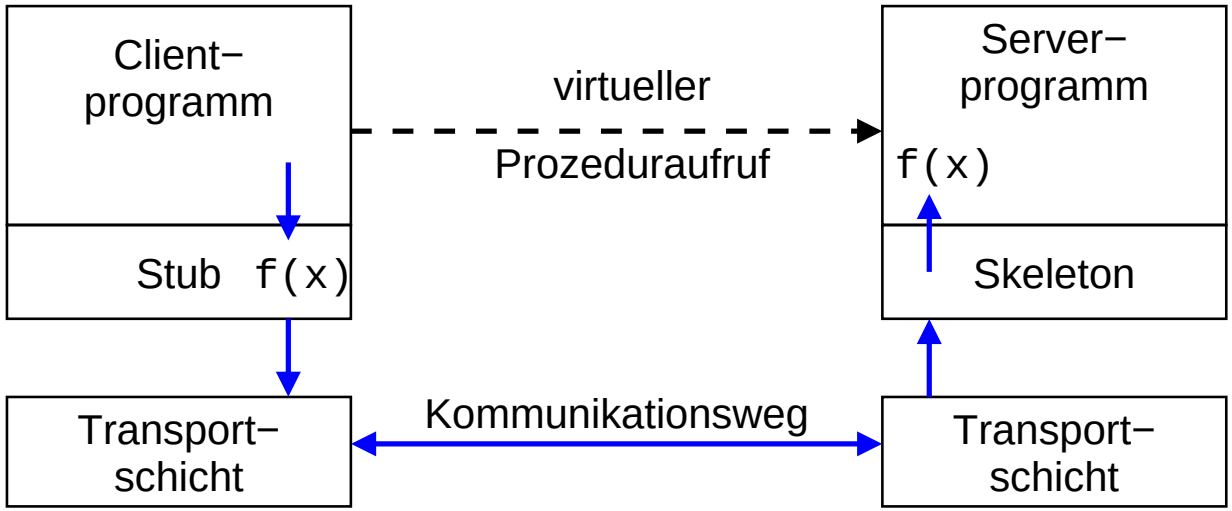
- transient (nicht-persistent): Referenz gültig, solange Objekt existiert
- persistent: Referenz gültig über mehrere Inkarnationen eines Objekts
- z.B. CORBA IOR (Interoperable Object Reference) hat Felder
  - interface repository identifier (Typ des Objekts)
  - Protokoll und Adresse, ggf mehrmals, für meherere Instanzen (z.B. IIOP, Host, Port)
  - object key (z.B. Adaptername und Objektname)

Rechnernetze und verteilte Systeme
Middleware

Entfernte Prozeduraufrufe

Kapitel 9.2

- RMI: Remote Method Invocation
- Rollen: Server (bietet Schnittstelle), Client (nutzt Schnittstelle)
- Integration in Software durch Skeletons und Stubs
- Hilfsmittel: Verzeichnisdienst



- Stubs/Skeletons: realisieren Ortstransparenz
- Client-/Serverprogramm: realisieren Dienst/Anwendung
- Marshalling: fertig stellen eines Aufrufs, so dass er versendet werden kann

- Protokollumsetzung: Anfrage/Ergebnis-Protokoll
  - Client stellt Anfragen; liefert Parameter
  - Server beantwortet Anfragen; liefert Ergebnis
- Referenzverwaltung
  - erzeugt entfernte Referenz bei abgehendem Aufruf (z.B. für lokale Objekte, die als Parameter übergeben werden)
  - erzeugt lokale Referenz für entfernte Objekte)
- Verzeichnisdienst: ermöglicht Registrierung von Server-Objekten, so dass ihr Ort (Maschine, Objekt-ID) von Clients abgefragt werden kann
- Proxy: lokaler Platzhalter für entfernte Objekte (client-seitig; ein Proxy-Objekt pro entferntes Objekt)
- Skeleton/Dispatcher: repräsentieren server-seitig ein Objekt, das von einem Client aufgerufen werden kann
- Servant: reale, server-seitige Implementierung eines Objekts; einem Servant entspricht ein Skeleton/Dispatcher-Paar.

Kommunikation unsicher ⇒ Aufruf/Ergebnis kann verloren gehen

- Keine Maßnahme zur Fehlertoleranz:
  - verlorener Aufruf: kann nach Ablauf eines Timers wiederholt werden
  - verlorenes Ergebnis: unklar, ob Prozedur durchgeführt wurde
- Wiederholung der Anfrage bis Antwort empfangen oder bis timeout (client-seitig)
  - resultiert, alleine genommen, in at-least-once-Semantik:
  - Aufrufer erhält entweder Ergebnis oder eine Fehlermeldung (Exception)
  - problematisch bei Versagen des Servers (der das Ergebnis enthält)
  - z.B. bei Sun RPC
- Wiederholung des Ergebnisses (server-seitig, ohne Neuberechnung)
- Duplikatsfilterung (server-seitig: Anfragen, client-seitig:Ergebnisse)
- at-most-once-Semantik: bei Nutzung aller Maßnahmen
  - Wiederholung Anfrage und Ergebnis; Duplikatsfilterung
  - Aufrufer erhält Ergebnis ⇒ Methode wurde genau einmal aufgerufen
  - Aufrufer erhält Fehlermeldung/Exception ⇒ Methode wurde höchstens einmal aufgerufen
  - z.B. bei CORBA, Java RMI

Aufgaben/Ziele

- Deklaration von Schnittstellen (von Objekten): Methodensignaturen, Datenstrukturen
- Generierung von Stubs/Skeletons aus der Schnittstellenbeschreibung
- Gemeinsame Syntax, unabhängig von Programmiersprache, BS etc
- Beispiele: CORBA-IDL[a], WSDL[b]
- IDL-Compiler: erzeugt aus Schnittstellenbeschreibung Stub- und Skeleton-Quellcode in einer gegebenen Programmiersprache

[a] IDL: Interface Definition Language
[b] Web Service Definition Language

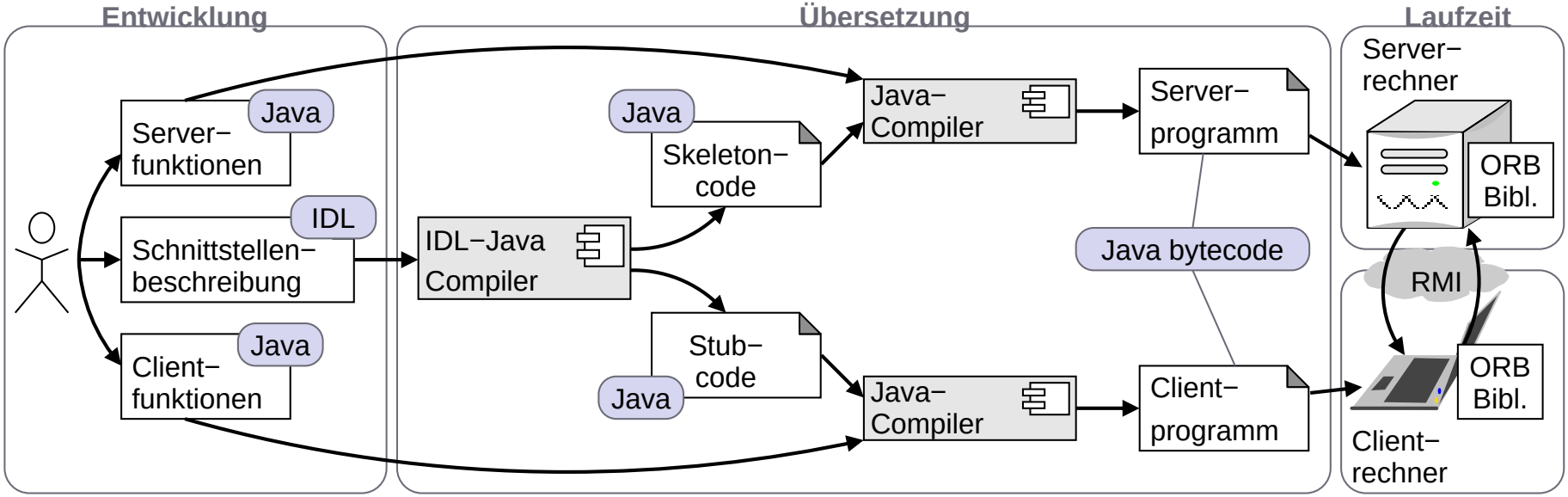
Beispiel: IDL-Deklaration

```
struct Rectangle {
  long width;
  long height;
  long x;
  long y;
}
interface Shape {
  Rectangle getEnclosing ( );
  unsigned long getArea ( );
}
```

Gegeben seien eine Programmiersprache (P) und eine Schnittstellenbeschreibungssprache (IDL).

1. Programmierer schreibt Schnittstellenbeschreibung in IDL-Sprache.
2. IDL-Compiler erzeugt Quellcode in P, enstprechend der Schnittstellenbeschreibung.
3. Programmierer implementiert die Schnittstelle in P.
4. P-Compiler übersetzt generierten sowie geschriebenen Quellcode.
5. Binder/Linker verbindet Objektcode mit Middleware-Bibliotheken (z.B. mit CORBA-ORB)

Beispiel: Erstellung einer verteilten Java/CORBA-Anwendung

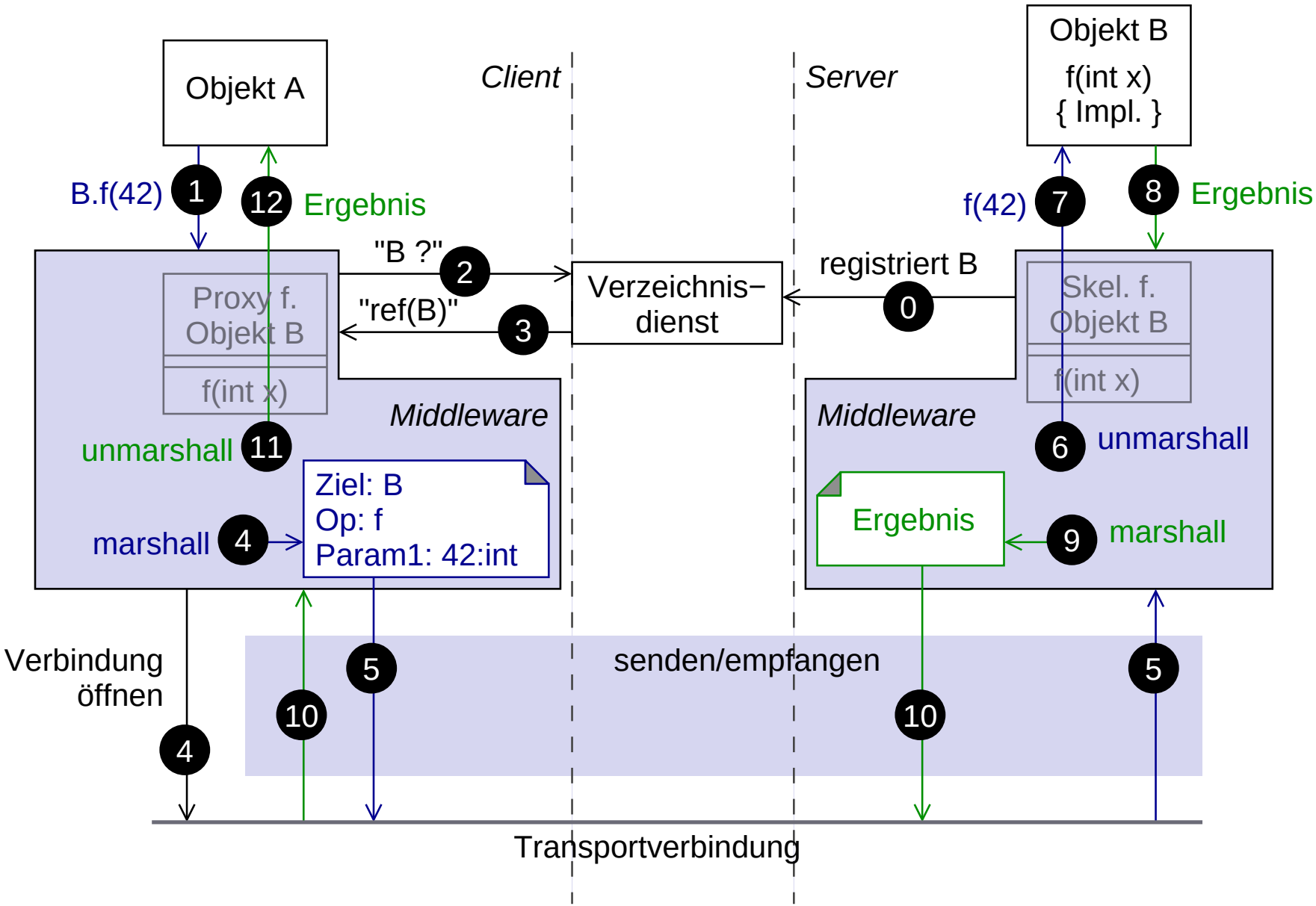


Durchführung eines Aufrufs

- Auflösung von Namen (mittels zentralem Verzeichnisdienst oder lokaler Referenzverwaltung)
- Verbindungsmanagement (z.B. *connection pool*)
- Marshalling/Unmarshalling (Ein-/Auspacken von Aufrufen und Ergebnissen)
- Fehlerbehandlung (z.B. bei verlorenen Tranport-PDUs)
- Austausch zwischen Stub/Skeleton und Nutzercode (Annahme von Aufrufen, Rückgabe von Ergebnissen/Exceptions)

Weitere Funktionen

- Interface Repository (IR)
  - Ablage für Schnittstellenspezifikationen
  - Zugriff durch Clients zur Laufzeit


Objekt A
Client
Server
Objekt B
f(int x)
{ Impl. }
B.f(42)
1
12
Ergebnis
f(42)
7
8
Ergebnis
"B ?"
2
"ref(B)"
3
Verzeichnis−
dienst
registriert B
0
Proxy f.
Objekt B
f(int x)
Skel. f.
Objekt B
f(int x)
Middleware
Middleware
unmarshall
11
6
unmarshall
Ziel: B
Op: f
Param1: 42:int
marshall
4
Ergebnis
9
marshall
Verbindung
öffnen
4
5
10
senden/empfangen
10
5
Transportverbindung

Rechnernetze und verteilte Systeme
Middleware

Protokolle

Kapitel 9.3

Middleware ist Nutzerin der Transportschicht: IP/TCP/GIOP
kann auch höher angesiedelt werden: IP/TCP/HTTP/SOAP

Beispiel: General Inter-ORB Protocol (GIOP)

- Nutzung: CORBA-ORBs, Java RMI
- Anpassung für Betrieb über TCP: IIOP = Internet Inter-ORB Protocol
- Asymmetrische Nutzung einer Verbindung, entsprechend Client-/Server-Rollen
- Spezifiziert in Standards der Object Management Group (OMG)

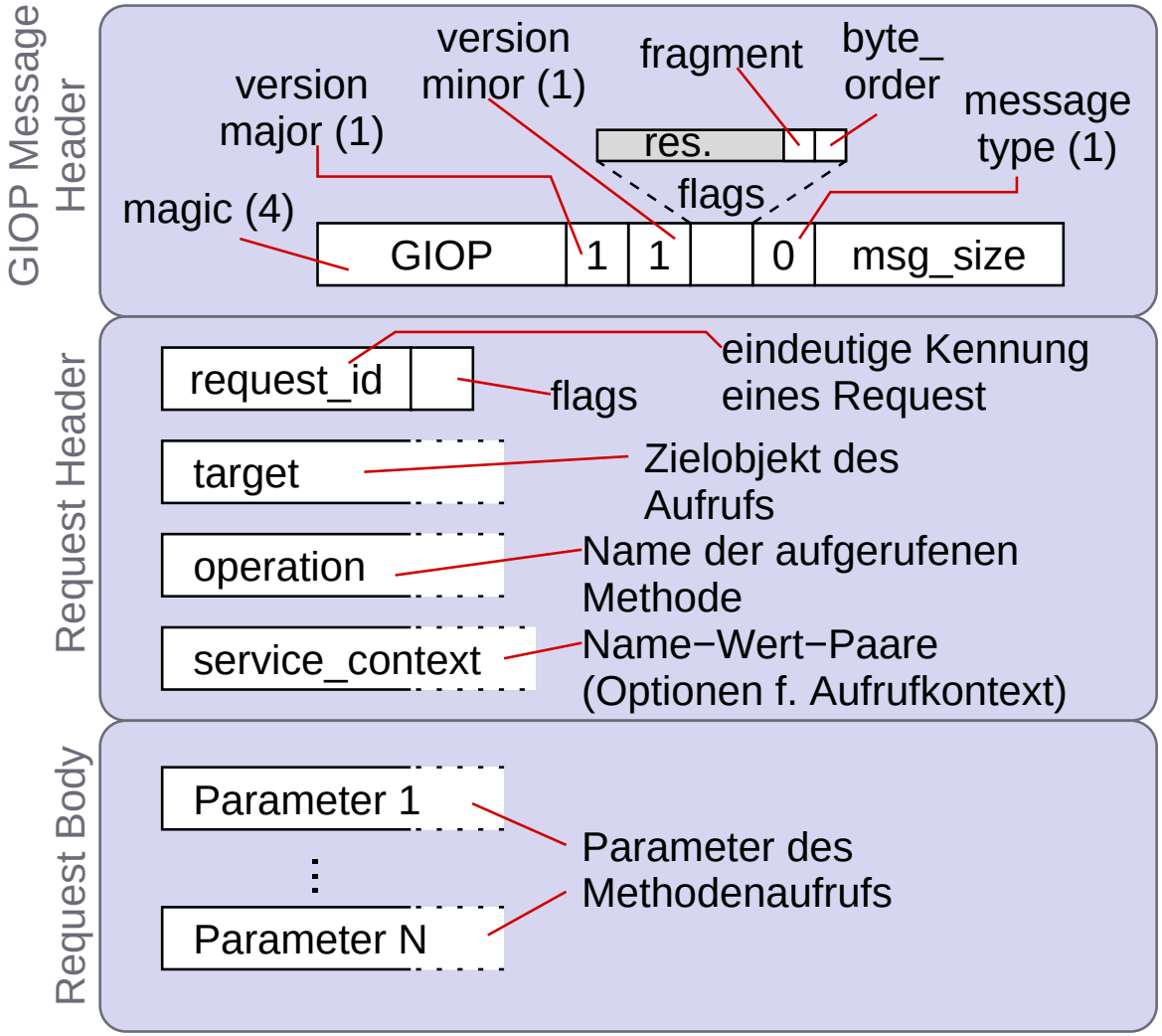
Beispiel: Simple Object Access Protocol (SOAP)

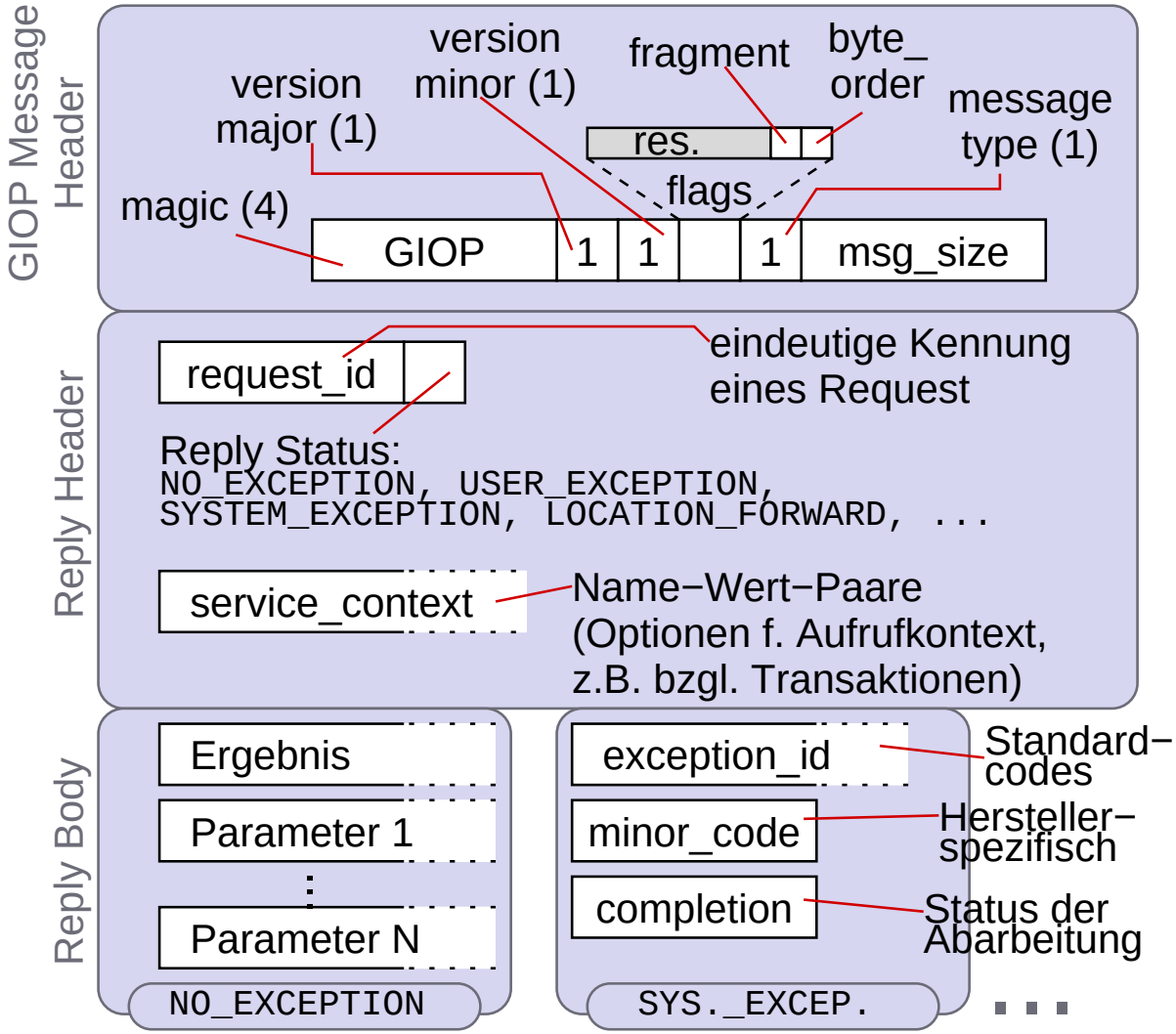
- Protokoll zum Austausch von XML-basierten Nachrichten und Dokumenten
- Im Prinzip unabhängig von Transportprotokoll
  oft: Einbettung in HTTP-Nachrichten
- Standardisiert durch das WWW Consortium (W3C)

- Multiplex von Anfragen/Aufrufen an ein Objekt (*request multiplexing*)
- Keine (auch keine relative) Ordnung bzw. Reihenfolge der Anfragen
- Syntax folgt dem Common Data Representation (CDR) Format

PDU-Typen von GIOP

Request	Methodenaufruf an CORBA-Objekt, u.U. fragmentierbar
Reply	Ergebnis eines Request, u.U. fragmentierbar
LocateRequest	Anfrage, ob bestimmte Methode bzgl. eines Objekts aufgerufen werden kann
LocateReply	Antwort auf LocateRequest
CancelRequest	Hinweis, dass Anwort nicht mehr erforderlich ist
CloseConnection	Server wird keine weiteren Requests verarbeiten und schließt Verb.
MessageError	Antwort auf falsch geformte Anfrage
Fragment	nachfolgend einer Request- bzw. Response-PDU mit Fragment-Bit=1


GIOP Message Header
version major (1)
version minor (1)
fragment
byte_ order
message type (1)
res.
flags
magic (4)
GIOP
1
1
0
msg_size
Request Header
request_id
flags
eindeutige Kennung eines Request
target
Zielobjekt des Aufrufs
operation
Name der aufgerufenen Methode
service_context
Name−Wert−Paare (Optionen f. Aufrufkontext)
Request Body
Parameter 1
Parameter N
Parameter des Methodenaufrufs


GIOP Message Header
version major (1)
version minor (1)
fragment
byte_ order
message type (1)
res.
flags
magic (4)
GIOP
1
1
1
msg_size
Reply Header
request_id
eindeutige Kennung eines Request
Reply Status:
NO_EXCEPTION, USER_EXCEPTION, SYSTEM_EXCEPTION, LOCATION_FORWARD, ...
service_context
Name−Wert−Paare (Optionen f. Aufrufkontext, z.B. bzgl. Transaktionen)
Reply Body
Ergebnis
Parameter 1
Parameter N
NO_EXCEPTION
exception_id
Standard−codes
minor_code
Hersteller−spezifisch
completion
Status der Abarbeitung
SYS._EXCEP.
. . .

Rechnernetze und verteilte Systeme
Middleware

Datenrepräsentation

Kapitel 9.4

- Verteilte Anwendung: Austausch von Nachrichten zur Erbringung eines Dienstes zugunsten eines Nutzer
- Netze und verteilte Systeme: zusammengesetzt aus heterogenen Hard- und Softwarekomponenten
  - Maschine: Bitreihenfolge, Wortbreite
    - Beispiel: 32-Bit bzw. 64-Bit Maschinen, Little bzw. Big-Endian
  - Übersetzer: verschiedene Datentypen (primitiv/komplex)
    - Beispiel ganze Zahl (Integer): Anzahl Bits? Vorzeichen?
  - Zeichenkodierung (z.B. ASCII, Universal Character Set (UCS), UTF[1]-8, UTF-16, ISO-8859-x ...
- Ziel: Verschattung der Heterogenität →keine Anpassung der Bestandteile einer Anwendung erforderlich
- Ziel: Abstraktion von den Kommunikationsvorgängen →Verschattung des Nachrichtenaustausches: Anwendung wirkt monolithisch, „aus einem Stück“

[1] (UTF = UCS Transformation Format)

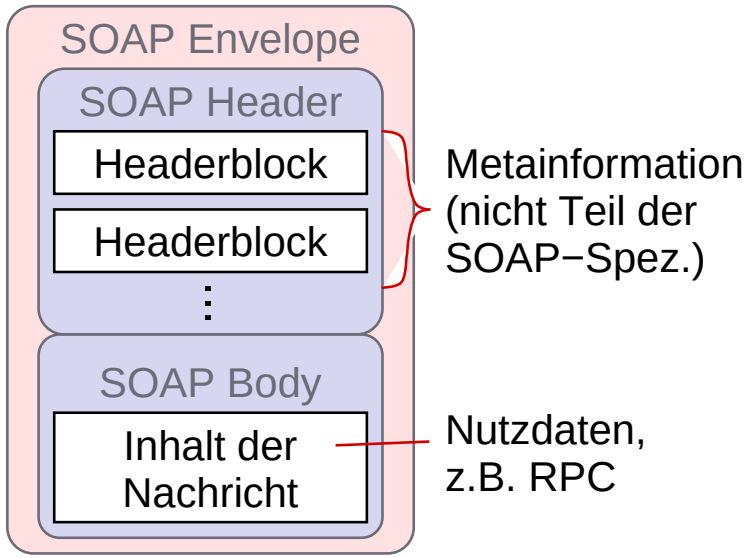
Common Data Representation, CDR

- 15 Primitivtypen: Zahlentypen, boolscher Typ, any (allgemeiner Platzhalter, kann jeden primitiven oder zusammengesetzten Typen repräsentieren)
- zusammengesetzte Typen (constructed type)
  - sequence (Länge, Elemente)
  - string (Länge, Zeichen)
  - array (Elemente in Reihenfolge), feste Länge
  - struct (Elemente in Reihenfolge ihrer Deklaration), wie C-struct
  - enumerated = unsigned long
  - union (Typ, Element)

Beispiele: Abbildung von CDR-Datentypen in Programmiersprachen

IDL-Typ	C++	Java
`long`	`CORBA::long`	`int`
`float`	`CORBA::float`	`float`
`string`	`char*`	`java.lang.String`
`sequence<` *Type* `>`	Mapping-Klasse für *Type*	*Type*`[]`

- XML-basierte Kodierung der Daten
- keine weiteren (Typen-)Vorgaben für Datenrepräsentation
- *envelope* („Umschlag”): fasst mehrere Nachrichtenköpfe und -rümpfe (header/body) zusammen
- Austausch von Envelopes entspricht einem Request-Responst-Protokoll



Beipiel: Header mit einzelnem Headerblock

```
<env:Header    xmlns:env="http://www.w3.org/2003/05/soap−envelope" >
<t:Transaction xmlns:t="http://example.org/2001/06/tx"
                    env:mustUnderstand="true" >
 5
</t:Transaction>
</env:Header>
```