

Jürgen Ebert  
 Universität Koblenz-Landau  
<http://www.uni-koblenz.de/~ebert>

## STOR: KOMPONENTEN UND MODELLE

## ZIELE DES VORTRAGS

- × Querbezüge zwischen Computervisualistik (AGAS, LB, CG) und Informatik (AGST) herstellen
- × Werbung für CV-Qualifikationsarbeiten in der Informatik machen

## VORLESUNGEN

- × Grundlagen der Softwaretechnik (GST)
- × Programmierung (PROG)
- × Vertiefung Softwaretechnik (VST)
- × Softwarearchitektur (SA)
- × Algorithmen und Datenstrukturen (A&D)
- × A&D der Computervisualistik (A&D4CV)
- × Effiziente Graphenalgorithmen (EGA)

## KONTEXT

- × Projekt STOR *DFG 2008-2010: Paulus/Ebert* (Software Techniques for Object Recognition)

offiziell:

Ein **komponentenorientiertes** Konzept zur Nutzung von **Modellen und Wissen** bei der Objektwiedererkennung in Bildern und Bildfolgen

## BETEILIGTE

- × Jürgen Ebert, Kerstin Falkowski, Judith Haas
- × Dietrich Paulus, Peter Decker, Stefan Wirtz

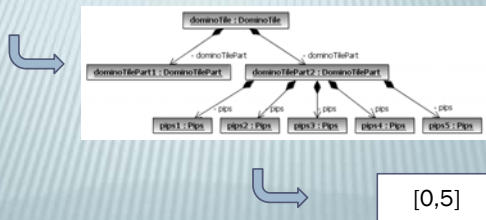
## VISION



STOR

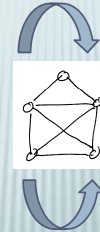
„Pförtnerhaus der Universität Koblenz-Landau“

### FALLSTUDIE 1 (2D): DOMINO-STEINE



[0,5]

### FALLSTUDIE 2 (3D): HAUSMODELLIERUNG



XML  
(CityGML)

### ÜBERBLICK

- ✘ Komponenten
  - + Grundlagen
  - + STOR-Komponenten
  - + Assemblierung
- ✘ Modelle und Wissen
  - + Grundlagen
  - + STOR-Modelle
  - + Modellbearbeitung

### KOMPONENTEN

### KOMPONENTEN: GRUNDLAGEN

### FESTSTELLUNGEN AUS „VST“

Eine **Komponente** ist ein wiederverwendbarer, abgeschlossener und vermarktbarer Softwarebaustein.

#### Konnotationen:

ausführbar, vermarktbar, wiederverwendbar, abgeschlossen, klare Schnittstelle, abgekapselt, kombinierbar, anpassbar, parametrisierbar, kommunikationsfähig, kooperationsfähig, selbstbeschreibungsfähig, mit dokumentierten Abhängigkeiten

## FESTSTELLUNGEN AUS „VST“

### Komponentenmodell:

Arten von Schnittstellen  
Möglichkeit der Anpassung  
Möglichkeit der Komposition

### Tätigkeiten:

Erstellung der Komponenten  
Assemblierung der Komponenten  
(Konfektionierung und Komposition)  
Verwendung des Systems



KOMPONENTEN: KOMPONENTENMODELL IN STOR

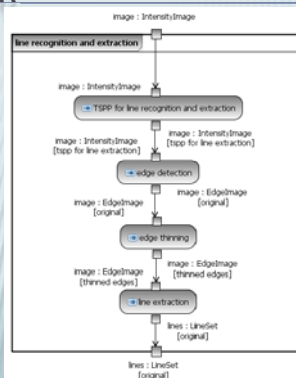
## VORÜBERLEGUNGEN

- × mittlere Granularität (etwa: Operatoren der BV)
- × wenige technisches Wissen voraussetzend
- × durch verschiedene APIs realisierbar  
(z.B. PUMA, KIPL, ITK, OpenCV)

## VORBILDER

```
nevedg -v $i.jpg $i.edg
HystLine $i.edg $i.lin
FindVert $i.lin $i.ver
hasplit -mode l $i.ver $i.seg
#hasplit -mode p -length 10 $i.ver $i.seg_l
#mergesmo $i.seg c $i.seg_l $i.seg
#smo2fig -v $i.seg $i.fig
smo2fig -arcpcol 10 -linepcol 20 $i.seg $i.fig
#smo2xml $i.seg c $i.xml
#smo2rect -arcpcol 10 -linepcol 20 $i.seg $i.xml
fig2dev -Leps $i.fig > $i.eps
xfig $i.fig&
```

## VORBILDER



## ANFORDERUNG

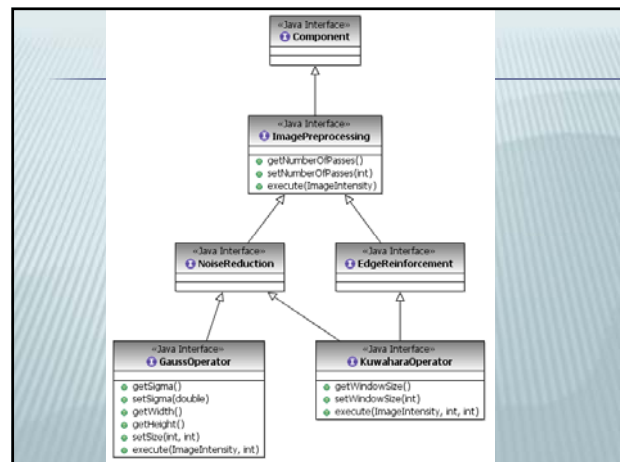
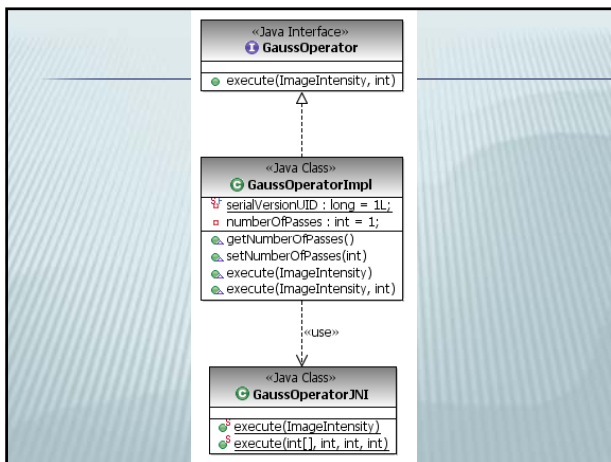
- × Das Komponentenmodell sollte (mindestens) Pipe-Filter-Architekturen aus der CV abbilden.

## FESTSTELLUNGEN AUS „PROGR“, „EGA“

- ✘ Das **Strategiepattern** empfiehlt, Verfahren als Objekt abzukapseln und auf diese Weise austauschbar und manipulierbar zu machen.
- ✘ Austauschbare Verfahren haben die gleiche Schnittstelle.


## STOR-KOMPONENTEN

- ✘ STOR-Komponenten sind Strategie-Objekte, die eine `execute()`-Methode besitzen
- ✘ Konventionen:
  - Setzen der Parameter über Getter und Setter
  - Übergabe der Argumente in der `execute()`-Methode
  - Später: Setzen des Kontexts über den Konstruktor oder eine Factory



- ✘ Die Komponenteninfrastruktur kann baut auf der Einhaltung der Konventionen auf (überprüft sie natürlich auch)
- ✘ Die Komponente rufen über JNI (Java Native Interface) die C++-Methoden der API auf.

- ✘ Bei Einhaltung der Konventionen können Komponenten wie Algorithmenbausteine beschrieben werden.



**KOMPONENTEN: SPEZIFIKATION**

## FESTSTELLUNGEN AUS „A&D“

Schritte zum Algorithmeinsatz:

- [0] Hintergrundwissen erwerben
- [1] Problem spezifizieren (MATH)
- [2] Algorithmus entwerfen (MAT/PROG)
- [3] Programm erstellen (PROG)
- [4] Lösung überprüfen

### [0] HINTERGRUNDWISSEN ERWERBEN

- × Ein mathematisches Glossar für STOR und PosE wurde erstellt:

Ein *2-D-Bild*  $I$  ist eine Abbildung von einem zweidimensionalen Ortsbereich  $2DLoc$  bestehend aus einer Untermenge des  $N^2$  in einen diskreten Wertebereich  $Val$ .

$I : 2DLoc \rightarrow Val$   
 $I : [0..h] \times [0..w] \rightarrow Val$  mit  $h, w \in N$

### [1] PROBLEM SPEZIFIZIEREN (MATH)

#### Problem specification

**Problem** "Nevatia & Babu operator application"  
**Input:** An intensity image  $I$ .  
**Output:** An edge image  $I'$  corresponding to  $I$ .

**Optional parameters**  
 number of edge direction kernels: default value is 6  
 kernel size: default value is 5

### [2] ALGORITHMUS ENTWERFEN (MAT/PROG)

The Nevatia & Babu operator calculates an edge image from an intensity image folding the intensity image values using  $n$  different  $m^d$  direction kernels. For every pixel of the resulting edge image the strongest of the  $n$  folding values and the direction of the corresponding direction kernel are set as edge likelihood and edge orientation.

### [3] PROGRAMM ERSTELLEN (PROG)

Strategieklassse NevatiaBabuOperator mit

#### Call pattern:

```
edgeImage = nevatiaBabuOperator.execute(intensityImage);
```

**Precondition:** true (intensityImage != null)

**Postcondition:** edgeImage is an edge image corresponding to the given intensity image intensityImage.

## SCHNITTSTELLEN

- × Komponenten haben i.Allg. mehrere Arten von Schnittstellen
- × Kontext (z.B. shell-Variablen, gewählte API)
- × Parameter (z.B. Sigma beim Gaußoperator)
- × Argumente (z.B. Ein-/Ausgabebilder)
- × weitere technische Parameter

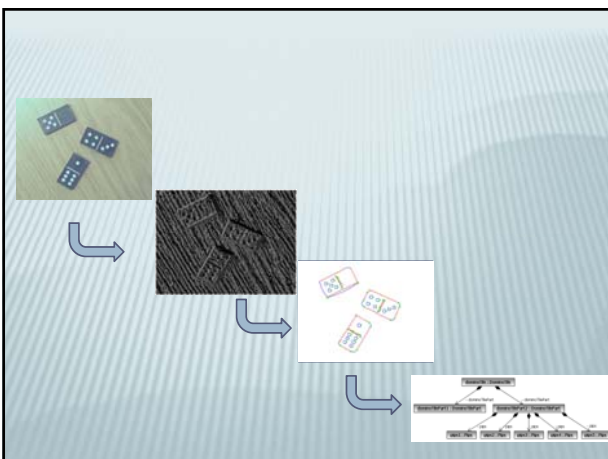
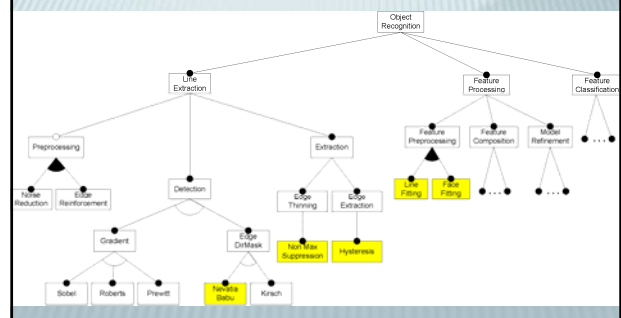
KOMPONENTEN: ASSEMBLIERUNG

## ASSEMBLIERUNG(SMÖGLICHKEITEN)

- × Strategieobjekte können in Programmen verwendet werden:
 

```
NevatiaBabuOperator nb = new ... ;
            nb.setNoOfPasses(4) ;
            Image i2 = nb.execute(i1) ;
```
- × Strategieobjekte können in einer Produktlinie zusammengefasst werden.

## PRODUKTLINIE (FODA-BAUM AUS „SA“)



MODELLE UND WISSEN



**MODELLE: GRUNDLAGEN**

### FESTSTELLUNGEN AUS „GST“

Ein **Modell** ist ein zielgerichtetes Abbild eines Systems (i.w.S.), das die Realität durch Abstraktion auf die problemrelevanten Aspekte vereinfacht und hierzu ähnliche Beobachtungen und Aussagen ermöglicht wie das Ausgangssystem.

Hier: deskriptive Modelle

### FESTSTELLUNGEN AUS „GST“

Die Art des Modells hängt ab von  
 der Domäne  
 der Anwendung  
 der Abstraktionsebene

### ABSTRAKTIONSEBENEN

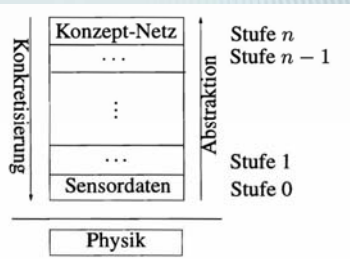
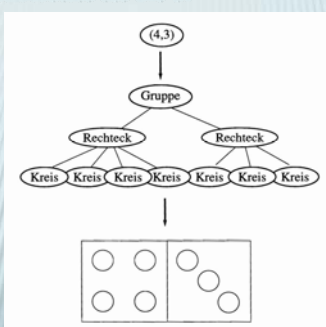


Abbildung 1: Konkretisierung und Abstraktion

### ABSTRAKTIONSEBENEN



### RAHMEN (AUFGABENTEILUNG)

- ✗ AGAS führt den BV-Erkennungsprozess durch.
- ✗ AGST erstellt und bearbeitet das Modell.



## MODELLIERUNGSWELTEN

- × „technological spaces“
  - Matrizen
  - RDF-Tripels
  - XML
  - Listengeflechte
  - Graphen

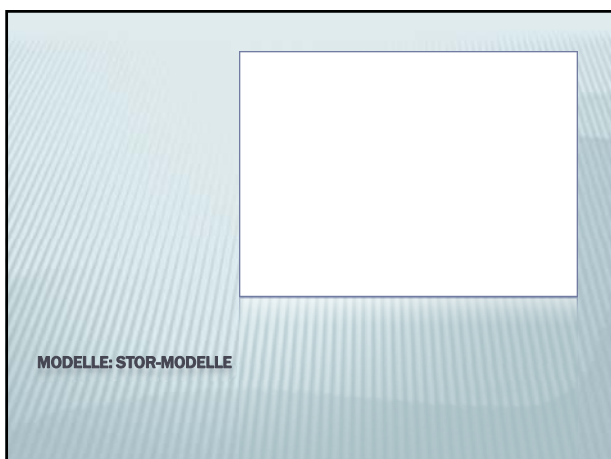
## FESTSTELLUNGEN „EGA“

**Graphen** sind ein vielseitiges Konzept,

- Modelle zu spezifizieren  
(weil sie mathematisch sind),
- Modelle anschaulich zu machen  
(weil sie visualisierbar sind),
- Modell zu speichern  
(weil es effiziente Speicherformen gibt),
- Modelle zu untersuchen  
(weil sie anfragbar sind) und
- Modelle zu transformieren  
(weil es effiziente Algorithmen gibt).

## FESTSTELLUNGEN „EGA“

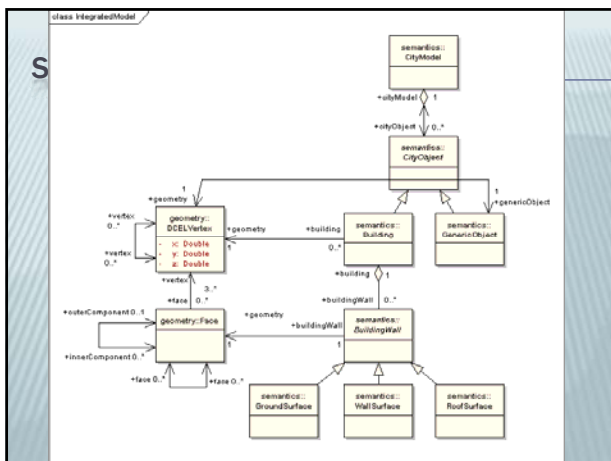
Mit der **TGraphen-Technologie** haben wir eine bewährte (in Java realisierte) Basis für die Entwicklung von Komponenten auf Graphen.



## MODELLIERUNG IN STOR

**STOR-Modelle** sind TGraphen.

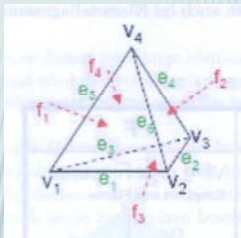
- Sie enthalten:
- die Topologie (Struktur)
  - die Geometrie (Attribute)
  - die Semantik (Bezug zu sem. Modell)
  - die Textur (Referenz-Attribute)



## TOPOLOGIE

Die **Struktur** wird gespeichert als Oberflächenmodell (boundary representation) genauer:  
als v-e-f-Graph (vertex, edge, face)  
im Stile der DCEL-(doubly-connected-edge-list)-Struktur.

## V-E-F-MODELL



## GEOMETRIE

Die **Geometrie** findet sich in Attributen wieder, hauptsächlich als Positionsattribute der Knoten.

## WISSEN, SEMANTIK

Die **Semantik** wird durch eine „light-weight“-Ontologie ebenfalls als Graph abgelegt.

(Die Anfragesprache GReQL auf TGraphen erlaubt effiziente Anfragen.)

Die Ontologie wird (initial) aus City-GML übernommen.

## ERSCHEINUNGSBILD (TEXTUR)

Zusätzliche **Textur**information wird als Java-Objekt an den Flächen (faces) aufbewahrt.

## INTEGRATION

Die vier Hauptaspekte der Modelle sind in einem gemeinsamen Graphen integriert.

## MODELLBEARBEITUNG

## VERWENDUNG DES MODELLS

- Modell **konsolidieren**  
z.B. aus Punkten das optimale ebene Polygon
- Modell **vervollständigen**  
z.B. aus zwei Seiten Quader rekonstruieren
- Modell **analysieren**  
z.B. oberster/unterster Punkt, Volumen, Konvexität
- Modell **transformieren**  
z.B. Triangulierung

## VERWENDUNG DES MODELLS

- Modell **annotieren**  
z.B. Fenster in Wänden erkennen
- Modell **klassifizieren**  
z.B. Matching mit Modellen bekannter Objekte
- Modell **exportieren**  
z.B. Darstellungen in verschiedenen (XML-)Formaten (CityGML, COLLADA, ...)

## ZUSAMMENFASSUNG

## ZUSAMMENFASSUNG

- Im Projekt STOR werden
- +effiziente Algorithmen aus der Bildverarbeitung und
  - +effiziente Algorithmen der graph-basierten Modellbearbeitung
- zu einem gemeinsamen komponenten-basierten BV-System zusammengefügt.

