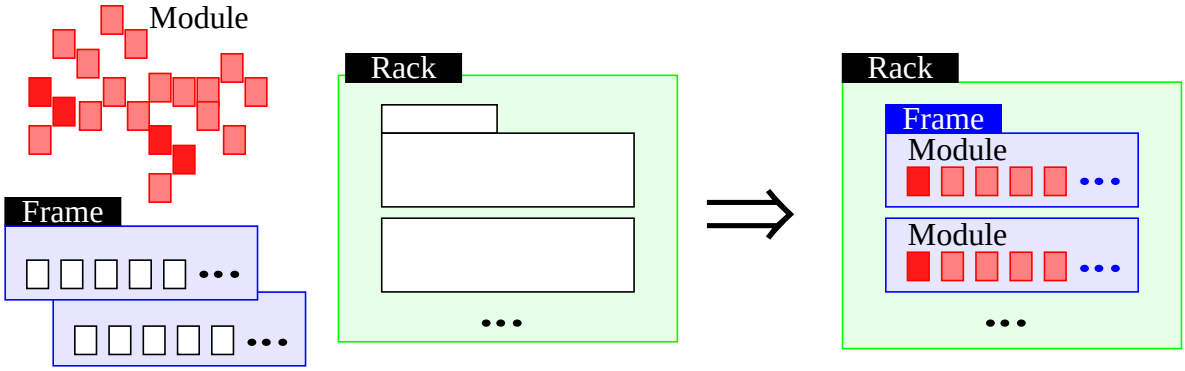
# Wissensbasiertes Konfigurieren

**Markus Stumptner**

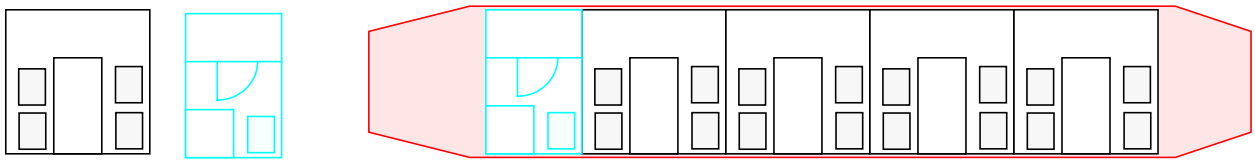
Institut für Informationssysteme
Technische Universität Wien

# Was ist Konfiguration?

- Zusammensetzen komplexer (meist technischer) Systeme aus einer Menge von Komponenten
- Typische Beispiele:
  - Elektronische Produkte (Computer, Telekommunikationssysteme):




  - Eisenbahnwaggons, Küchen, Gebäude, ...



  - Autos (Ausstattungsvarianten)

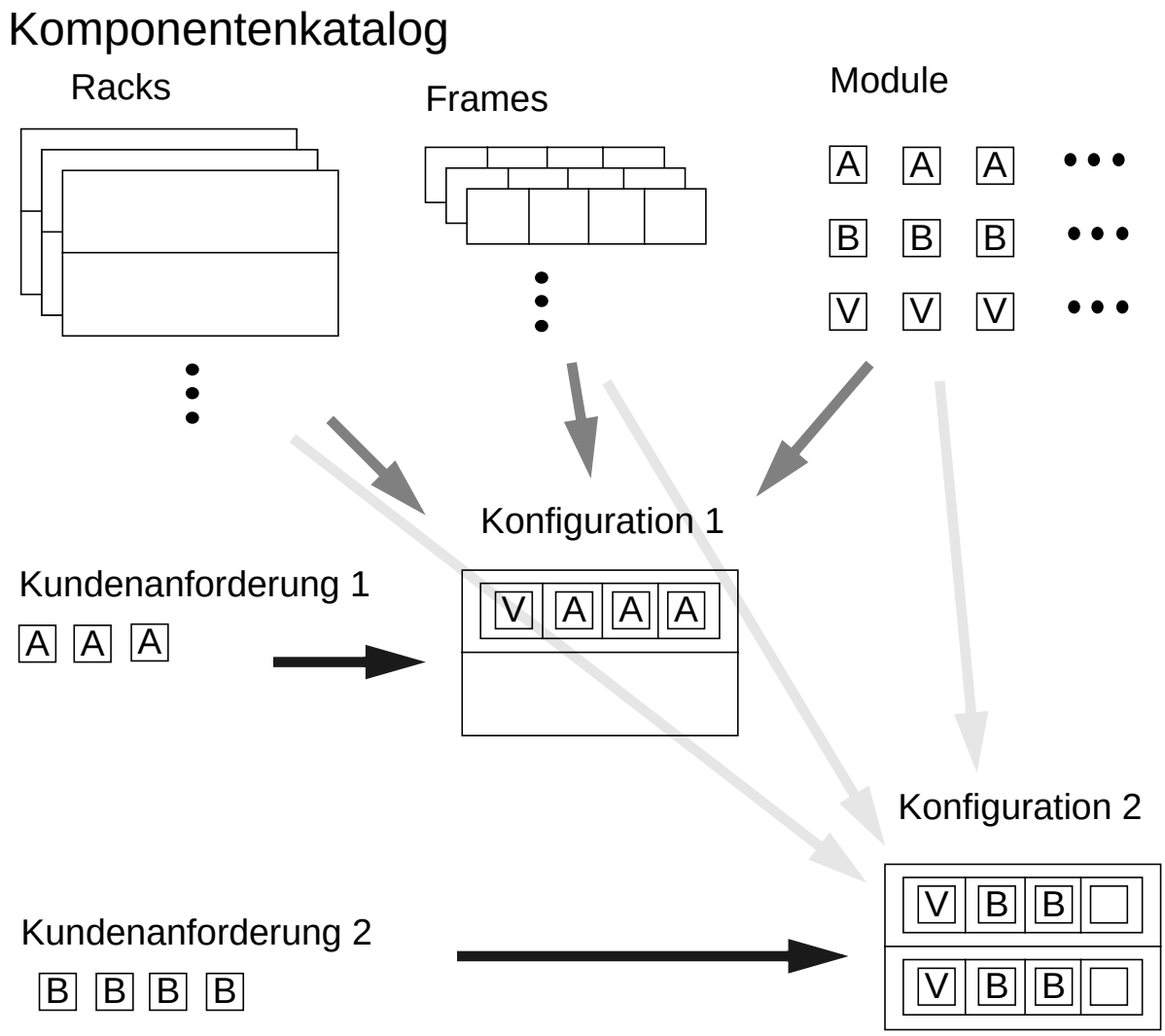
# Hintergrund

- Übergang von der Massenfertigung zur Spezialanfertigung
  - kürzere Produktentwicklungszyklen
  - stärkere Anpassung an spezielle Kundenforderungen
- Ausweg: “A la carte”-Fertigung:
  - Massenfertigung einzelner Komponenten
  - Zusammensetzung zu komplexen Systemen für den Einzelkunden
- aber: komplexere Produkte erfordern SW-Unterstützung
  - Zeitdruck bei paralleler Konfiguratorentwicklung -> Einsatz wissensbasierter Systeme
  - klassische Anwendung: Computersysteme (R1/XCON)

# Einsatzumfeld

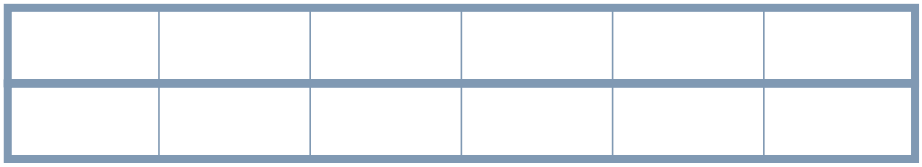
- Anforderungen (“Bestellformular”)
- Menge von Vorschriften
  - Entwicklung,Vertrieb, Montage, Service ...
- Mögliche Benutzer:
  - Techniker bei der Fertigung
  - Vertriebskonfiguration
  - Verkaufskonfiguration (Angebotserstellung, z.B. auch Verkäufer beim Kunden)
  - *Kunde auf der Webpage*
- Ergebnis: Zulässige Konfiguration (Komponenten + Verbindungen)
- Typische Annahme: Diskrete Komponenten, endliche Wertebereiche

# Aufgabenstellung Konfiguration (Beispiel)




**Component-Port-Modell**: Zusammenstecken von Komponenten an bestimmten Ports

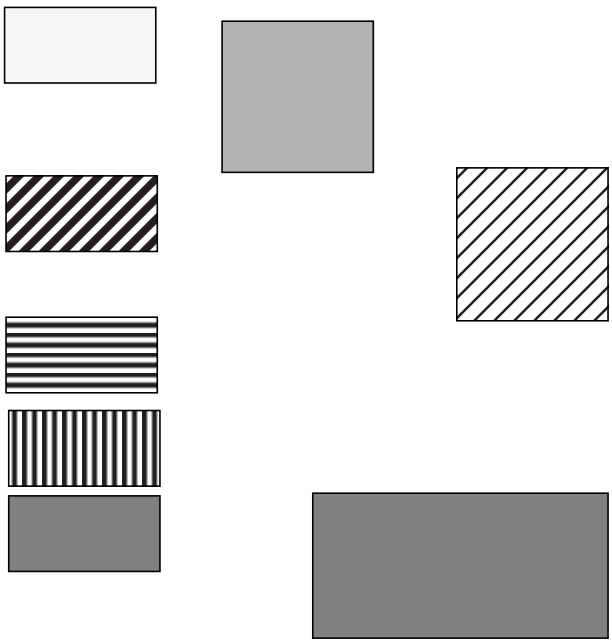
# Beispiel-Teilekatalog



Gehäuse

Grundlegende Teile: A,B,C,D (in jeweils 2 Varianten)

| Teil | benötigt | Größe |
|---|---|---|
| A-1 | 2 * B | halb |
| A-2 | 3 * B | ganz |
| B-1 | 2 * C | halb |
| B-2 | - | ganz |
| C-1 | - | halb |
| C-2 | - | halb |
| D-1 | B und 2*C | halb |
| D-2 | C-1 | doppelt |

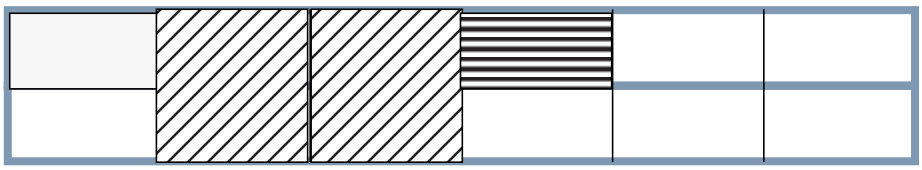


# Beispiel-Vorschriften

- Komponenten in alphabetischer Reihenfolge (links nach rechts, oben nach unten)
- Identische Komponenten ins selbe Gehäuse
- Mehrere Gehäuse: Erweiterungsstecker
- komplett leere Slots nur rechts
- oben nur leer, wenn unten auch

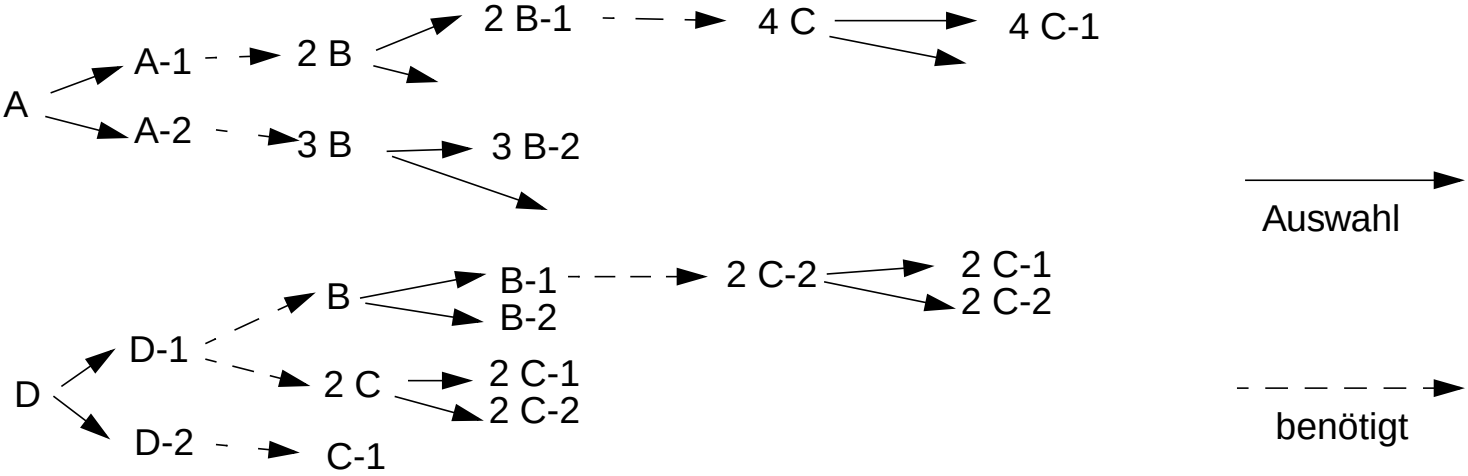
# Beispiel-Lösung

Typische Teilbelegung:

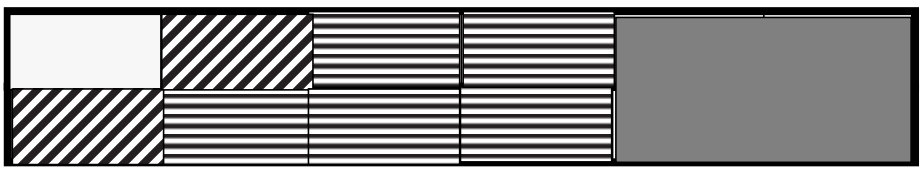


für Spezifikation: {A-1}

Expandieren der Bauteile für Spezifikation {A,D}:




Mögliches Ergebnis:



# Vorgangsweisen

- **Strukturbasierte Methoden**: Zerlegungsbäume können auch rein hierarchisch sein (Computer/Gehäuse/Bus)
  - entwickelt aus Stücklistenverarbeitung
  - einfachster Ansatz, aber beschränkt
- Vorgaben: z.B.,
  - bestimmte Teile müssen enthalten sein (key components) - siehe voriges Beispiel
  - globale Anforderungen:
    möglichst wenig Platz, minimale Kosten

# Schwelleneffekt

- kleine Unterschiede in der Spezifikation können große Effekte auslösen.
  - Z.B. {A, D, D} statt {A, D, B}: ein Gehäuse reicht nicht mehr - größere Kosten, Gewicht, oder größerer Stromverbrauch benötigt größeres Netzteil, ...
  - Problematik: globale Ressourcen (Gewicht, Kosten) abhängig von lokalen Entscheidungen

# Horizonteffekt

- spätes Feststellen von getroffenen Fehlentsc-heidungen
  - Bauteil A erfordert teurere Zusatzteile als B
  - Zusatzteile für Teil A erforderlich - vielleicht danach Schwellenüberschreitung
  - Ein Teil, das mit B inkompatibel ist, steht an anderer Stelle der Konfiguration (oder wird erst später dort eingefügt)
- Problem für heuristische Algorithmen (z.B. A*): in der Regel keine korrekte Abschrankung für Heuristiken möglich (es kann immer am Ende ein bisher nicht überprüfter Constraint auftauchen, der mir die Lösung inkonsistent macht)

# Spezielle Vorgehensweisen

- **Propose-and-revise**: Stelle eine Lösung zusammen, benutze Prüfungswissen, um Schwachstellen zu finden, benutze heuristisches Reparaturwissen, um zu reparieren (z.B. VT: System für Entwurf von Aufzügen)

- **Partial Commitment**:
  CD-Player A und B stehen zur Wahl, erfordern beide den Verstärker C.
  Richtige Reihenfolge: C festlegen, A/B noch unentschieden lassen (vielleicht gibt es noch unberücksichtigte Einschränkungen)

# Regelbasiertes Konfigurieren

- XCON (ab 1981) Klassisches regelbasiertes Konfigurationssystem
- ursprünglich VAX-Computer, später alle DEC-Rechner/Netz-Systeme
- Typische Regel:

  **IF**: The most current active context is assigning a power supply
  **and** a unibus adapter has been put in a cabinet
  **and** the position it occupies in the cabinet is known
  **and** there is space available in the cabinet for a power supply for that cab.
  **and** there is an available power supply
  **and** there is no H7101 regulator available
  **THEN**: add an H7101 regulator to the order.

- Regelbasis: um 1990 etwa 17000 Regeln
- Enormer Wartungsaufwand, eigene Entwicklungs-methodik eingeführt

# Wissensrepräsentation für Konfiguration

- Projektziel: Entwicklung eines problemunabhängigen Konfigurationstools, schnelle Beschreibung und leichte Wartung einer konkreten Anwendung
- Wissensdarstellung
  - deklarativ: was ist eine Lösung, nicht wie sie gebaut wird
  - natürlich: dem Problem angepaßt
- Schlußfolgerungsmechanismus fix eingebaut, flexibel genug für verschiedene Bereiche

==> Constraint Satisfaction Problem (CSP)

# Beispiel-CSP: Auto-Konfiguration

Variablen: Package, Roof, AirCond, Battery

Wertebereich: dom(Package)={lux,std}
dom(AirCond)={ac1,ac2,none},
dom(Battery)= {large,medium,small},
dom(Roof)={closed,Sunroof,Cabrio}

Constraints: z.B.

Package=lux -> AirCond = ac1 or AirCond = ac2;
Package=lux -> Roof = Sunroof or Roof = Cabrio;
Battery=small -> AirCond = none;
Roof = Cabrio -> Battery=large;

- Variable entsprechen “Plätzen” in der Konfiguration, Werte den Komponenten
- Lösung: alle Variablen zugewiesen, alle Constraints erfüllt
- Lösungsfindung: Prinzip der *Suche*

# Deklarative Darstellung

Jedes Modul muß in einem Frame stecken.

**Wenn** A Modul in Frame
**dann** muß Versorgung dem Typ V**x oder** Vy
angehören.

Obige Aussage ist **keine Regel**
(im Sinn von regelbasierten Systemen),
da **“oder”** in der **Schlußfolgerung** steht!

A → Vx **or** Vy.

B → Vy **or** Vz.

C → Vx **or** Vz.

Hinzufügen von
Komponententypen D, Vv:

A → Vx **or** Vy.

B → Vv **or** Vy **or** Vz.

C → Vv **or** Vx **or** Vz.

D → Vv **or** Vy.

# Problematik regelbasierter Darstellung

**A → Vx or Vy.**

**B → Vy or Vz.**

**C → Vx or Vz.**

Versuchte direkte Umsetzung in Regeln und Vorgehensweise:

Regeln:

**Nur A Module in Frame ⇒ Vx.**

**B ⇒ Vy.**

**C ⇒ Vx .**

**AB ⇒Vy.**

**AC ⇒ Vx.**

**BC ⇒ Vz.**

**ABC ⇒ ungültig.**

1 Constraint => mehrere (viele) Regeln
Bestimmte Möglichkeiten werden ausgelassen.
Ungültige Varianten müssen explizit deklariert werden.

# Wartung der Regelbasis

Neue Constraint-Wissenbasis (1 Constraint und 2 Literale mehr):

**A → Vx or Vy.**

**B → Vv or Vy or Vz.**

**C → Vv or Vx or Vz.**

**D → Vv or Vy.**

Geänderte Regelbasis (6 neue Regeln!):

**A ⇒ Vx.**

**B ⇒ Vv.**

**C ⇒ Vv .**

**AB ⇒ Vy.**

**AC ⇒ Vx.**

**BC ⇒ Vv.**

**ABC ⇒ ungültig.**

**D ⇒ Vv.**

**DC ⇒ Vv.**

**DB ⇒ Vv.**

**DA ⇒ Vv.**

**DCB ⇒ Vv.**

**DAB ⇒ ungültig.**

# Vergleich:

- **Regeln** (propositional)

  ersetzt durch: **Constraints**
  (ausdrucksstärkeres Konzept)

- Bereichsabhängige Vorgehenweise

  ersetzt durch:

  - **allgemeines Suchverfahren**
    - korrekt bzgl. Constraints
    - vollständig bzgl. aller Konfigurationen
    - steuerbar durch Heuristiken

- Weiterentwickelte Constraintverfahren für praktische Anwendungen
  - z.B. Generative Constraints: Dynamische Vergrößerung des Constraintgraphen, wenn neue Komponenten gebraucht werden

# Generative Constraints

- Objektorientierte Darstellung der Struktur des Problembereichs:

  Komponenten
  Anschlüsse (*ports*)
  Attribute (Eigenschaften von Komponenten)

  Zusammenbau von Komponenten: über Ports

- **dynamisch**

- Grundschritte:
  - Auswahl der korrekten Komponententypen
  - Auswahl der korrekten Verbindungen
  - Entnahme der benötigten Komponenten aus dem “Katalog”