7. Notationen für Architekturen

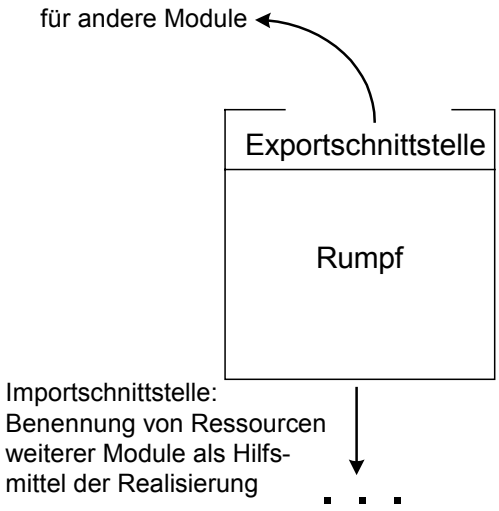
Ziele:
Übersicht integr. Architekturmodellierungssprachen
Jackson-Entwurfsmethode
Historisches HIPO
Details zu erstem Punkt → Architekturmodellierungsvorlesung

Module

Modulcharakteristika

- ”log. Einheit” in Gesamtzusammenhang, ein Satz zur Beschreibung
- M. ist abstr. Maschine oder Hilfsmittel
- M. repr. eine Entwurfsentscheidung, Arch. ist Gesamtheit aller Entwurfsentscheidungen
- M. ist Einheit aus Daten und Operationen
- M. hat ”bestimmte Komplexität” (Spezifikationsseiten, Quelltextseiten)
- stellt Ressourcen nach außen zur Verfügung für andere Module: Exportschnittstelle. Ress. einfach und orthogonal

- Interna (Rumpf) verkapselt
- Bei Implementierung Abstützen auf andere M.e: Importschnittstelle. Verbindung zu and. M.en.
- neben(seiten)effektfrei: Verwendung/Änderung der Parameter d. Exportschnittstelle, Importschnittstelle
- ersetzbar durch M. mit gleicher Exportschnittstelle: keinen Einfluß auf Semantik, aber Pragmatik



- Korrektheit des M.s ohne Kenntnis seiner Verwendung nachweisbar
- ”Korrektheit” der Entwurfsspezifikation ohne Kenntnis der Implementationen der M.e ”nachweisbar” (formale Anforderungsdef.)
- M. unabhängig entwickelbar: Arbeitseinheit
- M. getrennt übersetzbar
- Einheit der Wiederverwendung
- innen enge Kopplung
- Module lose miteinander gekoppelt
- . . .

- Module sind log. Einheiten der Architektur-modellierung
  keine PiK-Einheiten
    Unterprogramme/Makros
    sequentiell, nebenläufig, reentrant
    solche Details erst später
  keine Einheiten der Programmiersprache

- Modul aus versch. Teilen zusammengesetzt und mit anderen in Verbindung
    wird an versch. Stellen verwandt:
      hierfür Exportschnittstelle
    verwendet Ressourcen anderer Module:
      hierfür Importschnittstelle
    Realisierung:
      hierfür Rumpf und Importe

- Unterscheidung (Export)Schnittstelle und Rumpf

Programmieren-im-Großen-
Anteil des Moduls

```
abstract data object module ITEM_STACK is      --***Export-Schnittstelle***    --
   ...                                                                         --
   procedure PUSH (X: in ITEM_TYPE);           --Zugriffsoperationen einer     --
   procedure POP;                              --LIFO-Datenstrukt. beste-      --
   function READ_TOP return ITEM_TYPE;         --hend aus Veraenderungen       --
   function IS_EMPTY return BOOLEAN;           --und Abfragen. Die Bedeu-      --
   function IS_FULL return BOOLEAN;            --tung der einzelnen Opera-     --
   ST_UNDERFLOW, ST_OVERFLOW: exception;       --tionen ist die folgende:      --
   --PUSH legt ein Element auf dem Keller ab. POP loescht das oberste Ele-      --
   --ment. Mit READ_TOP kann das oberste Element abgefragt werden, mit          --
   --IS_EMPTY, ob der Keller leer ist, mit IS_FULL, ob er voll ist.             --
   --ST_UNDERFLOW und ST_OVERFLOW sind Ausnahmen (Fehleranzeigen).              --
   --Die erstere wird erweckt, wenn versucht wird, mit POP das ”oberste Ele-    --
   --ment” eines leeren Kellers zu loeschen, die zweite, wenn mit Hilfe von     --
   --PUSH ein weiteres ELement auf einem vollen Keller abgelegt wird.           --
end ITEM_STACK; ——————————————————————————————————————————————————————————  --
--= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =  --
module body ITEM_STACK is ——————————      Rumpf des Moduls ————————————————  --
   SIZE: INTEGER := 100;                                                       --
   SPACE: array (1..SIZE) of ITEM_TYPE;                                        --
   INDEX: INTEGER range 0..SIZE := 0;                                          --
   procedure PUSH (X: in ITEM_TYPE) is begin ... end;                          --
   ...                                                                         --
   function IS_FULL return BOOLEAN is begin ... end;                           --
   ...                                                                         --
end ITEM_STACK;  -- ************************************************          --
```

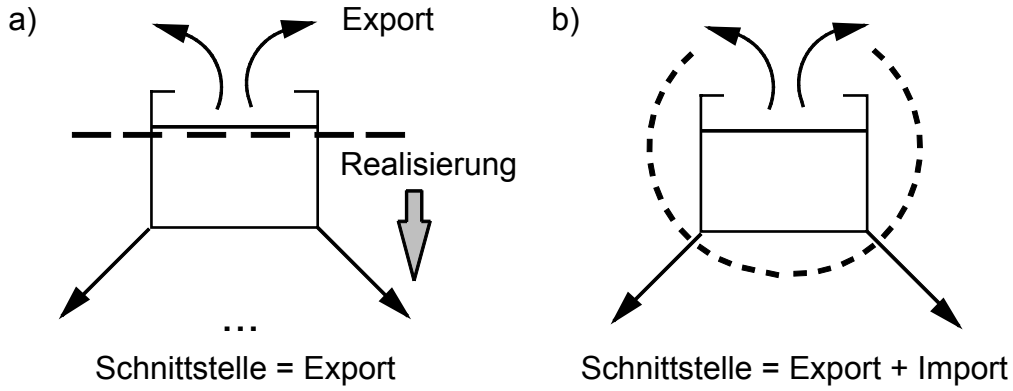
Programmieren-im-Kleinen-
Anteil des Moduls

- Übereinstimmung Schnittstelle - Rumpf

- Schnittstelle sichtbar, Rumpf verborgen

- Sinn des Verbergen des Rumpfs
  Information-Hiding-Prinzip:
    Einhalten des Abstraktionsschritts Rumpf -> Schnittstelle
    Grundlage des Architekturparadigmas
    Verwender will Rumpf i.a. nicht sehen
    Grundlage für Austauschbarkeit und Wiederverwendbarkeit

- Sprachgebrauch ”Schnittstelle”



- Module in Architekturdiagramm: Modulsymbol, Art, Name
  Module in textueller Detailbeschreibung:
  zusätzlich Exportschnittstelle, Importschnittstelle

Modularten und Funktionsmodule

- Abschnitt
  Klassifikation Modularten
  abschließende Einführung funktionaler Module

- Module
  funktionale Abstraktion:
    funktionaler Modul/Funktionsmodul
  Datenabstraktion
    abstr. Datenobjektmodul, abstr. Datentypmodul

- funktionale Abstraktion
  Module haben Transformationsverhalten
    Eingabedaten in Ausgabendaten transformieren
    funktionale Module ”aktionsorientiert”
    E/A-Daten erscheinen in der Schnittstelle
    oft mehr als eine Operation an der Schnittstelle
    kein Gedächtnis
  Formalisierung
    durch Vor- und Nachbedingungen bei jeder Op.

- Datenabstraktion
  Module folgen Datenabstraktions- oder Daten-
  verkapselungsprinzip
    Strukturen haben Gedächtnis, d.h. inneren Zust.
    Datenabstraktionsmodul ”passiv”
  Formalisierung
    algebraische Gleichungen: POP ○ PUSH ≡ ID
    Vor- und Nachbedingungen

- Datenabstraktion aus der Theor. Informatik: formale algebr. Spez.
  hier die softwaretechnische Bedeutung der Datenabstraktion

- beide Abstraktionsprinzipien zur Architektur-modellierung wichtig
  ohne DA: nicht anpaßbar
  ohne funkt. Abstraktion: unübersichtlich
  bei ”purer” OO-Modellierung: i. a. keine funktionale Abstraktion

- Erläuterung kurz
  wegen Vertrautheit mit funkt. Abstraktion
  jedoch mehr als eine Funktion an der Schnittstelle

- Beispiel eines funktionalen Moduls

```
functional module ZEICHNE_FUNKTION is -- ****************************** --
  --Eingabedaten jeweils in der Parameterliste, Ausgabedatum ist das      --
  --erstellte Plotterfile                                                 --
  procedure POLYGON_LIN(X,Y: in FELD; X_TEXT, Y_TEXT, UE_TEXT: --
       in STRING);                                                       --
  procedure INTPOL_LIN(X, Y: in FELD; X_TEXT, Y_TEXT, UE_TEXT:  --
       in STRING);                                                       --
  procedure APPROX_LIN(X, Y: in FELD; X_TEXT, Y_TEXT, UE_TEXT:  --
      in STRING);                                                        --
  -- alle weiteren Prozeduren mit der gleichen Parameterliste            --
  procedure POLYGON_HLOG(...);                                           --
  procedure INTPOL_HLOG(...);                                            --
  procedure APPROX_HLOG(...);                                            --
  procedure POLYGON_DLOG(...);                                           --
  procedure INTPOL_DLOG(...);                                            --
  procedure APPROX_DLOG(...);                                            --
  ...                                                                    --
  -- Angabe der Semantik von ZEICHNE_FUNKTION:                           --
  -- globale Vorbedingung, d.h. Vorbedingung für alle Operationen:       --
  --          Seien X, Y Parameter eines reellen Feldtyps FELD, d. h.    --
  --          Xi,  Yi ∈ REEL, i = 1, .. .,FELDGROESSE                    --
  --          X_TEXT, Y_TEXT, UE_TEXT ∈ STRING, d. h. Zeichenketten      --
  -- globale Nachbedingungen:                                            --
  --          Es erfolgt die Ausgabe mit Hilfe eines Plotters auf ein    --
  --          Zeichenblatt wobei für die Abzissen- und Ordinatenwerte    --
  --          geeignet skaliert wird, und X_TEXT an der X-Achse,         --
  --          Y_TEXT an der Y-Achse und UE_TEXT als Bildueber-           --
  --          schrift erscheint.                                         --
  --          Nachbedingungen für einzelne Operationen:                  --
  -- POLYGON_LIN verbindet die eingegebenen Punkte durch einen           --
  --          Polygonzug und traegt X- und Y-Achse linear auf.           --
  -- INTPOL_LIN verbindet die eingegebenen Punkte durch eine glatte      --
  --          Spline-Kurve und traegt X- und Y-Achse linear auf.         --
  ...                                                                    --
end ZEICHNE_FUNKTION; ------------------------------------------------   --

module body ZEICHNE_FUNKTION is ---------------------------------------  --
  ...                                                                    --
begin                                                                    --
   ...                                                                   --
end ZEICHNE_FUNKTION; --**********************************               --
```

Das Datenabstraktionsprinzip und Datenobjektmodule, Datentypmodule

- Abschnitt:
  Datenabstraktionsprinzip und seine softwaretechn. Bedeutung
  Einführung abstrakter Datenobjektmodule

- Idee der Datenabstraktion
  keine direkten Zugriffe auf Datenstrukturen
  Zugriffsop. mit Datenstruktur unauflösliche Einheit
  ausschließlich über Zugriffsop.: log. Schnittstelle
  Details der Realisierung verborgen
  Realisierung auswechselbar
  Anwendung des Prinzips des Information Hiding

- Name ”Datenabstraktion”, ”abstr. Datenobjekt” etc.
  Wie ”abstrakt” ist ein ado?
  Schnittstelle abstrakter als Realisierung
  Auftreten gehäuft in unt. Teilen einer Architektur
  Abstraktion von einer Vielzahl von Realisierungen

- Sprechweisen
  - Datenabstraktionsprinzip
  - abstraktes Datenobjekt (ado)
  - abstrakter Datenobjektmodul
  - abstrakter Datentyp (adt)
  - abstrakter Datentypmodul (auch Klasse)

- Datenobjektmodul: Beispiel
  Rumpf (PiK) betrachten

  Schnittstelle

```
abstract data object module AUSKUNFTEI is -- ::::::::::::::::                    —
    --Das folgende Lexikon stellt Operationen zum Ablegen (STORE), zum           --
    --Aendern (CHANGE) und zum Auffinden (FIND) von Eintraegen zu                --
    --Personen zur Verfuegung.                                                   --
    --Die Semantik der einzelnen Zugriffsoperationen ist die folgende: ...      --
    procedure FIND (KENNZ: in STRING_K; GES_INFO: out STRING_I);                --
    procedure STORE (KENNZ: in STRING_K; INFO: in STRING_I);                    --
    procedure CHANGE (    KENNZ_ALT, KENNZ_NEU: in STRING_K;                    --
                          INFO: in STRING_I);                                   --
    function IS_EL_OF (KENNZ: in STRING_K) return BOOLEAN;                      --
    function IS_SPACE return BOOLEAN;                                           --
    THERE_IS_NO_ENTRY, ALREADY_THERE, MEMORY_FULL: exception;                   --
end AUSKUNFTEI; ———————————————————————————————————————————                   --
```

- Beispiel Karten-Kästen-Beispiel
  DA vergessen:
    Kasten-Namensliste <- genauer
    Karten-Namensliste

```
abstract data object module KK_Namensliste is -- ************************** --
   --Dient der Handhabung einer Namensliste (abg. NL) von                    --
   --Karteikaesten (abg. KK). Die Semantik ist die folgende: ...             --
   procedure oeffne_KK_NL;                                                  --
   procedure schliesse_KK_NL;                                               --
   function ist_KK_NL_leer return BOOLEAN;                                  --
   function ist_noch_Platz_in_KK_NL return BOOLEAN;                         --
   function ist_KK_Name_vorh (Name: in KN_Typ) return BOOLEAN;              --
   procedure einfuege_KK_Name (Name: in KN_Typ);                            --
   procedure loesche_KK_Name (Name: in KN_Typ);                             --
   procedure positioniere_Anfang_KK_NL;                                     --
   procedure gib_akt_KK_Namen_aus (Name: out KN_Typ);                       --
   procedure naechster_KK_Name;                                             --
   procedure vorausgeh_KK_Name;                                             --
   function ist_KK_Name_Nachf_vorh return BOOLEAN;                          --
   function ist_KK_Name_Vorg_vorh return BOOLEAN;                           --
   KK_NL_leer, kein_Platz_fuer_KK_NL, KK_Name_ex_nicht,                     --
   KK_Name_Nachf_ex_nicht, KK_Name_Vorg_ex_nicht: exception    --
end KK_Namensliste; ———————————————————————————————————————————————
```

Gruppierung der Operationen
  Operationen auf der gesamten Liste
  Handhabung einzelner Elemente
  Bewegung auf der Liste
  Ausnahme in d. Reihenfolge der
    Sicherheitsabfragen

- Keller als abstraktes Datentypmodul

Programmieren-im-Großen-Teil

```
abstract data type module ITEM_STACK_STENCIL is ********************************** --
  ...                                                                               --
  type  ITEM_STACK_TYPE is private;                                                 --
  procedure INITIALIZE (ST: in out  ITEM_STACK_TYPE);                               --
  procedure PUSH (EL: in ITEM_TYPE; ST: in out ITEM_STACK_TYPE);                    --
  procedure POP (ST: in out ITEM_STACK_TYPE);                                       --
  function READ_TOP (ST: in ITEM_STACK_TYPE) return ITEM_TYPE;                      --
  function IS_EMPTY (ST: in ITEM_STACK_TYPE) return BOOLEAN;                        --
  function IS_FULL (ST: in ITEM_STACK_TYPE) return BOOLEAN;                         --
  ST_UNDERFLOW, ST_OVERFLOW, ST_NOT_INITIALIZED: exception;                         --
  -- Semantikbeschreibung: ...                                                      --
end ITEM_STACK_STENCIL; ------------------------------------------------------------    --

module body ITEM_STACK_STENCIL is ------------------------------------------------    --
  --Festlegung der Repraesentation des Datentyps,                                   --
  --(in Ada im physischen Teil der Schnittstelle):                                  --
  SIZE: constant INTEGER := 100;                                                    --
  type  SPACE_T is array (1..SIZE) of ITEM_TYPE;                                    --
  type ITEM_STACK_TYPE is                                                           --
    record                                                                          --
      SPACE: SPACE_T;                                                               --
      INDEX: INTEGER range 0..SIZE := 0;                                            --
    end record;                                                                     --
  ...                                                                               --
  -- Realisierung der Zugriffsoperationen (abhaengig von der                        --
  -- gewaehlten Repraesentation des Datentyps):                                     --
    ...                                                                             --
end ITEM_STACK_STENCIL; -- ************************************************       --
```

Programmieren-im-Kleinen-Teil

```
_____________________________________
...
  --Anzeige, daß Modul ITEM_STACK_STENCIL in einem anderen Modul
  --verwendet werden soll (Import). Dies wird spaeter behandelt.
```

Programmieren-im-Großen-Teil

```
_____________________________________
...
  --Verwendung im Rumpf dieses anderen Moduls:
    ST_1: ITEM_STACK_TYPE;
    EI_1 : ITEM_TYPE;
    if not IS_FULL (ST_1) then PUSH (EL_1, ST_1) else ... end if;
```

Programmieren-im- Kleinen-Teil

Modulbeziehungen

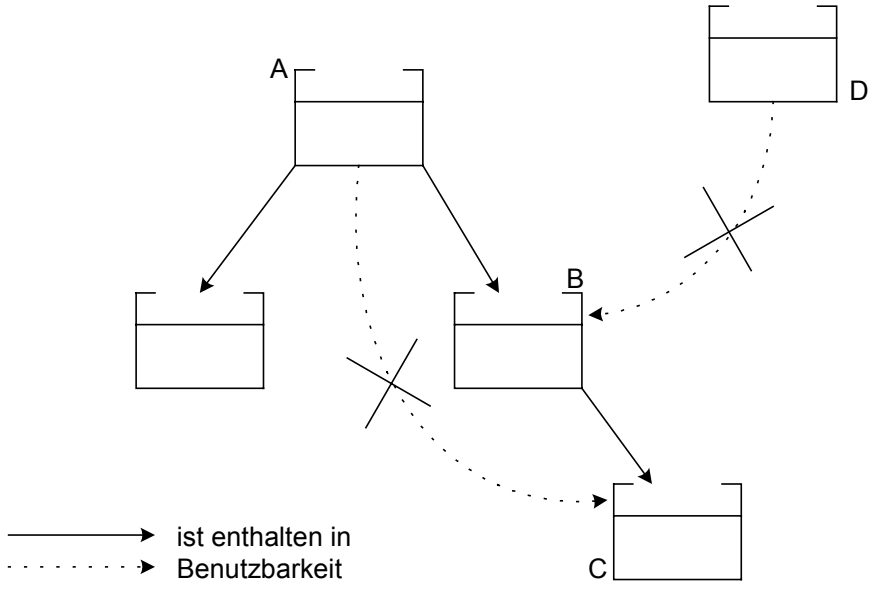
Die lokale Benutzbarkeit

- erste Art von Benutzbarkeit
  (weitere kommen)

- Unterscheidung von der ”Logik” der Verwendung
  auf Architekturebene
  nicht durch Programmiersprache, auf die wir abbilden
  Konzepte haben aber Ursprung in Programmier-
  sprachen:
  hier Schachtelungsprinzip/ Blockstruktur,
  Gültigkeit, Sichtbarkeit

- vorläufige Definition der lokalen Benutzbarkeit:
  ”Ein Modul ist in einem anderen enthalten und damit
  prinzipiell nur in einem lokalen Kontext benutzbar.
  Es muß explizit festgelegt werden, wo er benutzbar
  sein soll.”

- Was möchte man modellieren?

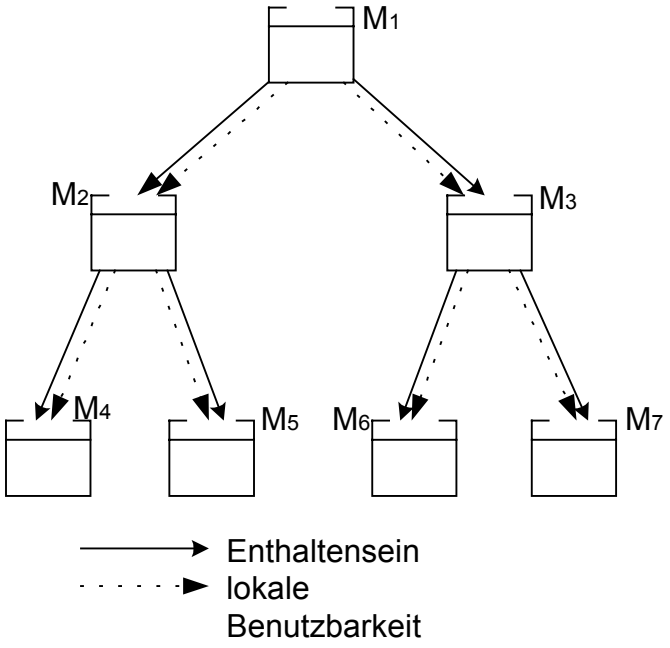


- Enthaltenseinsbeziehung (Strukturbez.)
  contains/is_contained_in-Klausel textuell

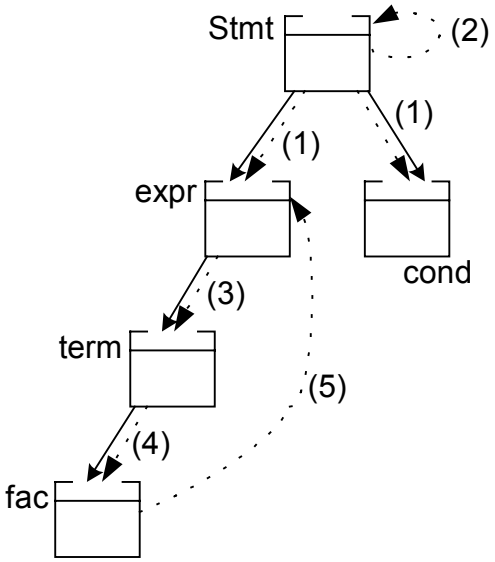
  lokale Benutzbarkeitsbeziehung (spez. Importbez.)
  local usability-Klausel mit Aufzählung einzelner Ressourcen

- Standardsituationen

a) Standardfall



b) Beispiel mit Rekursion

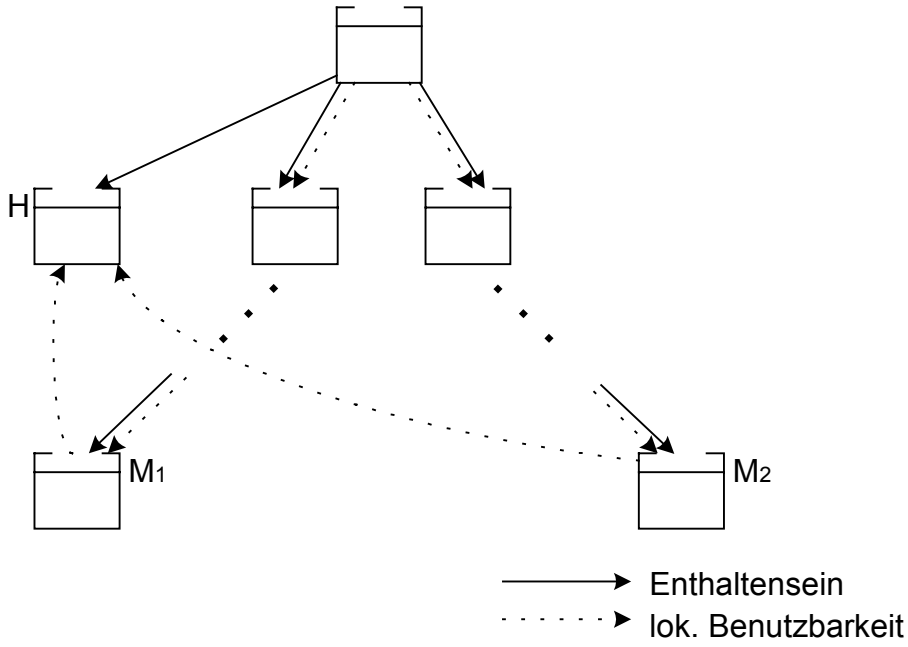


- Was haben wir für die Architekturmodellierung erreicht?
  lokale Bedeutung ausgedrückt durch
    Enthaltenseinbeziehung
  dadurch Einschränkung der prinzipiellen Benutzbarkeit
  explizite Festlegung im Rahmen dieser Möglichkeiten
  damit festgelegt
    welche Module M (lokal) verwenden darf
    und welche M (lokal) verwenden dürfen

Die allgemeine Benutzbarkeit

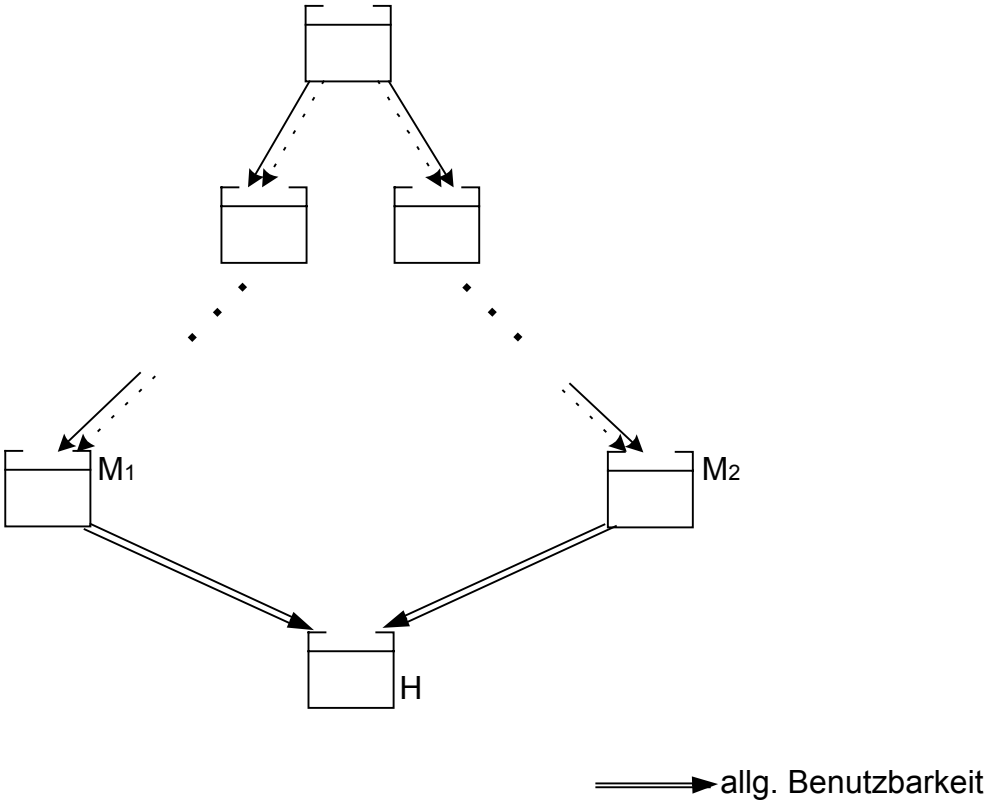
- zweite Art von Benutzbarkeitsbeziehung
  Ursprung Programmiersprachenkonzept: Importklausel
  logische Beziehung für Architekturmodellierung

- Warum genügt die lokale Benutzbarkeit allein nicht?
  (prinzipiell geht es: Algol, Pascal etc.)



Wohin hängen?
Nicht verständlich
Kann auch an anderer Stelle benutzbar gemacht werden
Kann auch in anderem Enthaltenseinsbaum
  benötigt werden

- Lösung



in Graphik andere Kanten

- vorläufige Definition allgemeiner Benutzbarkeit:
  ”Ein Modul (später auch Teilsystem) ist ein allgemeines Hilfsmittel in einem Softwaresystem, das zur Realisierung anderer Module benötigt wird. Es ist überall dort benutzbar, wo dies explizit festgelegt wurde.”

- textuelle Detailsprache:

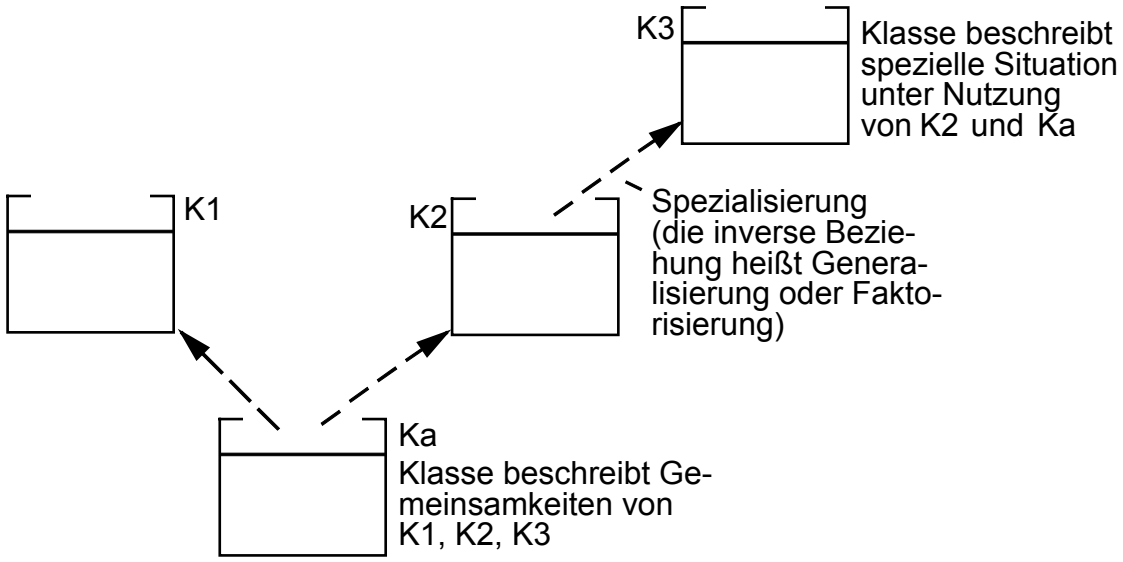
  general_usability-Klausel bei importierendem Modul mit Aufzählung der Ressourcen

  eine zugrundeliegende Strukturbeziehung (nimmt passenden allg. Baustein)

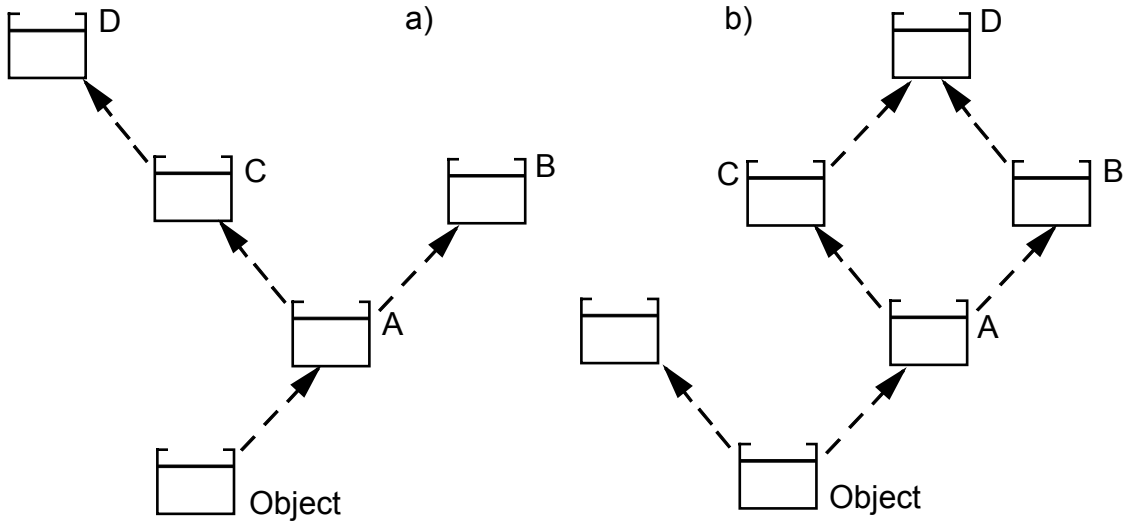
Objektorientierte Architekturmodellierung: Vererbung, Vererbungs-Benutzbarkeit

- Zielsetzung
  objektorientierte Konzepte zur Architekturmod.
  Verbindung zu den anderen Konzepten

- Zusammenf. d. objektorient. Strukturierungshilfsmittel
  Oberklasse - Unterklasse: Spezialisierung
  Vererbungs - Teilhierarchie (Faktorisierung, Generalisierung)
  spez. Methoden für spez. Objekte
  allg. Methoden für spez. Objekte
  spez. Methode von allg. induziert
  allg. Methode durch spez. induziert



- Unterklasse - Oberklasse



Semantik

Unterklasse B erbt alle Eigensch. der Oberklasse A,
sie sind für Objekte der Unterklasse verfügbar,
soweit nicht redefiniert

Objekte von B haben zusätzl. noch die in B fest-
gelegten Eigenschaften

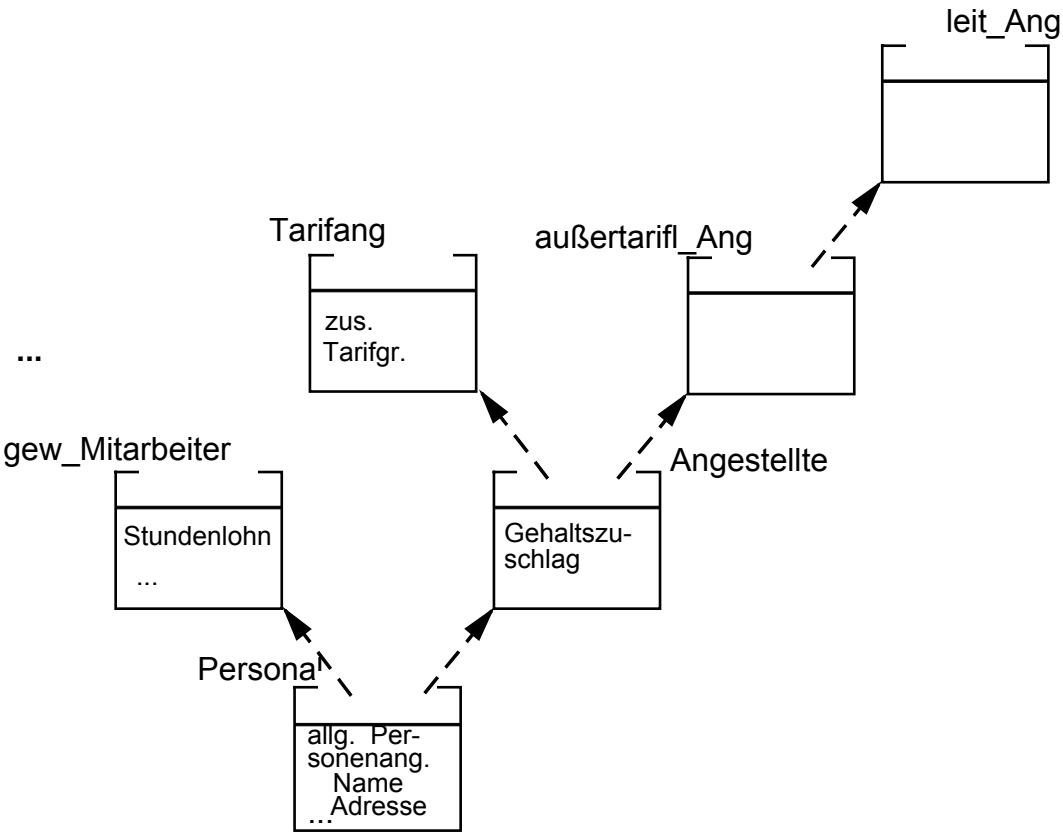
B ist Spezialisierung von A, A Generalisierung von B

Vererbung sollte nur Schnittstelle betreffen:
Meth. von A auch für Objekte von B anwendbar

Vererbungsbeziehung transitiv: Vererbungskette

Einfachvererbung - Mehrfachvererbung

- Beispiel Unterklasse, Vererbung

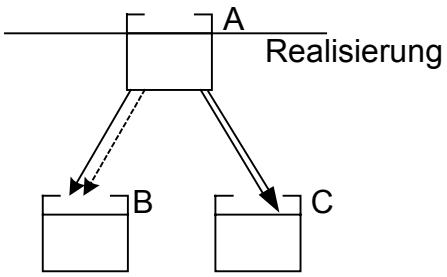


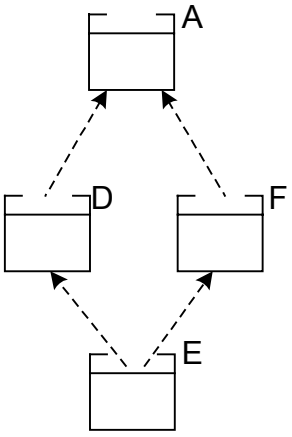
- Top-down
  Bottom-up } Auftragung der Bäume

  In diesem Sinne ist eine Spezialisierung abstrakter;
  nämlich weiter entfernt von Basismaschine

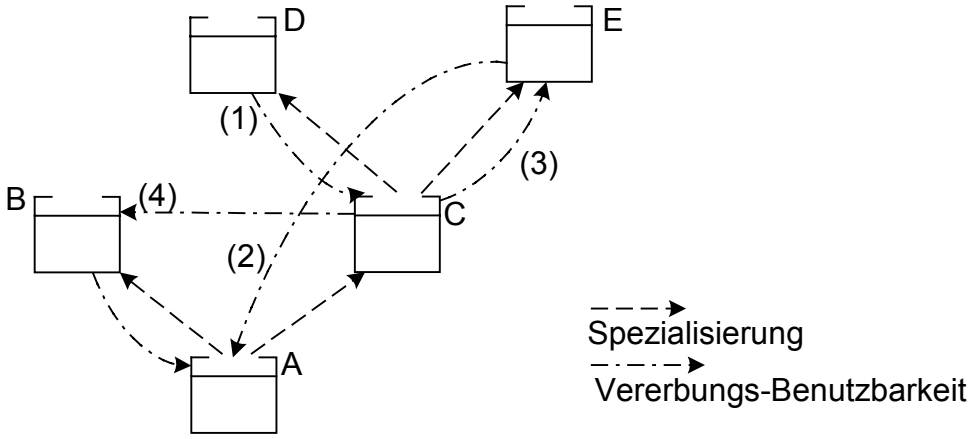
  (Faßt man abstrakt als allgemeingültig auf, so ist
  die Oberklasse abstrakter)

- Objektorientierung und Datenabstraktion





- Vererbung als Strukturbez. unseres Modulkonzepts



explizite Festlegung der Benutzbarkeit
Vererbungs-Benutzbarkeit: Kanal
Vielzahl von Vererbungs-Benutzbarkeit spiegelt innere Komplexität wider!

- Strukturbeziehung is_a-Klausel
spez. Importbez. innerhalb Vererbungshierarchien } textuell
mit Aufzählung der einzelnen Ressourcen

Teilsysteme

Größere Einheiten in einer Gesamtarchitektur

- Motivation der Einführung von Teilsystemen
  - Abschotten von Interna einer Teilarchitektur
  - Gruppe von Modulen zusammen benutzbar
  - Grobentwurf
  - Einheiten der Wiederverwendbarkeit
  - PO-Einheiten (Teilprojekt, Teilprodukt, Meilenstein)
  - Teststummel, Treiber: Zuordnung zu Modulgruppen

- somit Modellierungseinheit auf Architekturebene, größer als Modul
  - Verstehen einer bestimmten Architektur
  - Erstellung und Wartung einer Architektur
  - Wiederverwendbarkeit
  - Grundlage für Arbeitseinheit

- Forderungen:
  - Zusammengefaßte Module müssen log. zusammengehören
  - Teilsystem muß Unabhängigkeit und Eigenständigkeit besitzen

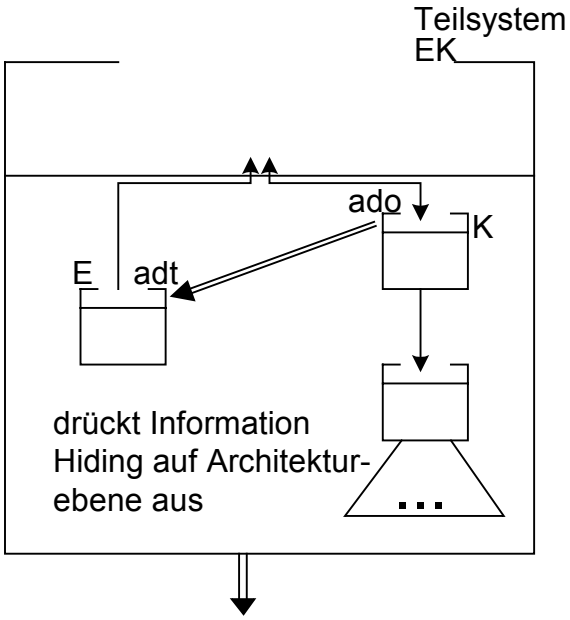
- ansonsten
  bel. Zusammenfassung von Modulen durch beliebige der hier eingeführten Beziehungen verbunden
  => Teilsystembegriff ist neues und zus. Konstrukt

Notation und Verwendung

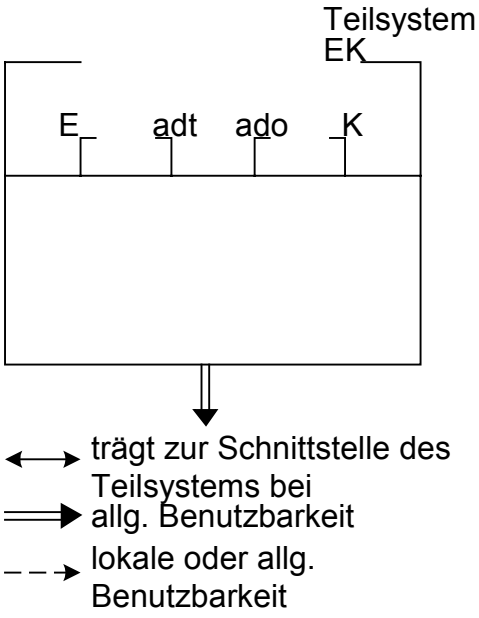
- Aus Forderung Unabhängigkeit Forderung für Einschränkungen der Modulbeziehungen im Rumpf und nach außen
  keine lokale Benutzbarkeit nach außen
  nach Möglichkeit keine allg. Benutzbarkeit innen

- graph. Notation für Teilsysteme

a) für den Entwerfer



b) für den Verwender



- textuelle Detaildarstellung für Teilsysteme

für den Verwender dieses Teilsystems
zu erstellen vom Entwerfer des Teilsystems

```
subsystem EK is
    --Exporte des Teilsystems gebildet aus Ressourcen der Schnittstellen der
    --exportierenden Module
    export from abstract data type module E is
        type ET is private;
        procedure op_i (F0:ET, ...);
        ...
        --Die Semantik der Schnittstellenoperationen ist die folgende:
        -- ...
    end E;
    export from abstract data object module K is
        procedure op_j (...);
        ...
        -- Die Semantik der Schnittstellenoperationen ist die folgende:
        -- ...
    end K;
end EK;
```

auszugestalten vom Entwerfer dieses Teilsystems

```
subsystem body EK is
    general import from Sub1.M1, Sub1.M2, Sub2.M3
        using ...  ;
    -- jetzt werden alle Module des Rumpfs in der üblichen Notation
    -- von Fig. 4.36 einschließlich ihrer Beziehungen aufgeführt
        ...
end EK;
```

- Entwurf mehrstufig
  - Übersichtsarchitektur (aus Modulen u. Teilsyst.)
  - Teilsysteme zu entwerfen (aus Modulen, Teilsyst.)

  meist genügt zweistufige Vorgehensweise:
  - Modellieren der Grobarch. (”Progr. im Größten”)
  - Modellieren der Teilarch. (Progr. im Großen)

Weitere Ansätze

Michael-Jackson-Entwurfsmethode: Motivation

- für best. Klassen von Programmen (kommerzielle Probleme)
  z. B. Personaldatenverwaltung
  Fakturierung (Rechnungserstellung)
  Verbrauchsabr. (Accounting)
  Bestandsverwaltung

- Charakterisierung kommerzieller Probleme
  - grosse Datenmengen in EA
  - wenig rechenintensiv, einf. Operationen
  - Verwendung/ Verknüpfung mehrerer Dateien
  - periodische Wiederholung d. Programmlaufs über längere Zeit

- Problemstruktur an Struktur eines COBOL-Programmes ablesbar

identification devision	
...	Organisatorisches
environment division	
...	Bez. zum Betriebssystem
data division	
...	Beschr. Daten, EA-Puffer
procedure division	
...	Anweisungen

- pufferorientierte EA

  Bsp. Datei enthält ”Lochkarten” für Personendaten Typ 01

  oder ”Lochkarten” für Dienstperiode Typ 02

In IO-Section der Environment Division wird dieser Datei etwa Kartenleser mit Namen ”Eingabe” zugeordnet:

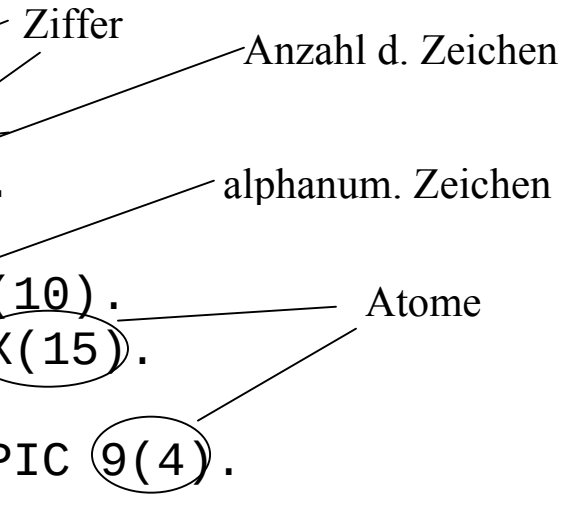
```
READ EINGABE
```

holt ”Karte” in den Puffer

File Section der Data Division enthält dann mögliche Beschreibung des Puffers

```
FILE SECTION
  FD EINGABE RECORDING F
        LABEL RECORD OMITTED
        DATA RECORDS ARE KARTE-1 KARTE-2

  01 KARTE-1.
     02 TYP PIC 99.
     02 PERS-NR PIC 9(6).
     02 NAME.
        03 VORNAME PIC X(10).
        03 NACHNAME PIC X(15).
     02 WOHNUNG.
        03 POSTLEITZAHL PIC 9(4).
        03 ORT PIC X(15).
        03 STRASSE PIC X(20).
        03 NUMMER PIC 9(3).
```

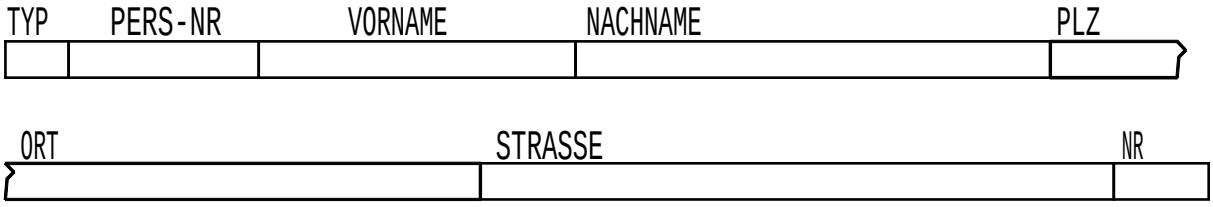


```
01 KARTE-2.
     02 TYP PIC 99.
     02 PERS-NR PIC 9(6).
     02 DIENST occurs 6 TIMES.
        03 DATUM.
           04 TAG PIC 99.
           04 MONAT PIC 99.
        03 ANFANG.
           04 STUNDE PIC 99.
           04 MINUTE PIC 99.
        03 ENDE.
           04 STUNDE PIC 99.
           04 MINUTE PIC 99.
```

Wiederholung

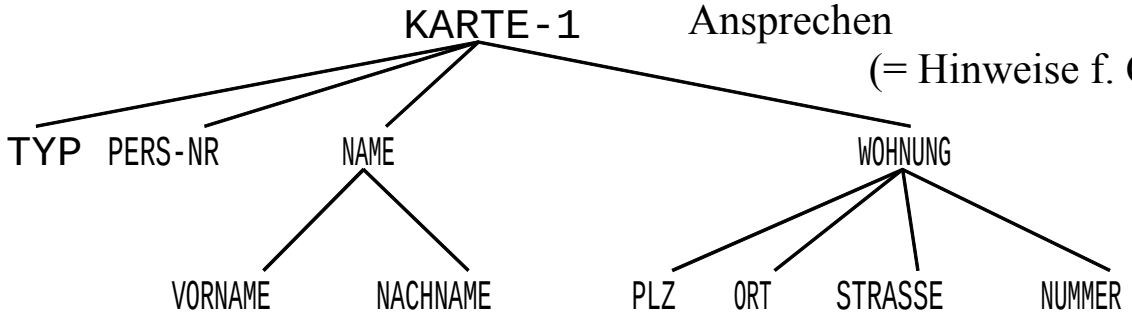
Die Angaben für KARTE-1 und KARTE-2 beziehen sich auf den gleichen Speicherplatz

- Speicherstruktur f. KARTE-1
  analog f. KARTE-2



Speicherung nur der Atome (sequentiell)

- log. Struktur f. `KARTE-1` logische Struktur zum Ansprechen (= Hinweise f. Compiler)



- Records aus Atomen
  - durch Sequenz (alle Daten mit gleicher Stufennummer) normalerweise unterschiedlich strukturiert
  - durch Iteration: Wiederholung gleichstr. Daten mith. Feldern; hier `occurs`-Klausel
  - durch Selektion: Interpretation des gleichen Platzes auf mehrere versch. Arten, hier `KARTE-1` und `KARTE-2` bzw. durch `redefines`-Klausel (variante Records)

- Korrespondenz mit Kontrollstrukturen wohlstr. Programme

  Sequenz:

```
PARAGRAPH-1
  MOVE A TO B.
  ADD B TO C GIVING D.
```

atom. Objekte = Anweisungseinh.

  Iteration:

```
PERFORM PARAGRAPH-1
  UNTIL X=0
```

  Selektion:

```
IF X<0 THEN
  PERFORM P1
ELSE
  PERFORM P2
END
```

- Idee von M. Jackson, wegen obiger Korrespondenz
  - gleiche Notation für aktive Objekte (Kontrollstr.) und passive Objekte (Datenstrukturen)
  - Programmstruktur läßt sich direkt aus der Struktur der Eingabedaten und Ausgabedaten ableiten

JSP (Jackson Structured Programming)

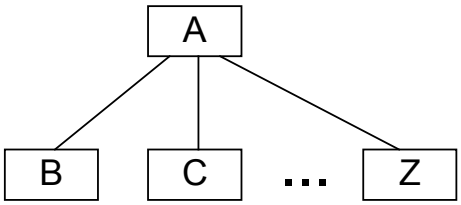
JSP (J. structured design) nicht betrachtet

- Charakteristika
  - datenstrukturorientierter Ansatz
    ("structured programming leaves the programmer alone")
  - " kann EDV-Laien gelehrt werden"
    (Kommunikation in Problemanalyse)
  - duale Sicht (Funktionen-Daten).
    Start: Datenstrukturen

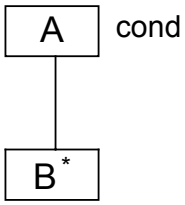
- Vorgehensweise
  1. Datenstrukturen in Diagrammform
  2. Zuordnungen
  3. Programm(grob)struktur
  4. Auflisten Operationen, Zuordnen Op.
  5. Ableiten "Pseudocode"
     (Übertragung manuell oder automatisch)

Elemente

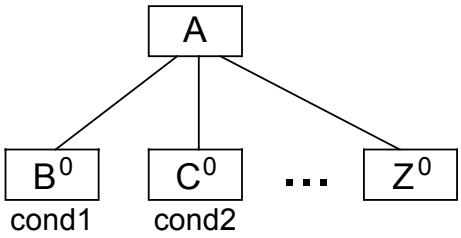
- Diagramm



Sequenz



Iteration



Selektion

Pseudocode:

```
A seq
  do B;
  do C;
  ...
  do Z;
A end;
```

```
A iter until not cond
  do B
A end;
```

```
A iter while cond
  do B
A end;
```

```
A select cond1
  do B
A or cond2
  do C
  ...
A or
  do Z
A end;
```

- Typ eines Knotens
- innere Knoten-Blätter
- Strukturdiagramm ≤ DIN A4-Seite

Beispiel 1

Datei aus Records d. Länge 80 und folg. Typen

T1: Personal-Daten

T2: Dienstzeit-Daten

T3: Lohn-/Zulagen-Daten

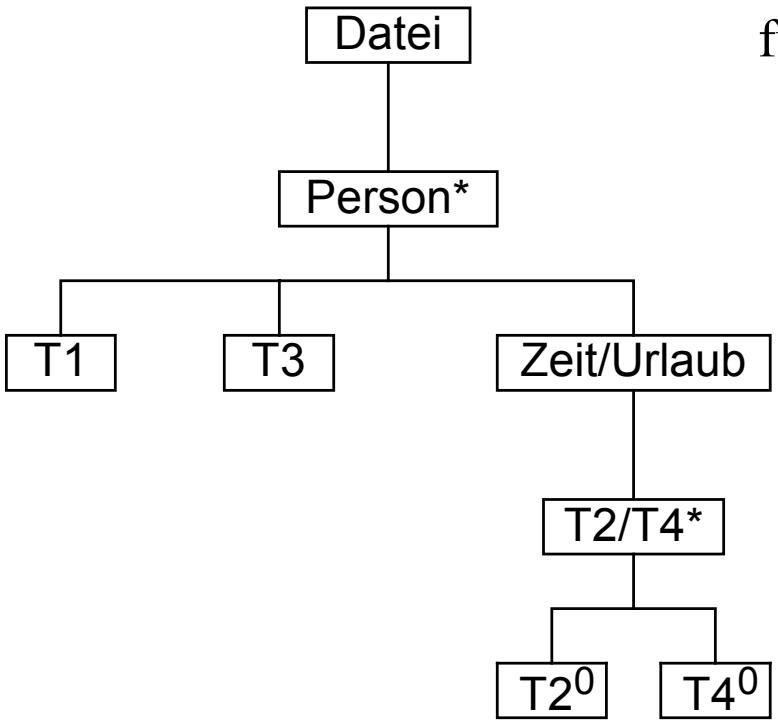
T4: Urlaubsdaten

mit folgender Struktur

$(T1\ T3\ (T2 \vee T4)^{+})^{+}$

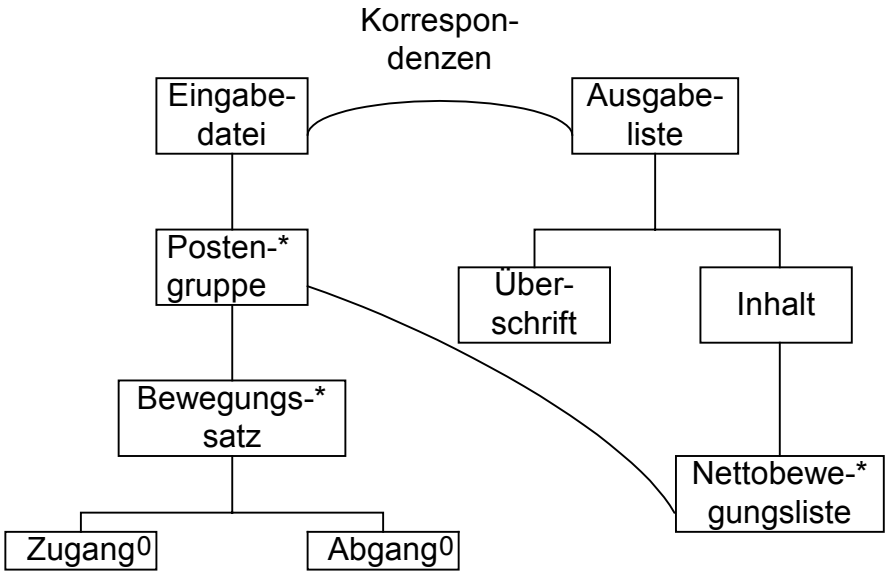
Sequenz Selektion Iteration

Baumstruktur für Beispiel 1

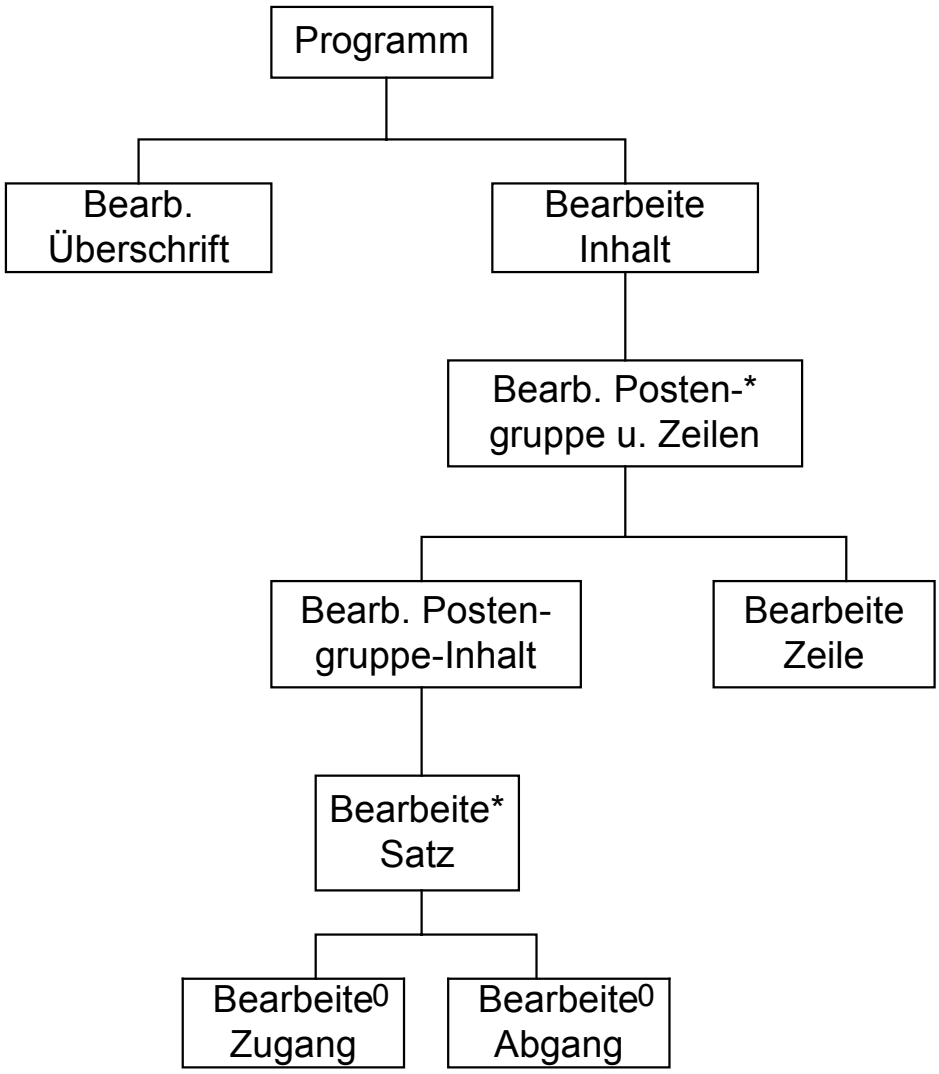


Beispiel 2

Eingabe $((Z \vee A)^{+})^{+}$ Ausgabe Üb. (Nettoz.)$^{+}$



- abgeleitete Programmstruktur

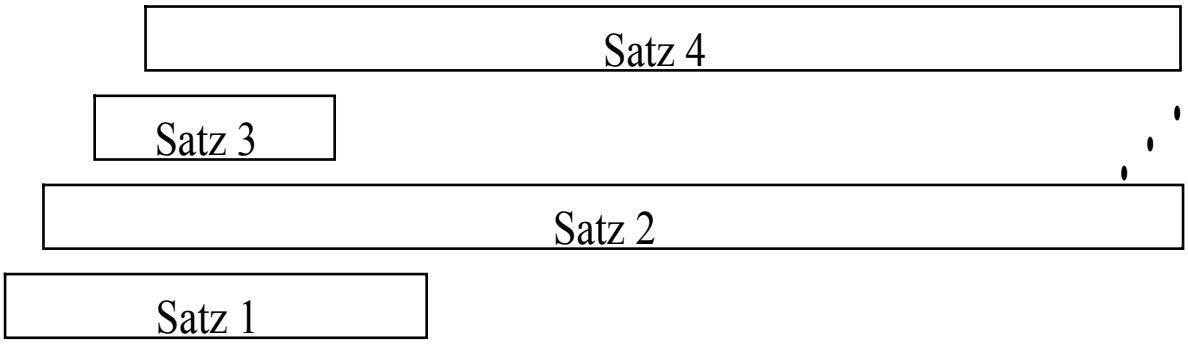


- Probleme mit Jackson-Methode
  - Strukturkonflikte (Matrix zeilenweise ein, spaltenweise aus)
  - Ausnahmebehandlung, die erst zur Laufzeit festgelegt werden kann (Verarbeitung von Listeneinträgen; Ausnahmebehandlung, falls bestimmter Eintrag nicht vorkommt)

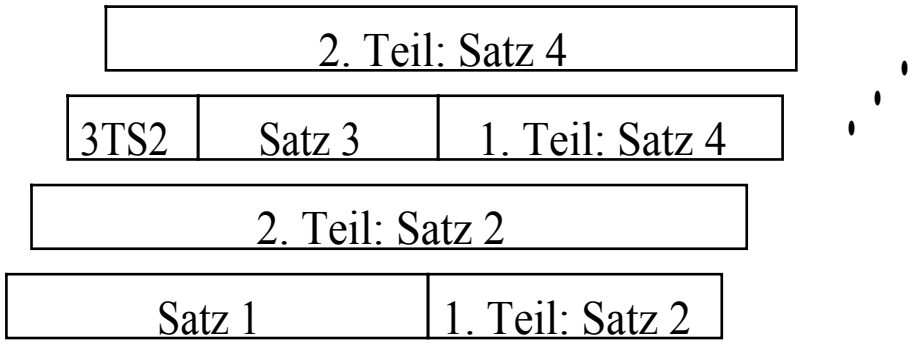
! - inflexibel gegen Veränderung der Struktur von E/A-Dateien, neue Programmstruktur jedoch ”hergeleitet”

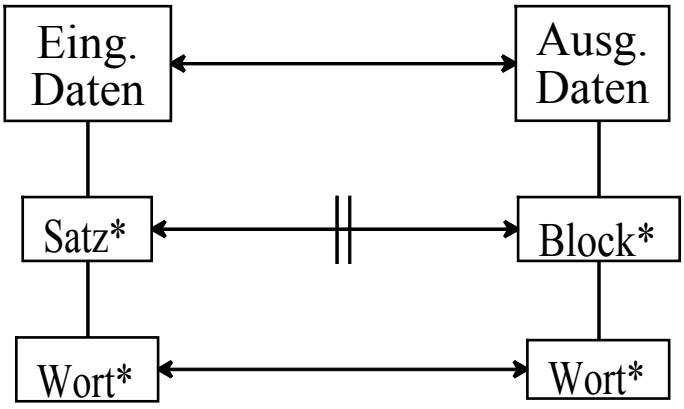
Strukturkonflikte / Programminversion

Eingabe: Sätze variabler Länge



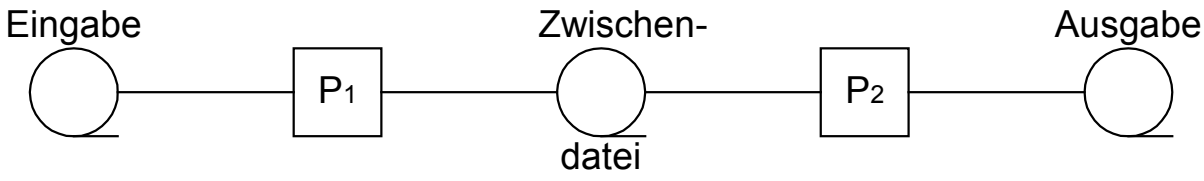
Ausgabe: Blöcke fester Länge

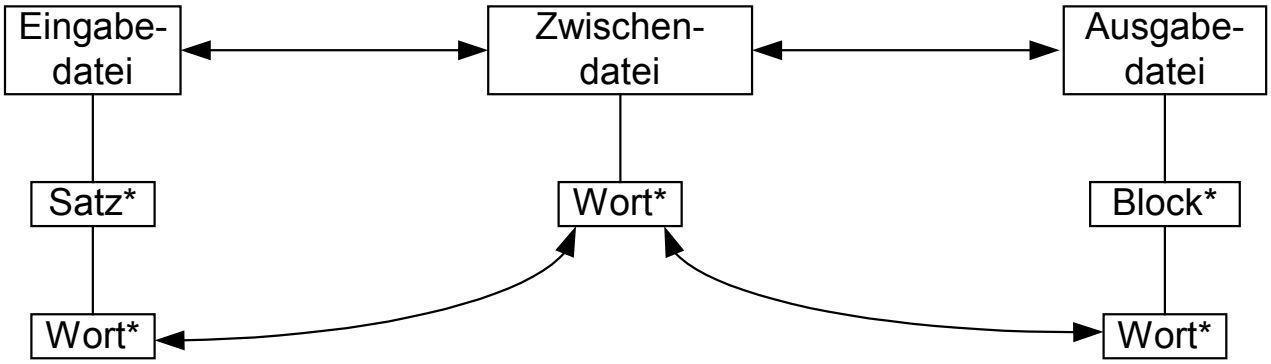




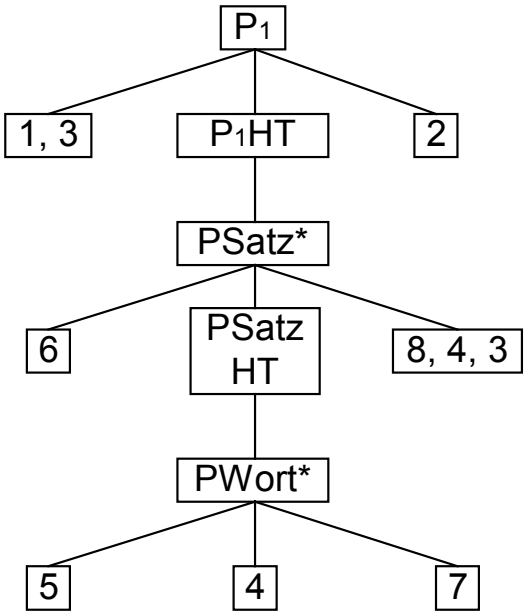
Begrenzungskonflikt (boundary clash)
andere: Reihenfolgekonflikt
Überlappungskonflikt

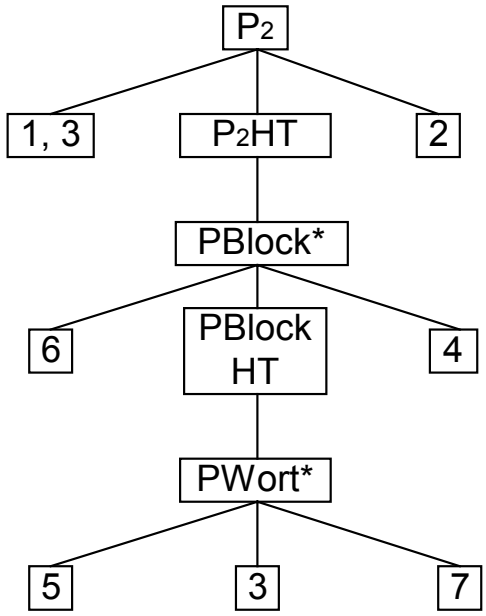
Lösung: Einführung einer Zwischendatei
Zwei Programme P_1, P_2





Grobe Lösung





1. Eröffne Dateien
2. Schließe Dateien
3. Lies Eingabedatei
4. Schreibe Zwischendatei
5. Wort in Ausgabebereich
6. Satzindex ::1
7. Satzindex incr 1
8. Letztes Wort in Ausgabebereich

1. Eröffne Dateien
2. Schließe Dateien
3. Lies Zwischendatei
4. Schreibe Block
5. Wort in Ausgabebereich
6. Blockindex ::1
7. Blockindex incr 1

Kann auf Zwischendatei verzichtet werden?

(Kann Hauptspeicher zum Datenaustausch genommen werden?)

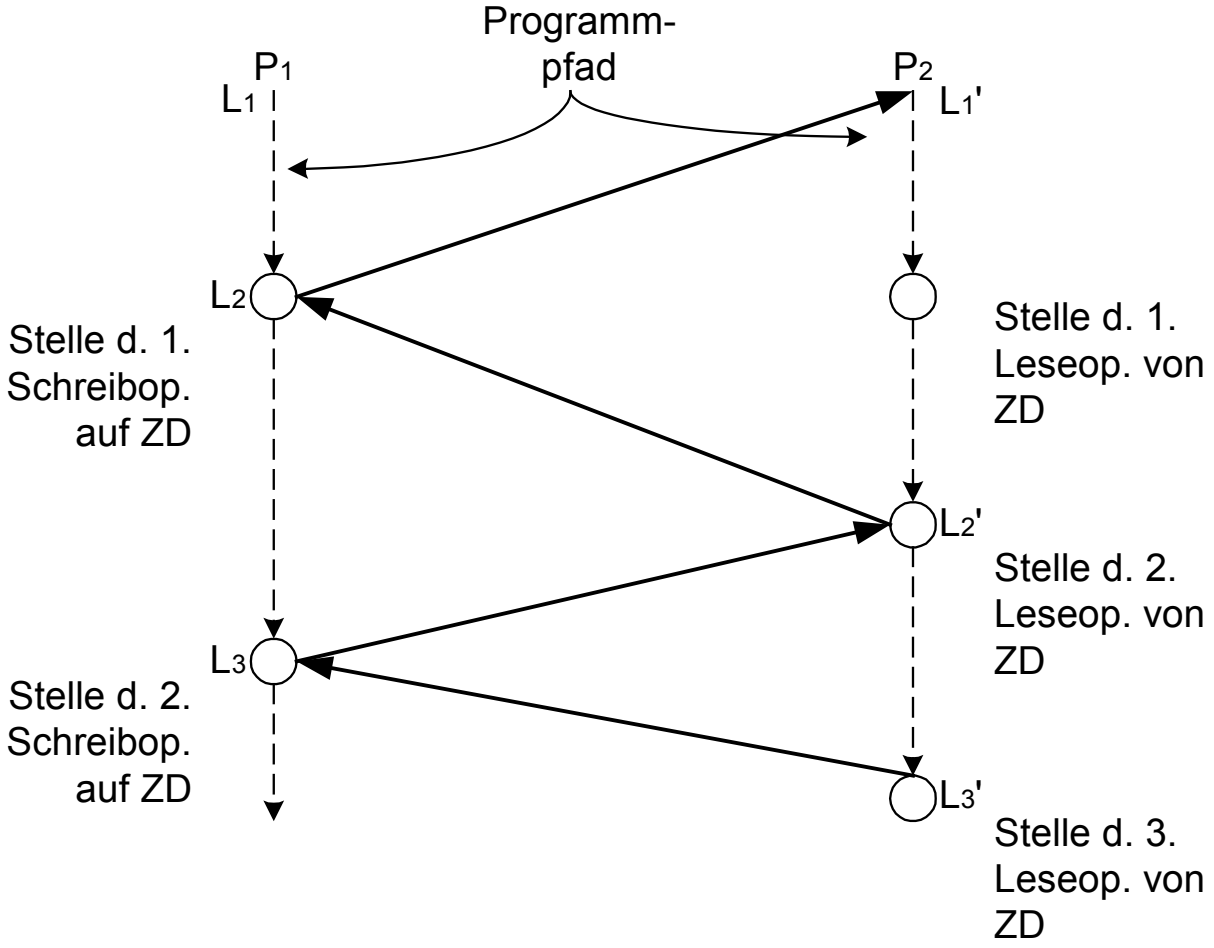
Lösung: Inversionstechnik

- Betrachte beide Programme als ”unabhängig laufend”

- Schreiboperation auf Zwischendatei
  Leseoperation von Zwischendatei
  jetzt überflüssig

- Synchronisation der Programme nötig:
  z. B. Wort von P_2 erst dann gelesen, wenn von P_1 erstellt

Anstelle von Lese-/Schreiboperationen springen zum anderen Programm nach Koroutinen-Schema:



Inversionstechnik ist spez. Implementierung (Scheduling) für die beiden ”unabhängigen” Programme, die sich an Lese-/Schreibop. synchronisieren müssen!

HIPO

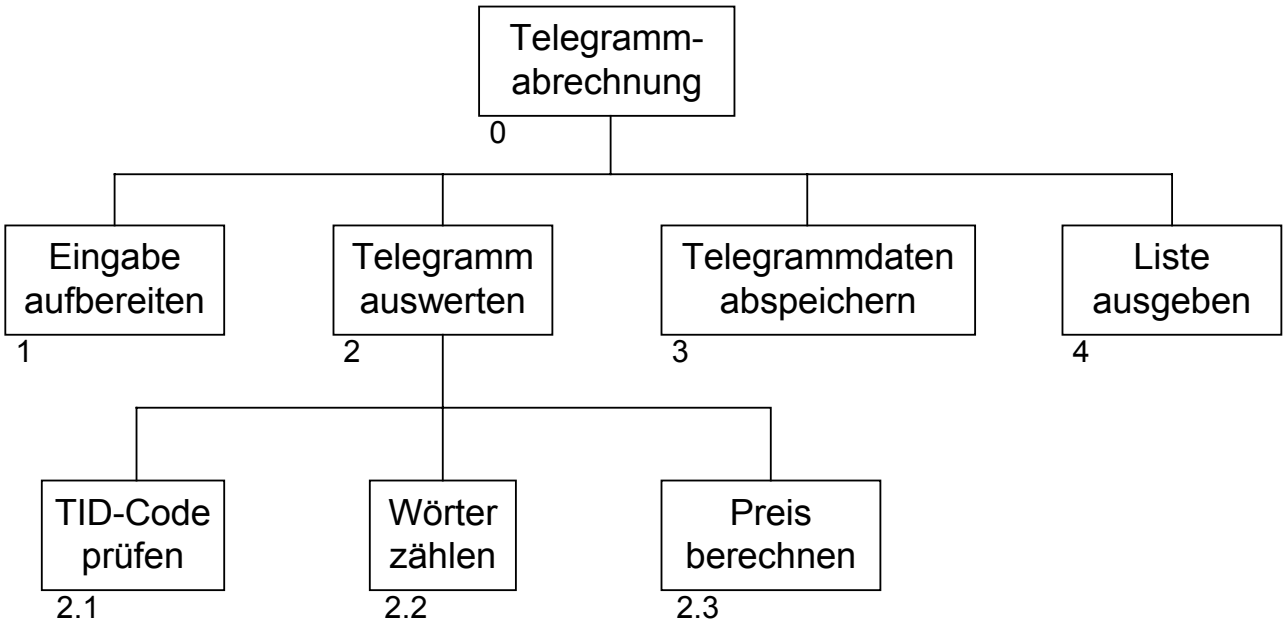
(Hierarchy plus Input-Process-Output) IBM 74

für gesamten Lebenszyklus (?)

besteht aus:

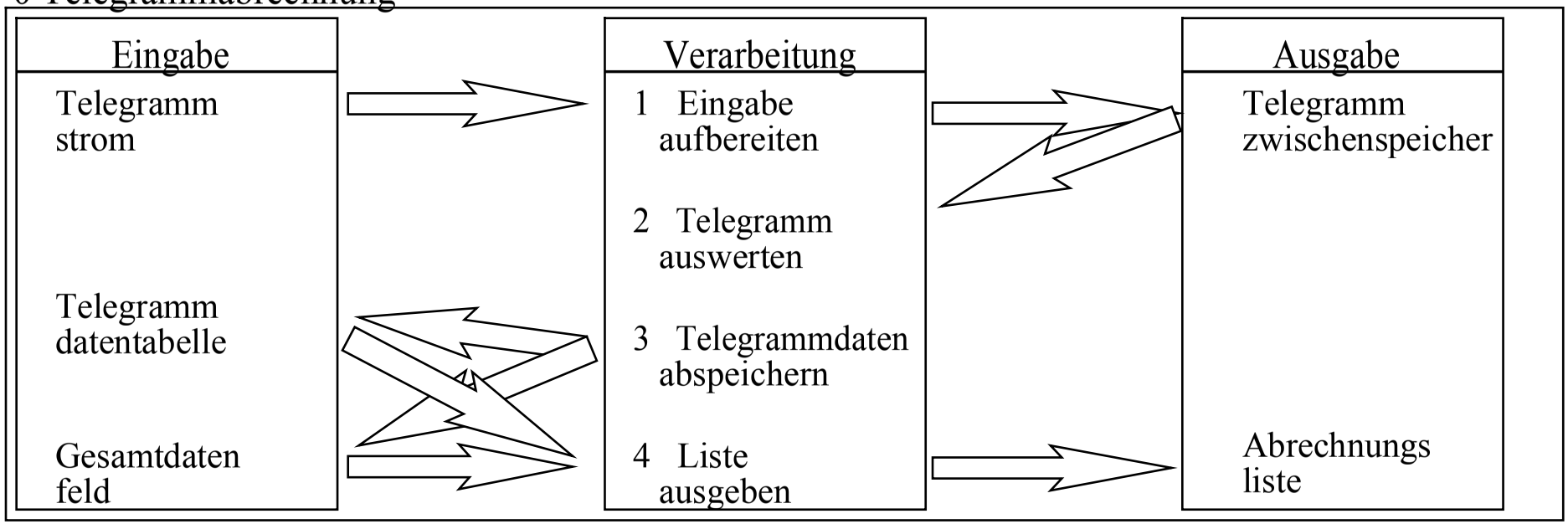
1) Visual Table of Contents (VTOC): Baum
   Zerlegung eines Programms in Teilfunktionen

2) IPO-Schema:
   Überblicksdiagramme
   Detaildiagramme
   sind Verknüpfung Eingabedaten mit Ausgabedaten über ”wesentliche” Verarbeitungsschritte

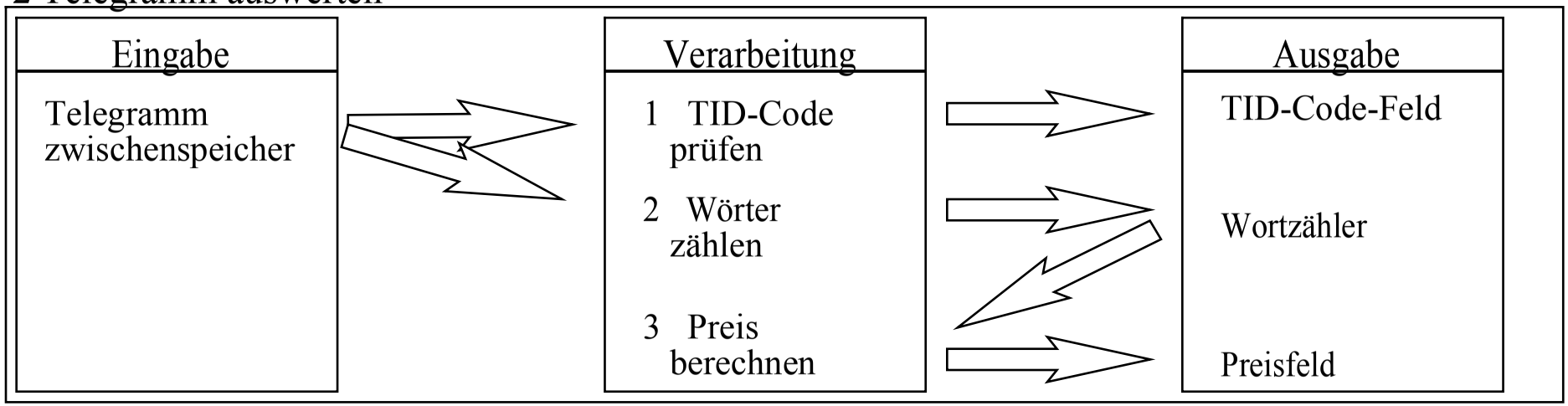


VTOC (Visual Table of Contents) des Telegrammabrechnungsprogramms (Ausschnitt)

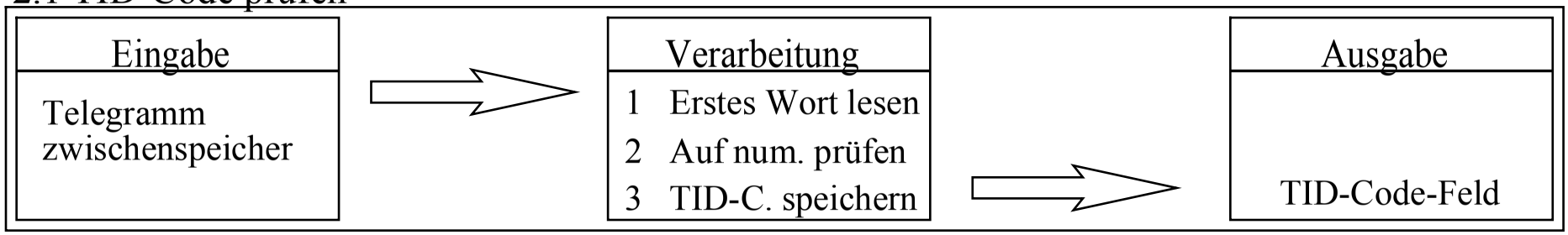
0 Telegrammabrechnung



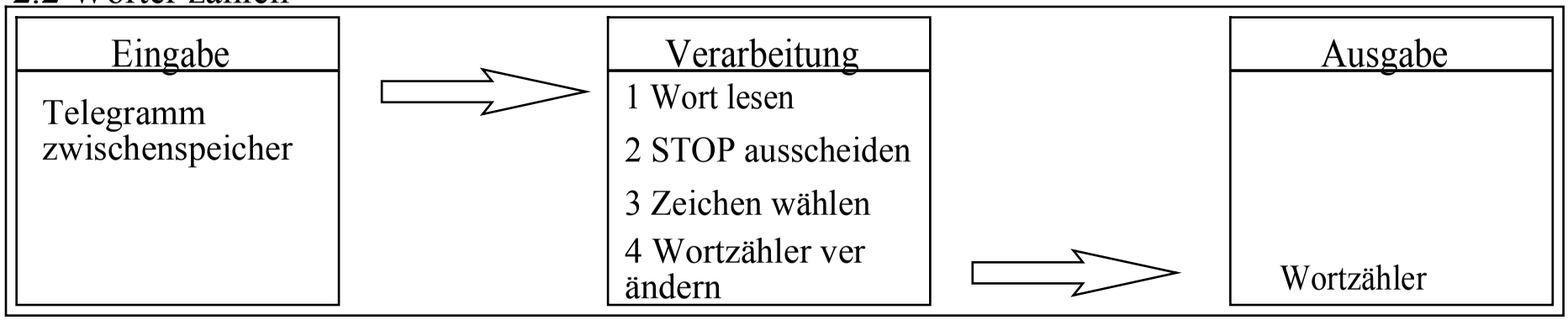
2 Telegramm auswerten



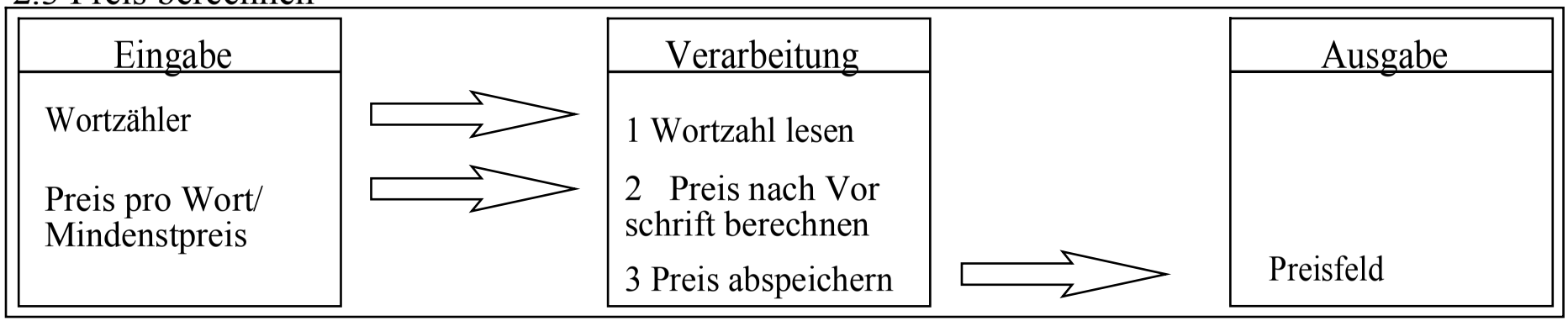
2.1 TID-Code prüfen



2.2 Wörter zählen



2.3 Preis berechnen



Wertung/Zusammenfassung

Schema: in VTOC und entspr. ebenso in IPO-Schema nur funktionale Abstraktion

Probleme mit HIPO:

- Zwischendaten bei Ein- oder Ausgabedaten durch ”falsche” Pfeile gekennzeichnet
- Kein Datenverfeinerungsmechanismus
  Gefahr der Über- oder Unterspezifikation
- Zur Strukturierung nur Sequenz
- keine Ausnahme- oder Fehlerbehandlungsspezifikationen

mögliche Konsistenzprüfungen:

- Konsistenz VTOC mit IPO-Schemata
- Konsistenz der IPO-Schemata untereinander

erweiterte HIPO-Beschreibung:

unter jedes Diagramm Kasten mit umgangssprachlicher Erläuterung (z. B. für Datenstrukturierung, Fehlerbehandlung, Implementierungsvorschriften etc.)