

Kommunikation und Datenhaltung

11. Middleware



Prof. Dr. Martina Zitterbart
Dipl.-Inform. Martin Röhrich
[zit | roehricht]@tm.uka.de



1. Einführung
2. Physikalische Grundlagen
3. Protokollmechanismen
4. Geschichtete Architekturen
5. Sicherungsschicht: HDLC
6. Beschreibungsmethoden
7. Sicherungsschicht:
Lokale Netze
8. Netzkopplung und Vermittlung
9. Die Transportschicht
10. Anwendungssysteme
11. Middleware

11.1 Aufgaben und Dienste
11.2 Überwindung von Verteilung
11.3 Überwindung von
Heterogenität

- **Betriebssystem**

- Softwaresystem, das anwendungsnahe Dienste erbringt
 - ▶ Abstraktion von der unterliegenden Hardware
 - ▶ Schnittstelle zwischen Anwendung und Hardware
 - Ursprünglich lokal auf dem Rechner
 - Steuerung der Vergabe von Betriebsmitteln, Prozessorzuteilung etc.
- ⇒ **Verteilte Betriebssysteme:** nicht mehr auf einen Rechner fokussiert

- **Middleware**

- Softwaresystem, das Dienste zur Unterstützung verteilter Anwendungen erbringt
 - ▶ Sitzt „oberhalb“ des Betriebssystems
- Rechnerübergreifend, also verteilt
- Verbirgt Heterogenität von Datenübertragung, Betriebssystemen und Programmiersprachen vor den Dienstnehmern
- Programmiermodell für verteilte Applikationen

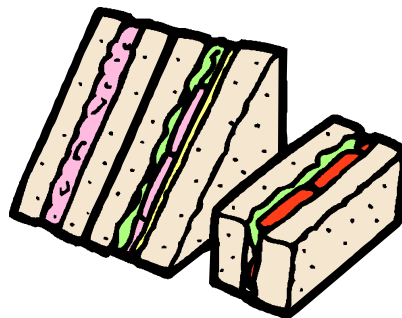
⇒ Im Folgenden wird Middleware als „Kitt und Mörtel“ zwischen Dienstnehmer und Dienstgeber betrachtet

⇒ Lokale Betriebssysteme sind ebenfalls vorhanden

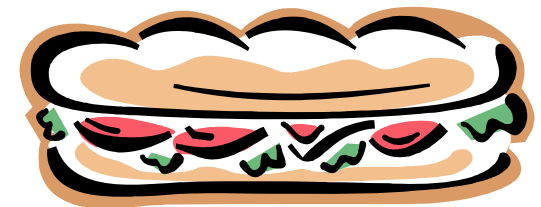


⇒ „Middleware is all about integration“ – „*Middleware is everywhere*“

- Aufgabe
 - Zusammenführung von Dienstnehmern und Dienstgebern
 - ▶ Kommunikation
 - ▶ Verteilte Transaktionsverarbeitung
 - ▶ ...
- Ziele
 - Skalierbarkeit
 - Offenheit
 - Dienstekopplung
- Die Middleware macht den Unterschied aus ...
 - Was wäre ein Sandwich ohne was „dazwischen“?



Tomate?
Salat?
Mayonnaise?
... ?



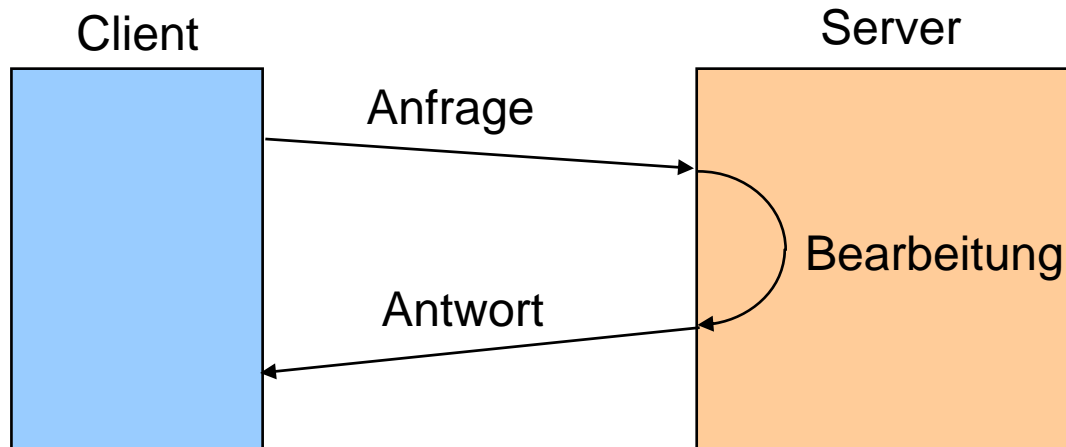
- Überwindung von **Verteilung** (Ortstransparenz)
 - Dienstnehmer und Dienstgeber müssen nicht im selben Adressraum ablaufen
- Überwindung von **Heterogenität**
 - Technische und semantische Interoperabilität
- **Vermittlung** von Dienstgebern
 - Bevor Dienstgeber in Anspruch genommen werden kann, muss dieser „gefunden“ werden
- Dienstaktivierung und -beendigung
 - Wann wird Dienst aktiviert?
- Lastverteilung
 - Evtl. Verteilung der Anfragen auf unterschiedliche Rechner
- Sicherheit
 - Herstellen gegenseitigen Vertrauens
- Persistenz
 - Dauerhaftigkeit für zustandsbasierte Dienste
- Transaktionen
 - Konfliktserialisierbarkeit

- Flexibilität
 - Hohe Leistungsfähigkeit
 - Hooks für „Power-User“
 - Konfiguration
 - Vielzahl an Möglichkeiten
 - Übergreifendes Ziel
 - Integration von Anwendungen
 - ▶ Über Intranets hinausgehend
- ⇒ **Web Services** stellen eine neue wichtige Entwicklung in dieser Richtung dar

} Gegensätzlich!

- Verteilte Systeme
 - Dienstnehmer und Dienstgeber laufen nicht unbedingt im selben Adressraum
 - ▶ Entfernter Dienstgeber
 - ▶ Entfernter Funktionsaufruf (*Remote Procedure Call*, RPC)
 - ▶ ...
 - „Milderung“ von Verteilungsaspekten
 - Annäherung des entfernten Programmiermodells an das lokale
 - ⇒ Für den Client ist der Aufenthaltsort des Servers verdeckt
 - Aber: Transparenz ist zwangsläufig unvollkommen
 - Abweichungen bei der Parameterübergabe/-rückgabe
 - ▶ Insbesondere bei Zeigerparametern und verzeigerten Strukturen
 - Komplexeres Fehlermodell
 - Auswirkungen auf die Leistung (Verzögerung)
- ⇒ Programmierer muss sich Zugriffen auf potenziell entfernte Dienste bewusst sein!

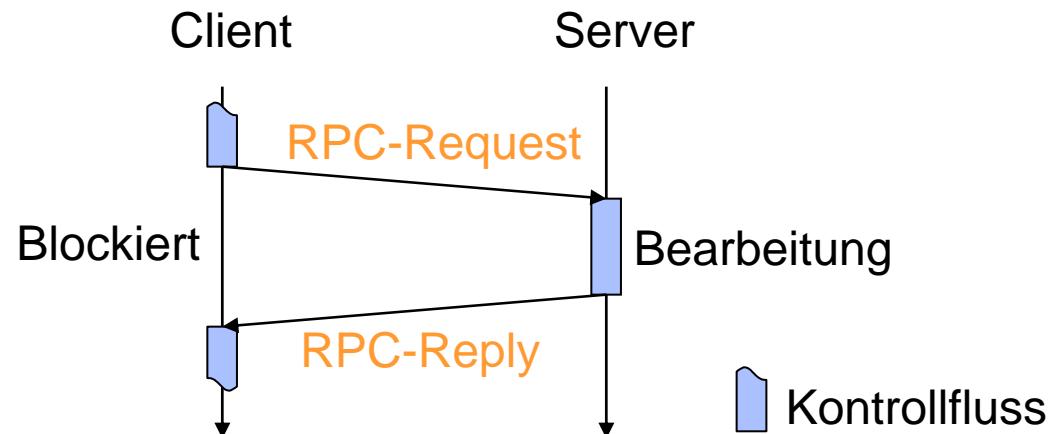
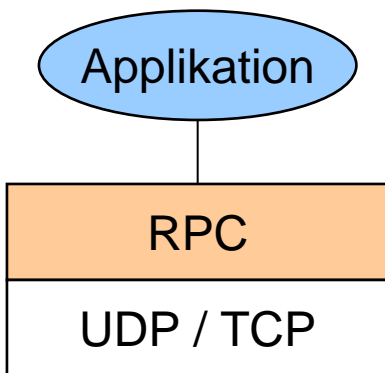
- Grundkonzept
 - Server stellen Dienste zur Verfügung
 - Clients benutzen diese Dienste, indem sie Anfragen an die Server senden
 - Server liefern Ergebnisse der Anfrage an Client zurück
- Unterschiedliche Rollen von Client und Server (Asymmetrie)
- Weiterleitung von Anfragen möglich, Server wird dann zum Client



- Im Java-Umfeld
 - *Remote Method Invocation (RMI)*
 - ▶ Client-Objekt ruft Methode auf, die ein Server-Objekt auf einem entfernten Rechner bereitstellt

- Kommunikationsmechanismus für Client/Server-Anwendungen
 - Ermöglicht das Aufrufen von Prozeduren auf entfernten Rechnern
 - ▶ Programmiersprachliche Einbettung
 - Für den Programmierer bleibt der Datenaustausch transparent
- RPC ist im Schichtenmodell über UDP oder TCP angesiedelt
 - In der Realisierung meist Bestandteil der Anwendung selbst
- Schema

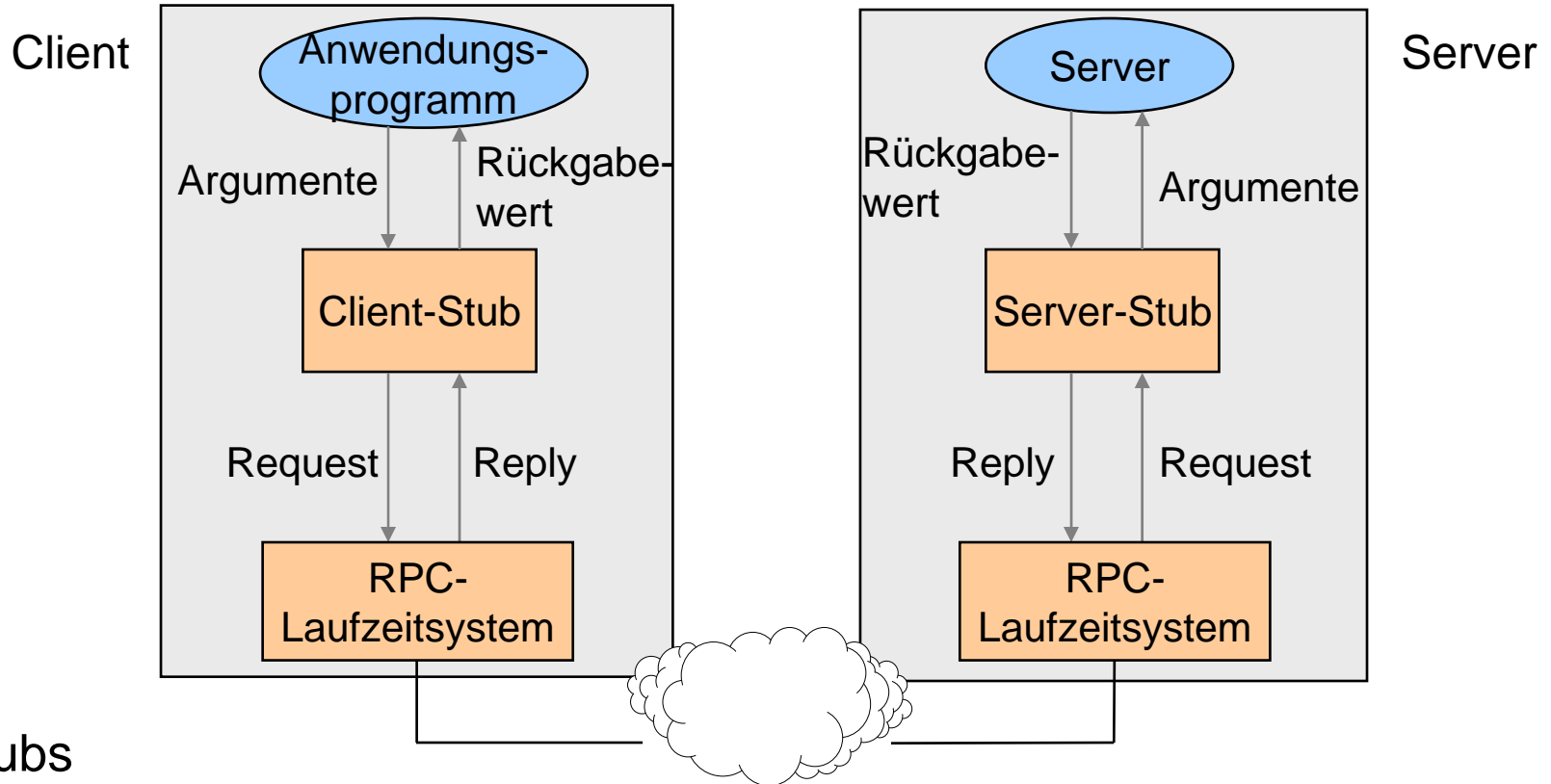
Ablauf



- Erweiterung des Prozeduraufrufs zum **entfernten Prozeduraufruf**
- Definition (nach Nelson und Birell, 1984)
 - Synchroner Übergabe des Kontrollflusses
 - ▶ Aufrufender Client wird blockiert
 - Ebene der Programmiersprache
 - Getrennte Adressräume
 - ▶ Keine netzweit eindeutigen Adressen (Zeiger!)
 - Kopplung über schmalen Kanal
 - ▶ Weniger leistungsfähig als lokale Interaktionsmechanismen
 - ▶ Achtung: Fehler möglich, die im lokalen System nicht auftreten
 - Datenaustausch: Aufrufparameter und Ergebnisse
 - ▶ Achtung: unterschiedliche Darstellung von Daten auf heterogenen Maschinen!
- Ablauf
 - Aufruf im Wartezustand
 - Parameter- und Aufrufübertragung ins Zielsystem
 - Prozedurausführung
 - Rückmeldung
 - Fortsetzung der Programmausführung

- Bei der Verwendung entfernter Prozeduraufrufe können eine Reihe von Fehlern auftreten
 - Anfragen oder Antworten können bei der Datenübertragung verloren bzw. verfälscht werden
 - Client oder Server können im Verlauf eines RPCs (unabhängig voneinander) abstürzen
- Bzgl. der Fehlerbehandlung werden folgende Fehlersemantik-Klassen unterschieden

Fehlersemantik	Anfrage einer Wiederholung	Filterung Duplikate	Wiederausführung/ Wiederholung Antw.
Maybe	Nein	Nein	Nein
At-least-once	Ja	Nein	Wiederausführung
At-most-once	Ja	Ja	Wiederholung Antw.
Exactly-once	Ja	Ja	Nein



- Stubs
 - Aufrufkodierung / Ergebnisdekodierung
 - ▶ Transformation der Daten in eine flache Repräsentation
 - ▶ Nutzt Spezifikation der Prozedurschnittstelle
- RPC-Laufzeitsystem
 - Aufrufübertragung / Ergebnis- bzw. Aufrufempfang / Ergebnisübertragung

- Anwendungsprogramm ruft entfernte Prozedur auf (die der Client-Stub zur Verfügung stellt) und wird in den Ruhezustand versetzt
- Aufrufkodierung durch den Client-Stub (**Marshalling**)
- Senden der erzeugten Nachrichten an das entfernte System mit Hilfe des RPC-Laufzeitsystems
- RPC-Laufzeitsystem nutzt Transportprotokoll zum Senden der Nachrichten
- RPC-Laufzeitsystem leitet empfangene Nachrichten an Server-Stub weiter
- Server-Stub dekodiert die Nachrichten (**Unmarshalling**)
- ...

- ...
- Server-Stub führt lokalen Prozeduraufruf durch
 - Übergibt Parameter an tatsächliche Server-Prozedur
- Server-Prozedur übergibt nach Bearbeitung Ergebnis an Server-Stub
- Server-Stub kodiert das Ergebnis
- Server-Stub sendet mittels RPC-Laufzeitsystem die erzeugte Ergebnismeldung an Client
- RPC-Laufzeitsystem leitet Daten an Client-Stub weiter
- Client-Stub dekodiert Daten, aktiviert das Anwendungsprogramm und reicht Ergebnis zum Anwendungsprogramm weiter

- Für den entfernten Prozeduraufruf wird eine einheitliche Beschreibung der Prozeduren und Parameter benötigt
 - Schnittstellenbeschreibungssprachen
 - ▶ Formale Sprache zur Beschreibung von Prozeduren und Parametern
 - ▶ Abstrahieren von unterschiedlichen Programmiersprachen (ähnlicher Art)
 - ▶ Dienen ausschließlich zur Schnittstellendefinition
 - ⇒ Strikte Trennung von Schnittstelle und Implementierung
 - Auf der Basis der Schnittstellenbeschreibungssprache können Stubs automatisch erzeugt werden
- Beispiele
 - ASN.1 (Abstract Syntax Notation One)
 - XDR (eXternal Data Representation)
 - IDL (Interface Definition Language)

- Voraussetzung für Inanspruchnahme eines Dienstes
 - Einigkeit über Dienstschnittstelle und Protokoll
- Probleme in verteilten Systemen können entstehen durch
 - unterschiedliche Programmiersprachen
 - unterschiedliche Typsysteme und Aufrufkonventionen
- Ziel
 - Reduzierung der Vielfalt
 - ▶ Nicht für jede Kombination aus unterschiedlichen Komponenten soll ein eigenes Verfahren definiert werden ⇒ Komplexität

- Unterschiedliche Wertebereiche
 - z.B. Typ `long`
 - ▶ 4 Bytes auf i386, PowerPC, Sparc
 - ▶ 8 Bytes auf Alpha, IA64, Sparc64, x86_64
- Speicherausrichtung
 - Byte-Padding von Compiler
 - ▶ GCC ermöglicht bspw. Angabe von `__attribute__((packed))`
- Nicht unterstützte Datentypen
 - Bsp.: Typ `unsigned int` unter Java nicht vorhanden
- Byteorder: Little oder Big Endian
 - Bsp.: Datei `test.c`
 - ▶

```
struct foo {
    char s[6];
    int i;
};

struct foo bar = {"FooBar", 0x12345678};
```
 - Objektcode erzeugen mittels
 - ▶ `gcc -c -o test.o test.c`
 - Assembler-Ausgabe erzeugen mittels
 - ▶ `gcc -S -o test.s test.c`

```
$ hexdump -C test.o
00000000 fe ed fa ce 00 00 00 12 00 00 00 00 00 00 00 01 |.....|
00000010 00 00 00 03 00 00 01 6c 00 00 20 00 00 00 00 01 |.....l.....|
00000020 00 00 01 04 00 00 00 00 00 00 00 00 00 00 00 00 |.....|
00000030 00 00 00 00 00 00 00 00 00 00 0c 00 00 01 88 00 |.....|
00000040 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |.....|
00000050 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |.....|
00000060 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |.....|
00000070 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |.....|
00000080 00 00 00 00 00 00 00 00 46 6f 6f 42 61 72 00 00 |.....FooBar..|
00000090 12 34 56 78 00 00 00 00 |.4Vx|
000000a0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |.....|
000000b0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |.....|
000000c0 00 00 00 00 00 00 00 00 01 00 00 00 00 00 00 00 |.....|
000000d0
```

- test.s auf PowerPC (Big Endian)

```
.section __TEXT,__text,[...]
.section __TEXT,[...]
.machine ppc
.globl _bar
.data
.align 2_
bar:
.ascii "FooBar"
.space 2
.long 305419896
.subsections_via_symbols
```

```
$ hexdump -C test.o
00000000 7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00 |.ELF.....|
00000010 01 00 03 00 01 00 00 00 00 00 00 00 00 00 00 00 |.....|
00000020 b8 00 00 00 00 00 00 00 34 00 00 00 00 00 28 00 |.....4.....(|
00000030 07 00 06 00 46 6f 6f 42 61 72 00 00 78 56 34 12 |...FooBar..xV4.|
00000040 00 47 43 43 3a 20 28 47 4e 55 29 20 34 2e 31 2e |.GCC: (GNU) 4.1.|
00000050 33 20 32 30 30 37 30 39 32 39 20 28 70 72 65 72 |3 20070929 (pre|
00000060 00 00 00 00 00 00 00 00 2f 56 62 6e 77 77 00 00 |.lease) (.....|
00000070 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |.....|
00000080 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |.....|
00000090 00 00 00 00 00 00 00 00 01 00 00 00 00 00 00 00 |.....|
000000a0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |.....|
000000b0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |.....|
000000c0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |.....|
000000d0
```

- test.s auf Intel x86 (Little Endian)

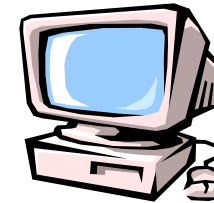
```
.file "test.c"
.globl bar
.data
.align 4
.type bar, @object
.size bar, 12
bar:
.ascii "FooBar"
.zero 2
.long 305419896
.ident "GCC: (GNU) 4.1.3 [...]"
```



System 1

```
TYPE Intf =
  RECORD
    index: INTEGER
    beschr: ARRAY
      [1..20] OF CHAR
    aktiv: BOOLEAN;
  END
```

lokale Darstellung



System 2

```
typedef struct {
  int index;
  char *beschr;
  int aktiv;
} intf;
```

lokale Darstellung

```
WLAN-Interface ::= {
  index      0,
  beschreibung „Intel Wireless Pro“,
  aktiv      true
}
```

Konkretes Beispiel

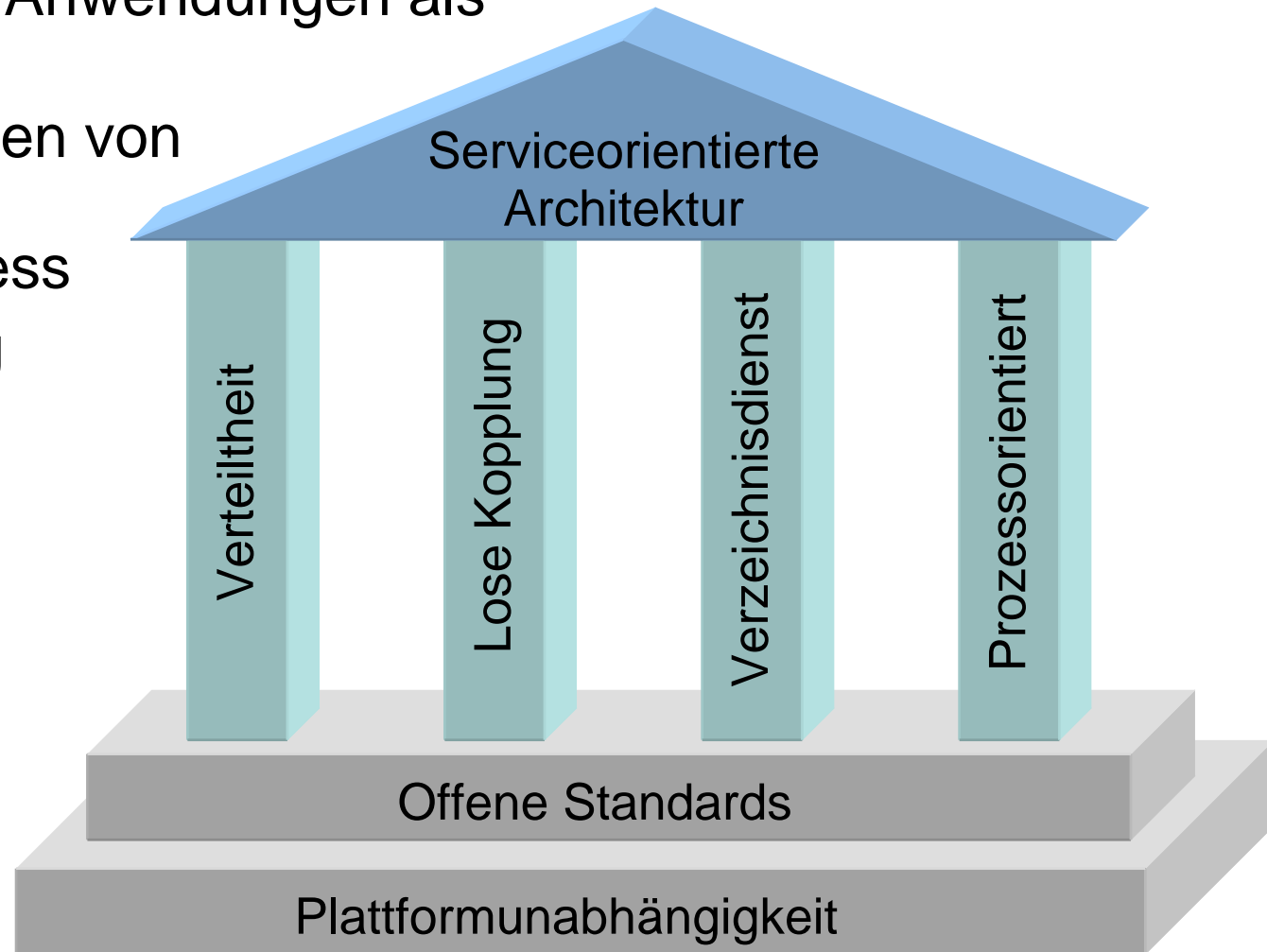
```
Interface ::= SEQUENCE {
  index INTEGER;
  beschreibung IA5STRING;
  aktiv BOOLEAN
}
```

Abstrakte Syntax,
z.B. ASN.1

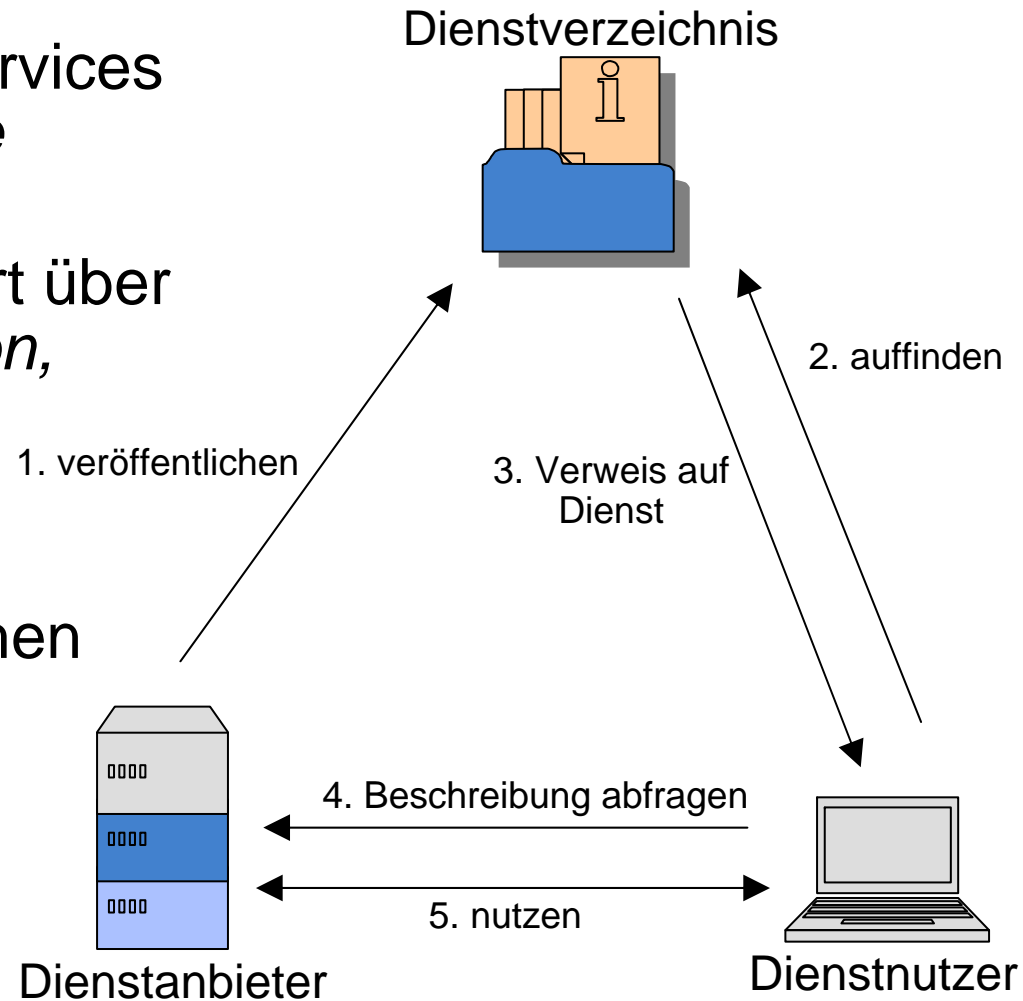
Kodierung/De-Kodierung, z.B. BER

Transfersyntax

- Kapselung von Anwendungen als Dienst
- Zusammenfassen von Diensten als Geschäftsprozess
- Kostensenkung
- Wiederverwendbarkeit
- Auslagern von Diensten

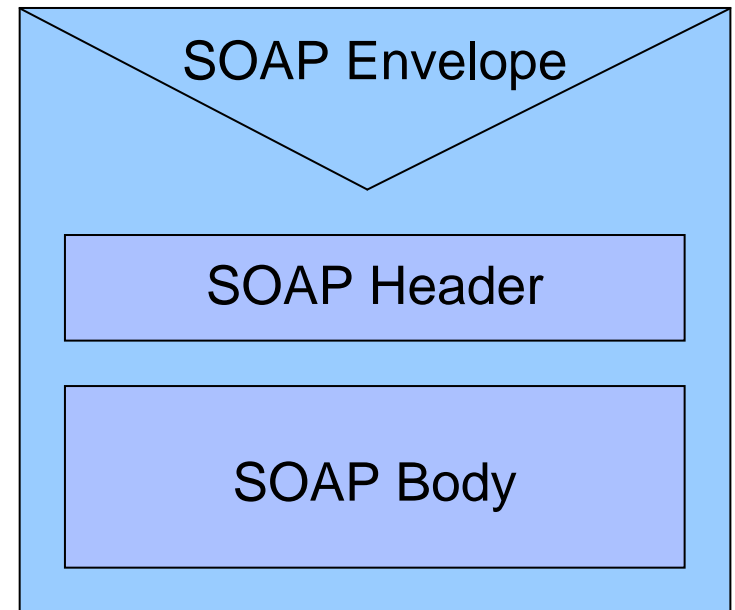


- Beschreibung von Web-Services durch WSDL (*Web-Service Description Language*)
- Verzeichnis Dienst realisiert über UDDI (*Universal Description, Discovery and Integration*)
- Zugriff auf Web-Services und Kommunikation zwischen Dienstanbieter und -nutzer über SOAP



- Datenbank von WSDL-Beschreibungen verschiedener Dienste
- Untergliedert in vier Haupttabellen
 - White Pages
 - ▶ Auffinden eines Dienstes anhand eines Dienstanbieters/Unternehmens
 - Yellow Pages
 - ▶ Dienste nicht nach Anbietern sondern nach Geschäftsfeldern/Branchen geordnet
 - Green Pages
 - ▶ Auflisten von Dienstbeschreibungen wenn weder Anbieter noch Branche bekannt sind
 - Service Type Registration
 - ▶ Green Pages in maschinenlesbar Form

- SOAP (urspr. Simple Object Access Protocol)
- Transport über HTTP und TCP
 - Andere Kombinationen möglich
- XML Dokument bestehend aus 3 Teilen
 - SOAP Envelope
 - ▶ Verwendete SOAP Version
 - ▶ XML Wurzelement
 - SOAP Header
 - ▶ Optional einmalig am Anfang einer Nachricht
 - SOAP Body
 - ▶ Enthält zu übertragende Information

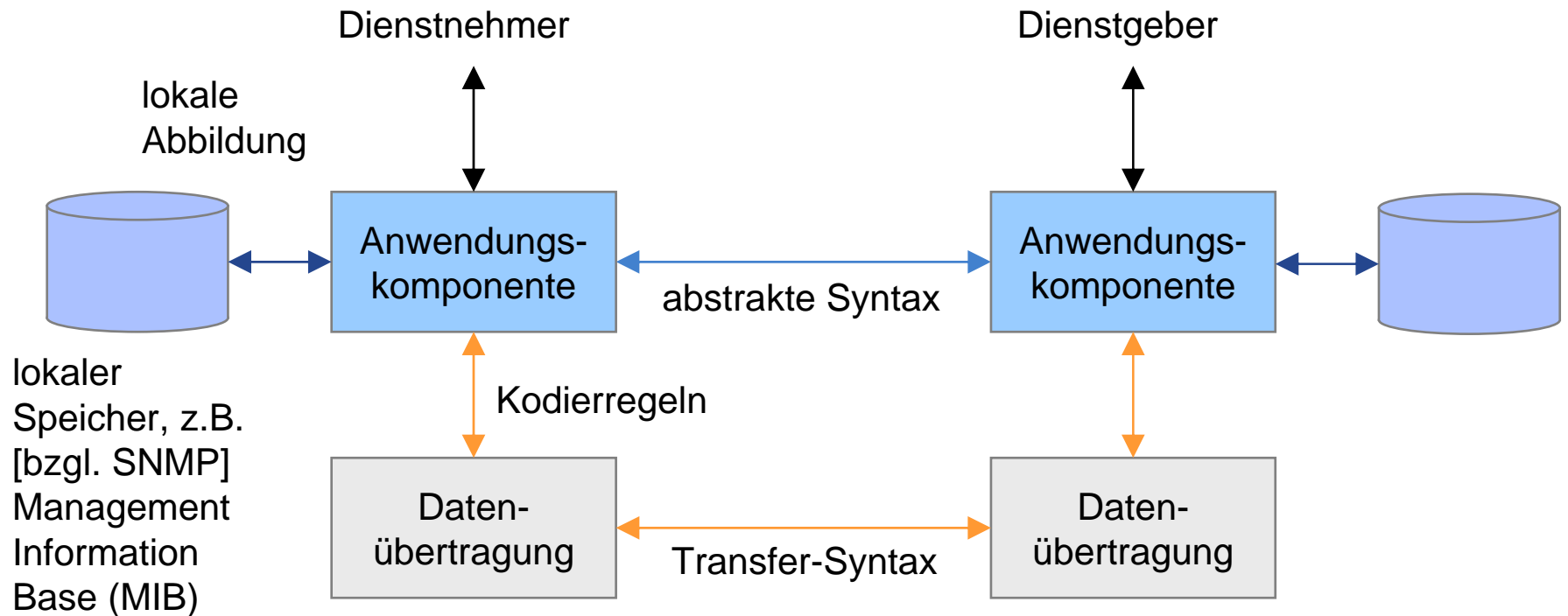


- WSDL dient der Beschreibung von Web-Services
 - XML-basierte Grammatik zur Beschreibung von Diensten als eine Menge von Endpunkten
- Definition über 6 XML Hauptelemente unterteilt in
 - Abstrakte Definitionen
 - ▶ Types
 - ▶ Datenaustauschformate
 - ▶ Messages
 - ▶ PortTypes
 - ▶ One-way, Request-response, Solicit-response, Notification
 - Konkrete Definitionen:
 - ▶ Bindings
 - ▶ Legt verwendetes Protokoll für den Nachrichtenaustausch fest
 - ▶ Endpoints
 - ▶ Exakte Adresse eines Web-Services z.B.: eine URI
 - ▶ Services

- Abstrakte Syntax
 - Menge von Typdefinitionen für Datenobjekte, die in einem Anwendungsprotokoll verwendet werden
 - Sie besagt, was dargestellt ist
 - **ASN.1** ist eine Notation zur Definition einer abstrakten Syntax
 - ▶ Stark an der Programmiersprache C orientiert
 - ▶ Menschliche Lesbarkeit war bei der Entwicklung zweitrangig

- Transfersyntax
 - Konkrete Repräsentation der durch eine abstrakte Syntax beschriebenen Daten
 - Kodierregeln definieren Transformation zwischen abstrakter Syntax und Transfersyntax
 - Für eine abstrakte Syntax können mehrere Transfersyntaxen existieren
 - ▶ Normale Kodierung
 - ▶ Verschlüsselte Kodierung
 - ▶ Komprimierte Kodierung
 - Für ASN.1 sind bisher die **Basic Encoding Rules (BER)** als Kodierregeln standardisiert
 - ▶ Ziel: möglichst kompakte Darstellung, um Übertragungsaufwand zu minimieren

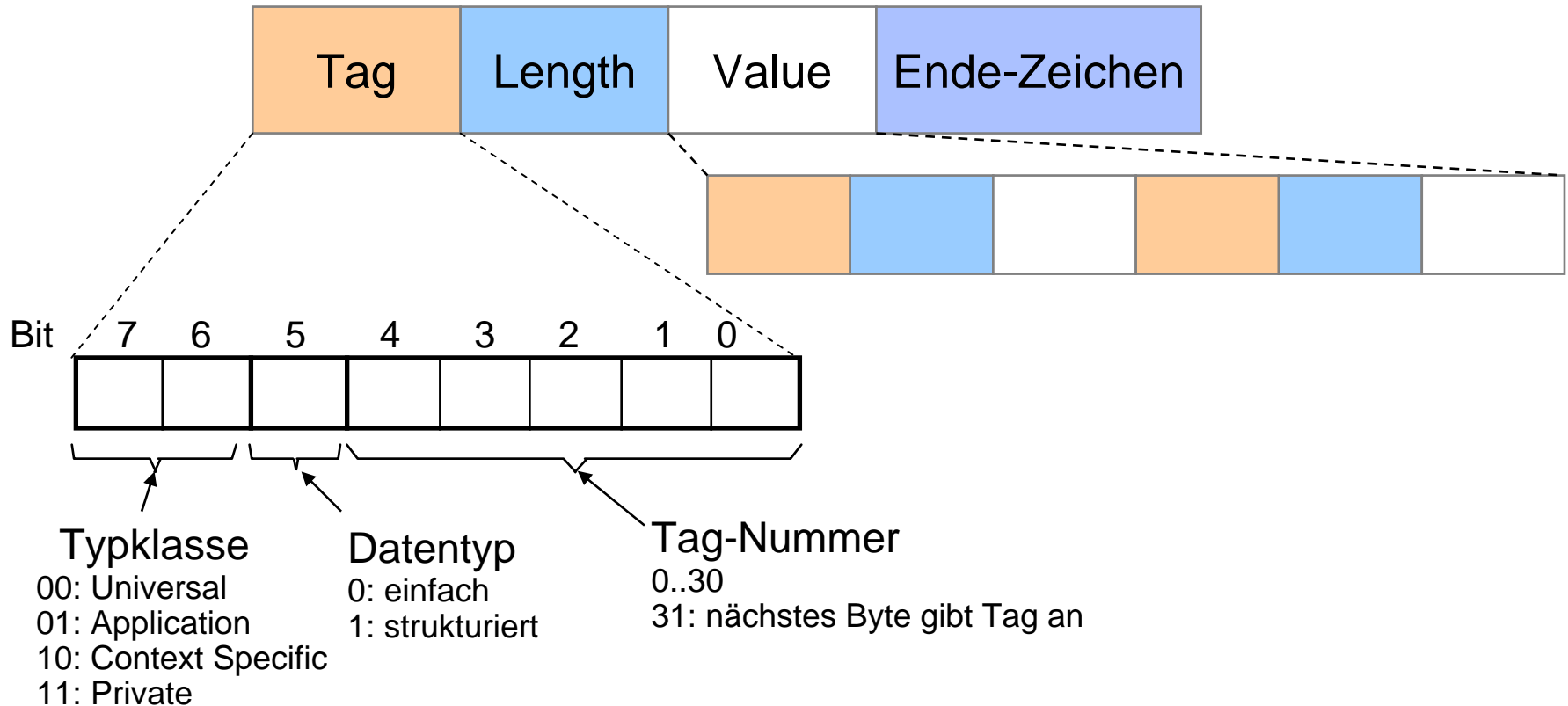
- Schema



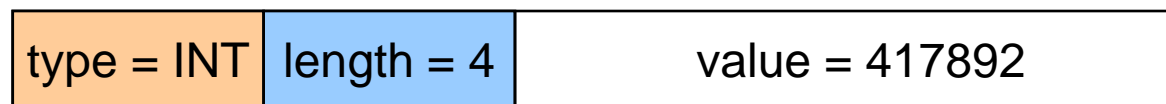
- Abstract Syntax Notation One (ASN.1)
 - Eine Sprache zur Spezifikation der Parameter von Benutzerdaten
 - Erlaubt die Spezifikation von Typen und Werten, ohne die Wertrepräsentation zu bestimmen
 - Angegeben als formale Grammatik in EBNF
- Kodierregeln
 - Spezifizieren die Abbildung der Benutzerdaten in die Form der zu übertragenden Daten
- Basic Encoding Rules (BER) für ASN.1
 - Eine spezielle Menge von Kodierregeln, die international standardisiert sind
- Verwendung unter anderem bei Netzmanagementsystemen (z.B. **SNMP: Simple Network Management Protocol**)
 - Datenrepräsentation via ASN.1
 - Datenaustausch via UDP/IP
- Trend geht bei neueren Protokollen zur Verwendung von XML als Spezifikations- und Beschreibungssprache

- Jede Kodierung eines Datenwertes umfasst gemäß BER (Basic Encoding Rules) die folgenden Teile
 - Tag
 - ▶ Ein oder mehrere Oktette, welche eine Kodierung für Datentyp-Klasse und Tag-Nummer enthalten
 - Länge
 - ▶ Gibt die Länge des Inhalts in Oktetten an oder kennzeichnet, dass ein spezielles Ende-Zeichen vorhanden ist
 - Ende-Zeichen
 - ▶ Ein Ende-Zeichen, bestehend aus zwei 0-Oktetten
 - ▶ Kann vorhanden sein
 - Inhalt
 - ▶ Hier wird der eigentliche Datenwert kodiert. Der Inhalt eines zusammengesetzten Typs wird in naheliegender Weise gebildet, z.B.
 - ▶ SequenceType: Der Dateninhalt ist eine Folge von Datenwerten korrespondierend zu den in der Definition aufgeführten Typen.
 - ▶ SequenceOfType: Der Dateninhalt besteht aus null oder mehreren Datenwerten zu dem angegebenen Typ.
 - ▶ ChoiceType: Der Dateninhalt ist gleich dem Dateninhalt zu dem ausgewählten Typ.
 - Anmerkung: Oktett (engl.: Octet) = Byte mit 8 Bits

- Grundlegender Aufbau



- Beispiel



- Bekannte Länge
 - Kurze Form [1 Byte]
 - ▶ Erstes Bit 0
 - ▶ Die anderen 7 Bits kodieren die Länge als eine binäre Ganzzahl
 - Lange Form [n Bytes, $n > 1$]
 - ▶ Erstes Bit 1
 - ▶ Alle anderen Bits des ersten Bytes kodieren die Anzahl der Längenoktette
 - ▶ Alle anderen Bits aller folgenden Längenoktette kodieren die Länge als eine binäre Ganzzahl

- Unbekannte Länge
 - Kennung [1 Byte]
 - ▶ Erstes Bit 1
 - ▶ Alle anderen Bits 0
 - Wird bei der Kodierung zusammengesetzter Datentypen verwendet, bei denen die Länge nicht sofort verfügbar ist
 - ▶ Im SNMP nicht zugelassen

- BOOLEAN: TRUE

$$\begin{array}{|c|c|c|c|} \hline 00 & 0 & 00001 & \\ \hline \end{array} \quad \begin{array}{|c|c|c|c|} \hline 00000001 & & & \\ \hline \end{array} \quad \begin{array}{|c|c|c|c|} \hline 11111111 & & & \\ \hline \end{array} = 1_{16} \ 1_{16} \ FF_{16}$$

- INTEGER: 100

$$\begin{array}{|c|c|c|c|} \hline 00 & 0 & 00010 & \\ \hline \end{array} \quad \begin{array}{|c|c|c|c|} \hline 00000001 & & & \\ \hline \end{array} \quad \begin{array}{|c|c|c|c|} \hline 01100100 & & & \\ \hline \end{array} = 2_{16} \ 1_{16} \ 100_{10}$$

- INTEGER: 256

$$\begin{array}{|c|c|c|c|} \hline 00 & 0 & 00010 & \\ \hline \end{array} \quad \begin{array}{|c|c|c|c|} \hline 00000010 & & & \\ \hline \end{array} \quad \begin{array}{|c|c|c|c|} \hline 00000001 & & & \\ \hline \end{array} \quad \begin{array}{|c|c|c|c|} \hline 00000000 & & & \\ \hline \end{array} = 2_{16} \ 2_{16} \ 256_{10}$$

- BITSTRING: "0111110111"

$$\begin{array}{|c|c|c|c|} \hline 00 & 0 & 00011 & \\ \hline \end{array} \quad \begin{array}{|c|c|c|c|} \hline 00000010 & & & \\ \hline \end{array} \quad \begin{array}{|c|c|c|c|} \hline 01111101 & & & \\ \hline \end{array} \quad \begin{array}{|c|c|c|c|} \hline 11000000 & & & \\ \hline \end{array} = 3_{16} \ 2_{16} \ 0111110111_2$$

- OCTET STRING: "abcd"

$$\begin{array}{|c|c|c|c|} \hline 00 & 0 & 00100 & \\ \hline \end{array} \quad \begin{array}{|c|c|c|c|} \hline 00000100 & & & \\ \hline \end{array} \quad \begin{array}{|c|c|c|c|} \hline 01100001 & & & \\ \hline \end{array} \quad \begin{array}{|c|c|c|c|} \hline 01100010 & & & \\ \hline \end{array} \quad \begin{array}{|c|c|c|c|} \hline 01100011 & & & \\ \hline \end{array} \quad \begin{array}{|c|c|c|c|} \hline 01100100 & & & \\ \hline \end{array} = 4_{16} \ 4_{16} \ 97_{10} \ 98_{10} \ 99_{10} \ 100_{10}$$

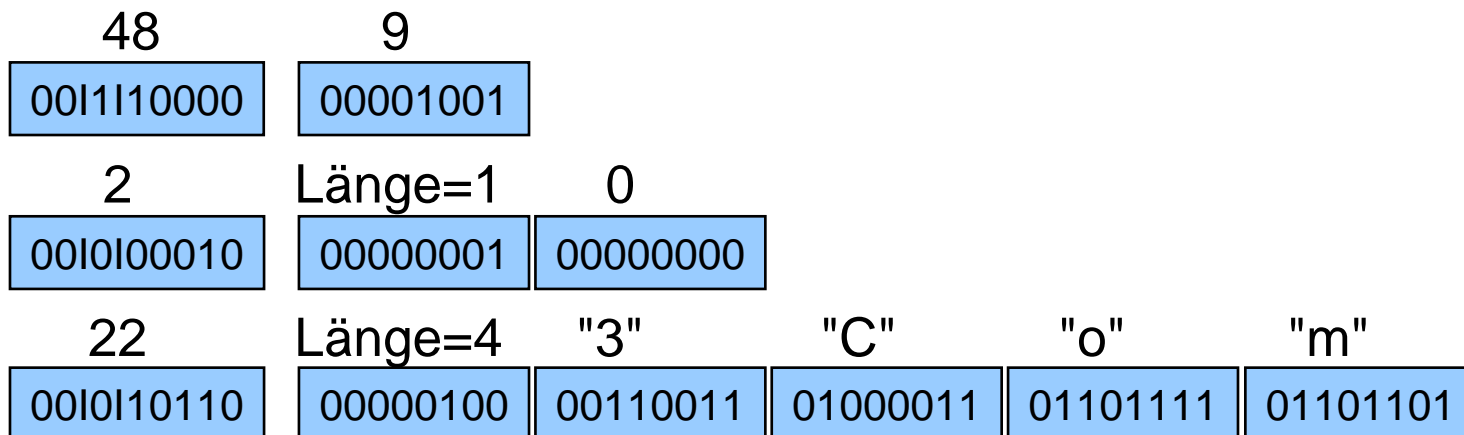
- ASN.1 Sequence

```

interface ::= SEQUENCE {
    index          INTEGER,
    beschreibung  IA5STRING
}

LAN-Interface ::= {
    index          0,
    beschreibung  "3Com"
}
    
```

- BER Kodierung für LAN-Interface



- Middleware als „Vermittler“ zwischen Dienstnehmer und Dienstgeber
 - hat verschiedene Aufgaben
 - mit dem Ziel der Zusammenführung
- Entfernte Prozeduraufrufe (RPC)
 - Ablauf und Problembehandlung (Fehlersemantiken)
- Überwindung von Plattform-Heterogenität
 - Abstrakte Syntax abstrahiert von konkreter Programmiersprache
 - Transfersyntax definiert Regeln zur Übertragung der in abstrakter Syntax angegebenen Daten
 - Kodierregeln zur Transformation zwischen abstrakter Syntax und Transfersyntax

[Vino02] S. Vinoski; Where is Middleware?; IEEE Internet Computing; März/April 2002

[Come08] D. Comer; Computer Networks and Internets with Internet Applications; 5th edition; Prentice-Hall, 2008