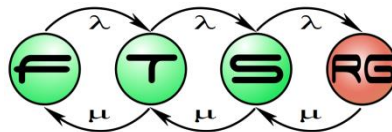


Ausführung der Modelle, Codegenerierung

Budapest University of Technology and Economics
Fault Tolerant Systems Research Group



Inhalt

**Die Funktionen einer
Modellierungsumgebung**

Kodegenerierung

**Eclipse-Basierte
Modellierungsumgebungen**

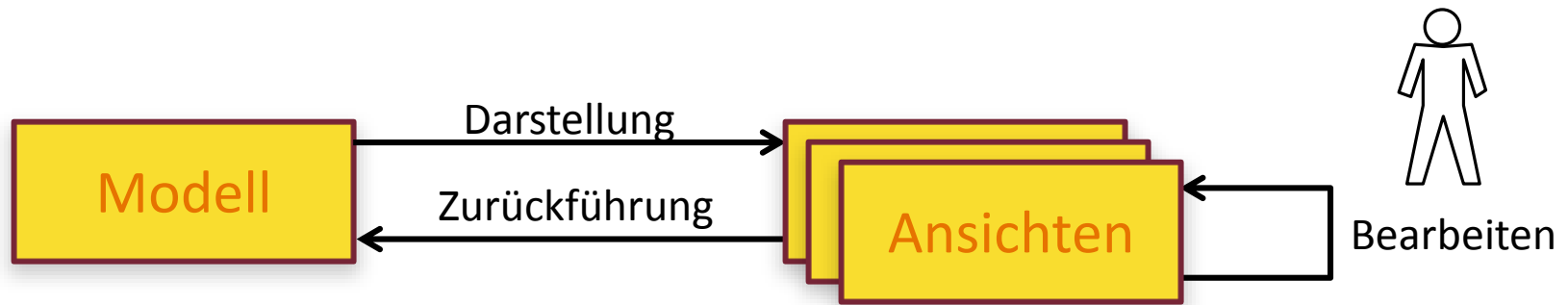
Funktionen

Kodegenerierung

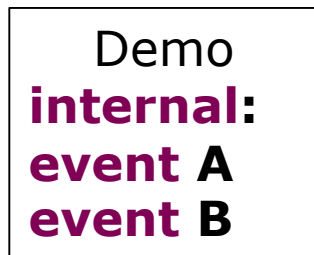
Eclipse-basierte Umgebungen

DIE FUNKTIONEN EINER MODELLIERUNGSUMGEBUNG

Funktionen einer Modellierungsumgebung



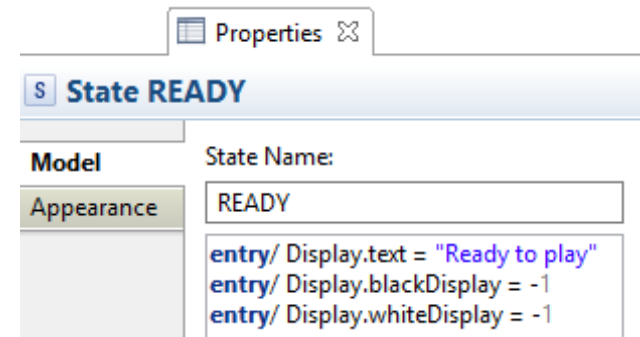
Textuell



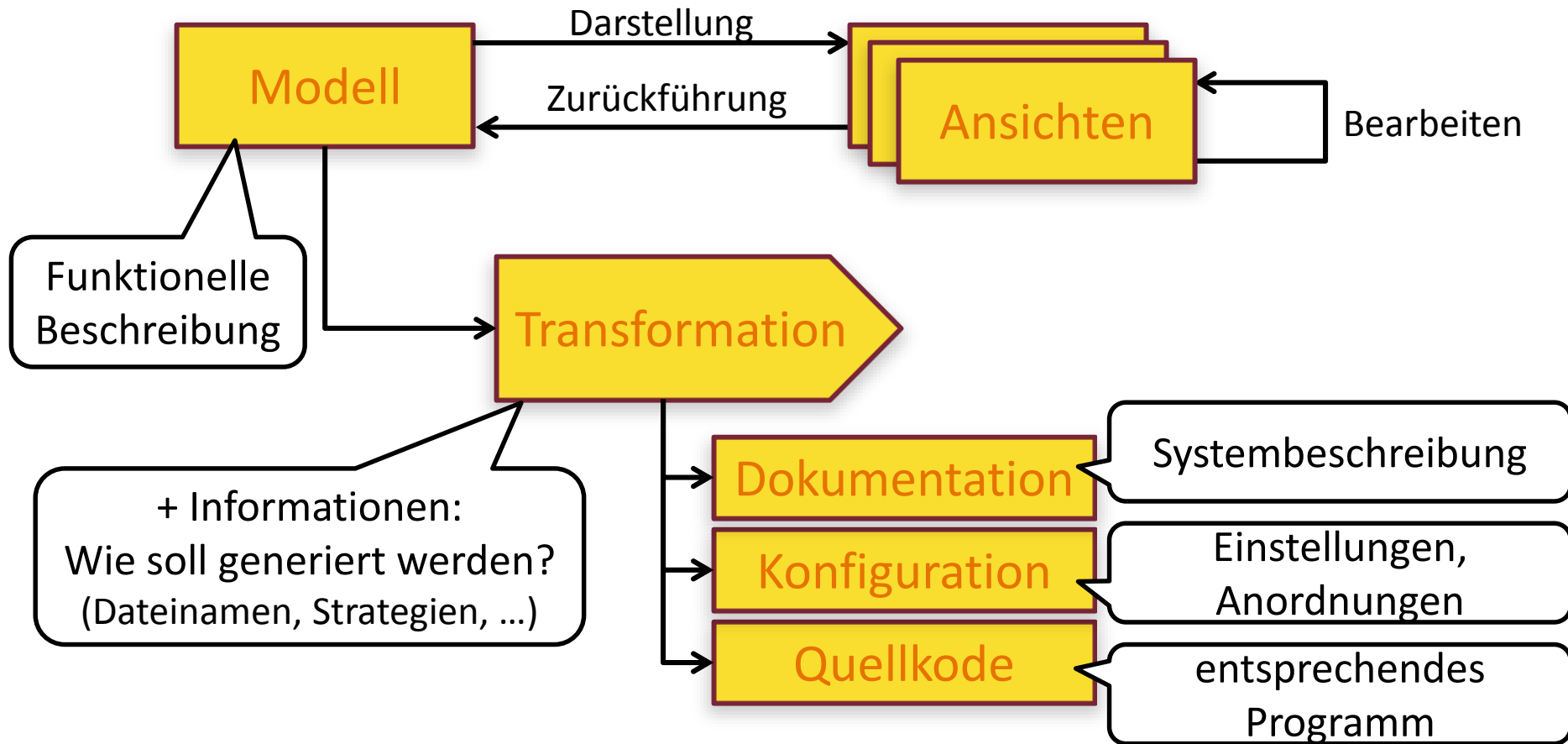
Graphisch



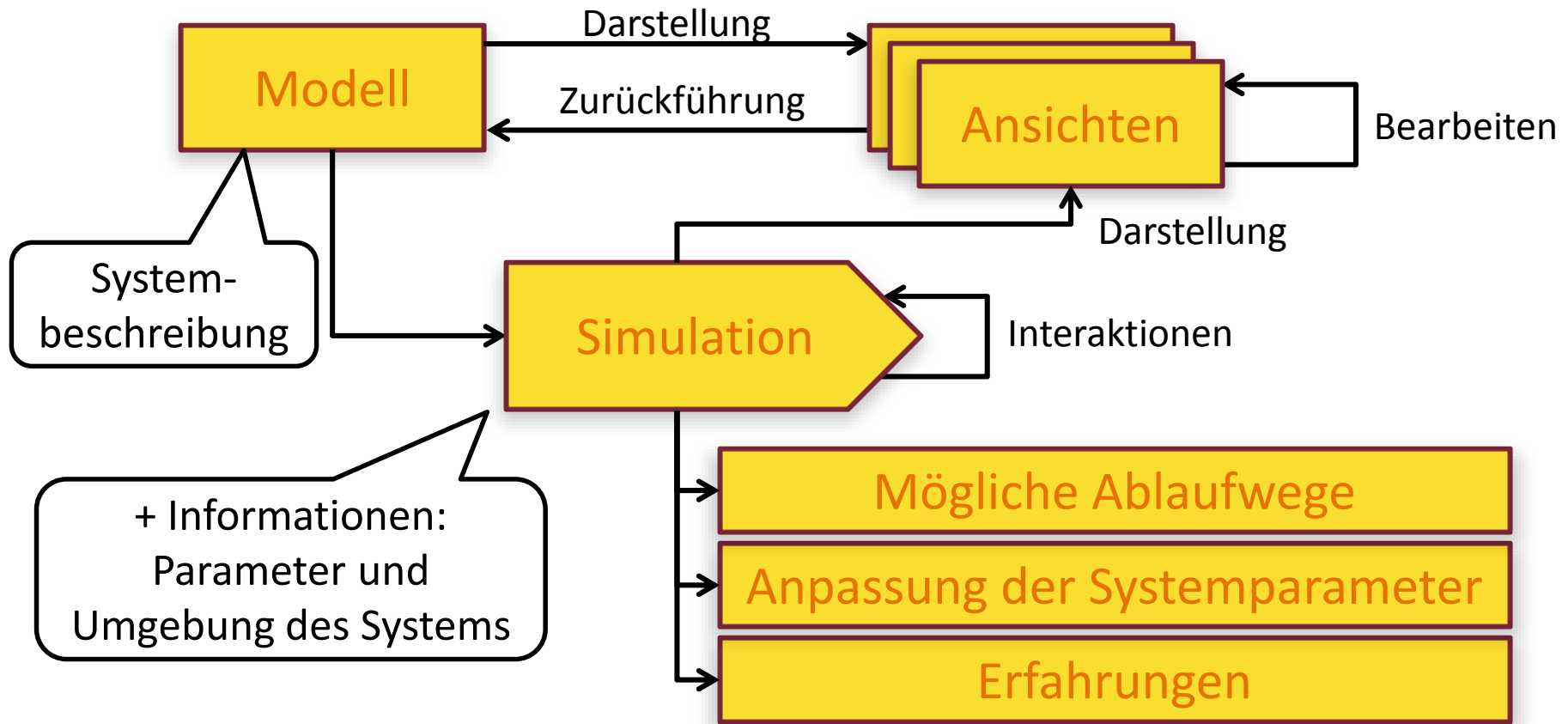
Andere Umgebungen



Funktionen einer Modellierungsumgebung



Funktionen einer Modellierungsumgebung

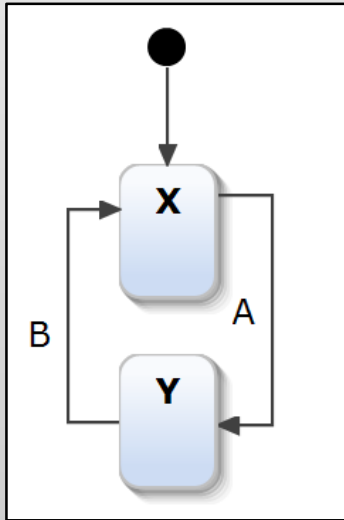


Modellierungsfunktionen von Yakindu

Konkrete Syntax

(für den Benutzer)

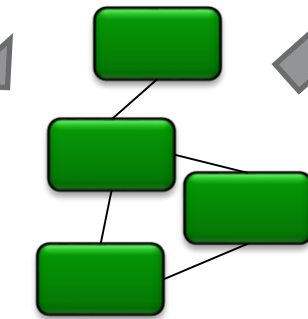
Graphisch:



Textuell:

```
Demo
internal:
event A
event B
```

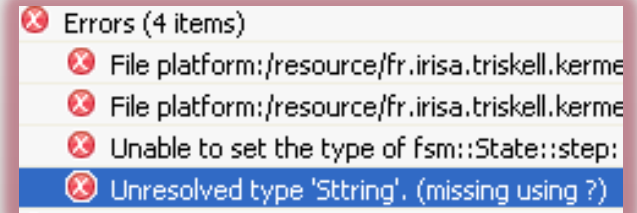
Syntax → Semantik



Modell
(Abstrakte Syntax)

Modellierungsfunktionen

Modellüberprüfung



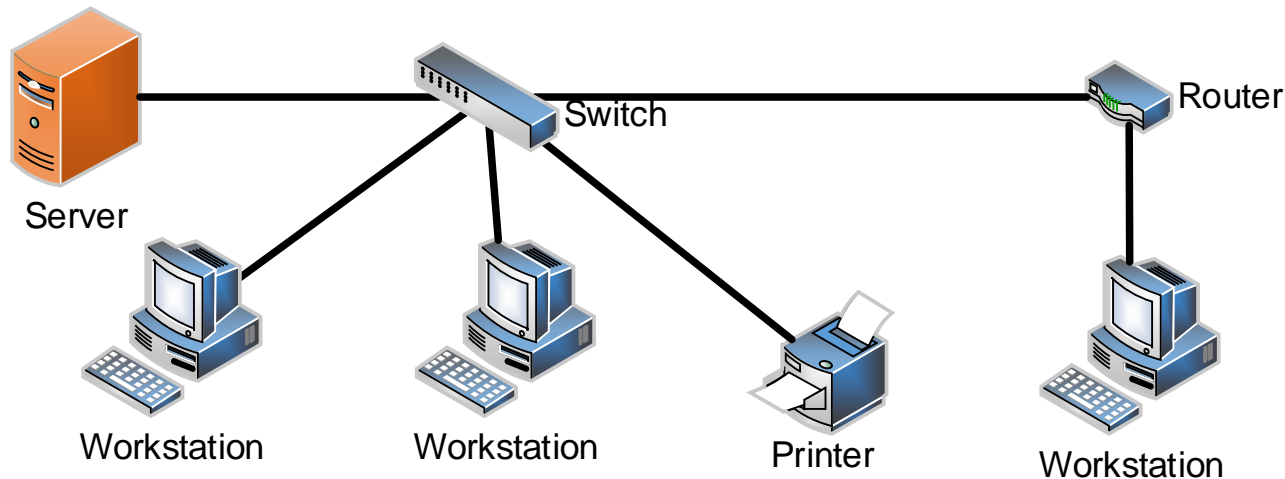
Kodegenerierung

```
</membership>
<profile defaultProvider="Sitefinity">
  <providers>
    <clear/>
    <add name="Sitefinity" connectionS
  </providers>
  <properties>
    <add name="FirstName"/>
    <add name="LastName"/>
    <!-- SNP specific properties -->
    <add name="NickName" />
    <add name="Gender" />
```

(Quellencode, Dokumentation,
Konfiguration)

Abstrakte Syntax

- **Definition:** Strukturelles Modell des zu editierenden Systems
- Wird von der Modellierungsumgebung verwaltet
- Zur Erinnerung: Strukturelles Modell = **Graph**
 - **Graph von Knoten, Kanten, Eigenschaften**

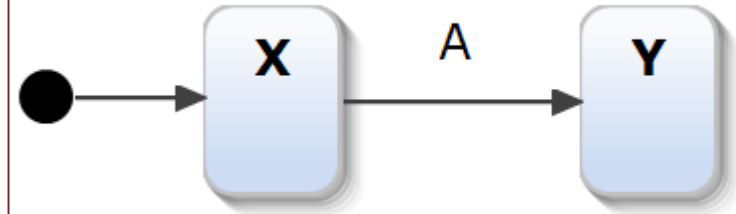


Beispiel – Abstrakte Syntax: Yakindu

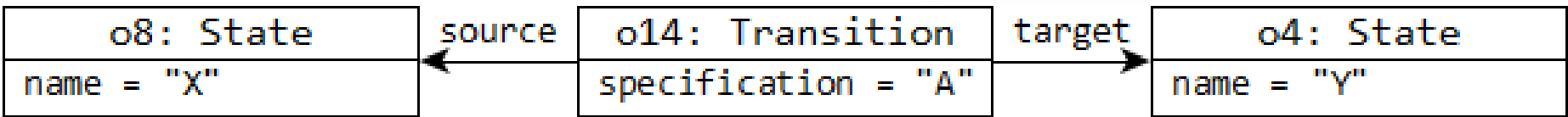
Frage:

Wie würden wir eine Modelleirungsumgebung implementieren?

Beispiel: Yakindu Modell



Abstrakte Syntax



Beispiel – Abstrakte Syntax: Yakindu

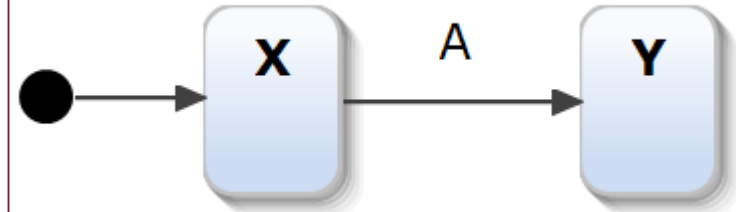
Frage:

Wie würden wir eine Modelleirungsumgebung implementieren?

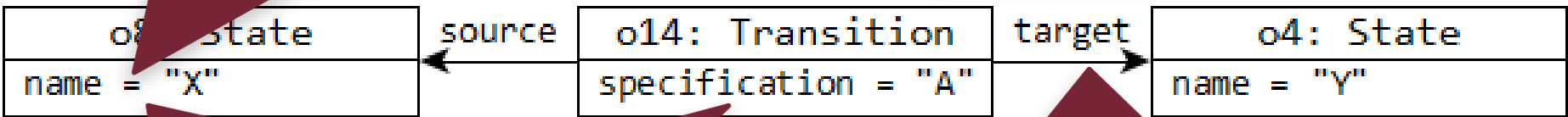
Namen werden als String gespeichert

```
name = "X"
```

Beispiel: Yakindu Modell



Abstrakte Syntax



Modellelemente als Objekte

Relationen als Referenzen

Antwort: Es ist ein einfaches objekt-orientiertes Program mit Extrafunktionen

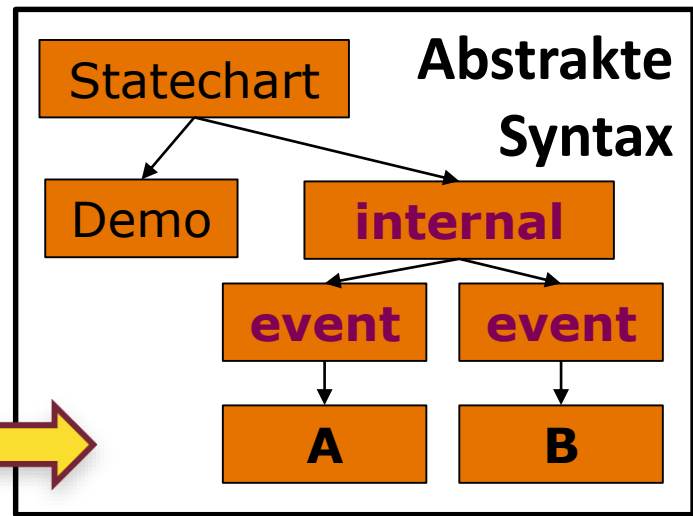
Konkrete Syntax: Textuelle Syntax

- **Ziel:** Repräsentation \Leftrightarrow das Modell dahinter
- Textuelle Syntax (z.B. Programme)
 - Aufgabe: Text \rightarrow Modell
 - Regelbasiert (sonst wird es schwierig!)

Demo
internal:
event A
event B

Grammatik

```
<Statechart> ::= <Name> <Interface>*  
<Interface> ::= ("internal" | <Name>)  
                ":" <Event>*  
<Event>      ::= "event" <Name>  
<Name>      ::= ...
```



Mit geeigneten Technologien (z.B. Xtext) kann jeder eine eigene Modellierungs-/Programmierungssprache implementieren!

Gegenbeispiel für „geeigneten Methoden“

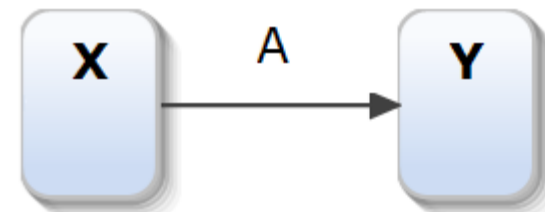
- **Aufgabe:** entscheiden ob ein Text eine Emailadresse ist

```
(?:\r\n|\t)*(?:\s|<@;\.\[\ \000-\031]+(?:\r\n|\t)|\Z|(?=[\s|<@;\.\[\ \000-\031]+(?:\r\n|\t)|\Z|(?=[\s|<@;\.\[\ \000-\031]+(?:\r\n|\t)|\Z|(?=[\s|<@;\.\[\ \000-\031]+(?:\r\n|\t)|\Z|(?=[\s|<@;\.\[\ \000-\031]+(?:\r\n|\t)|\Z|(?=[\s|<@;\.\[\ \000-\031]+(?:\r\n|\t)|\Z|(?=[\s|<@;\.\[\ \000-\031]+(?:\r\n|\t)|\Z|(?=[\s|<@;\.\[\ \000-\031]+(?:\r\n|\t)|\Z|(?=[\s|<@;\.\[\ \000-\031]+(?:\r\n|\t)|\Z|(?=[\s|<@;\.\[\ \000-\031]+(?:\r\n|\t)|\Z|(?=[\s|<@;\.\[\ \000-\031]+(?:\r\n|\t)|\Z|(?=[\s|<@;\.\[\ \000-\031]+(?:\r\n|\t)|\Z|(?=[\s|<@;\.\[\ \000-\031]+(?:\r\n|\t)|\Z|(?=[\s|<@;\.\[\ \000-\031]+(?:\r\n|\t)|\Z|(?=[\s|<@;\.\[\ \000-\031]+(?:\r\n|\t)|\Z|(?=[\s|<@;\.\[\ \000-\031]+(?:\r\n|\t)|\Z|(?=[\s|<@;\.\[\ \000-\031]+(?:\r\n|\t)|\Z|(?=[\s|<@;\.\[\ \000-\031]+(?:\r\n|\t)|\Z|(?=[\s|<@;\.\[\ \000-\031]+(?:\r\n|\t)|\Z|(?=[\s|<@;\.\[\ \000-\031+...)
```

- Eine einzige Adresse oder eine Liste
- Wegspezifische Informationen können erscheinen
- Optional können auch Namen geschrieben werden
- Hierarchischer Aufbau mit '.' Separator

Konkrete Syntax: Graphische Syntax

- **Ziel:** Repräsentation \leftrightarrow das Modell
- Graphische Syntax (z.B. Diagramm)
 - Aufgabe: Diagramm \rightarrow Modell
 - Übersichtlicher, schwieriger zu schreiben, regelbasiert



Kondition auf dem Modell

Id*:

Domain Class*:

Semantic Candidates Expression:

Anlegen des Diagrammelementes

Label Alignment: Left Center Right

Label Expression:

Label Position:

Color*:

Label Color*:

Border Color*:

Kondition erfüllt \rightarrow Diagrammelement wird angelegt
Diagramm wird geändert \rightarrow Modell ändert sich auch

Konkrete Syntax: Graphische Syntax

Ergebnis:

The screenshot shows a modeling tool interface. At the top is a toolbar with various icons for editing and navigation. Below the toolbar is a diagram area containing two states: 'X' (a solid blue square) and 'Y' (a blue square with a dashed black border). Below the diagram area is a 'Properties' panel. The panel has tabs for 'Properties' and 'Problems'. The 'Properties' tab is active, showing a table for 'State Y'.

Property	Value
State Y	
Composite	<input checked="" type="checkbox"/> false
Documentation	<input type="checkbox"/>
Incoming Transitions	→ X -> Y (A)
Leaf	<input checked="" type="checkbox"/> true
Name	<input type="checkbox"/> Y

Mit geeigneten Technologien (z.B. Sirius) kann **jeder** eine **eigene** Modellierungs-/Programmierungssprache implementieren!

Modellvalidierung: Syntaktische Überprüfung

- Syntaktische Überprüfung: die Modellierungsumgebungen verbinden die logisch zusammengehörenden Elemente

Schnittstellendeklaration:

```
var clock: integer = 60
```

Verwendung im Modell:

```
after 1 s [clock>0]/ clock-=1
```

- Syntaxgesteuerte Editoren

- Fehler während Bearbeitung → **Couldn't resolve reference**
- Moderne Umgebungen: Vorschlagen der Kandidaten

- Kode+Diagramm gemeinsam

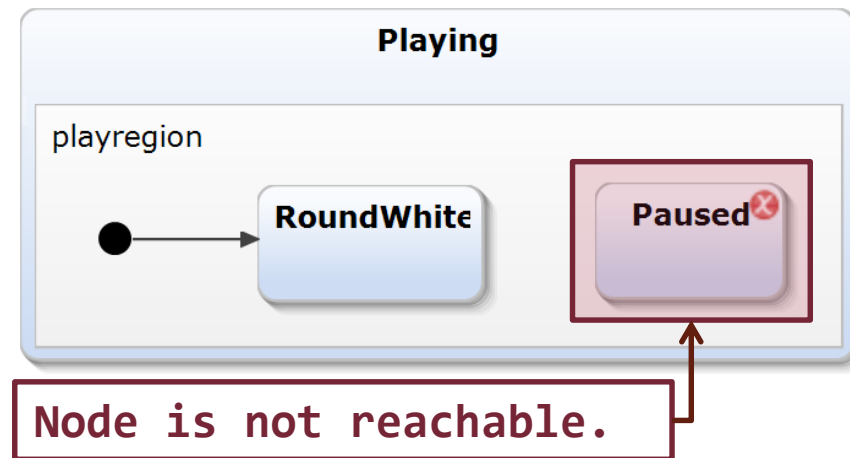
```
after 1 s [clock>0]/ clock-=1
```



- Programmieren: **fehlerhaft** während der Bearbeitung
- Modellieren: **korrekt** während der Bearbeitung

Modellvalidierung : Strukturelle Überprüfung

- Strukturelle Überprüfung: Untersuchung des Modellgraphen
- Suchen nach Fehlermuster während der Bearbeitung
- z.B. unerreichbarer Zustand:



- Weitere Überprüfungen: fehlender Anfangszustand, Verklemmung, fehlerhafte Wertzuordnungen, etc.

Funktionen

Kodegenerierung

Eclipse-basierte Umgebungen

KODEGENERIERUNG

Aufgaben der Codegenerierung

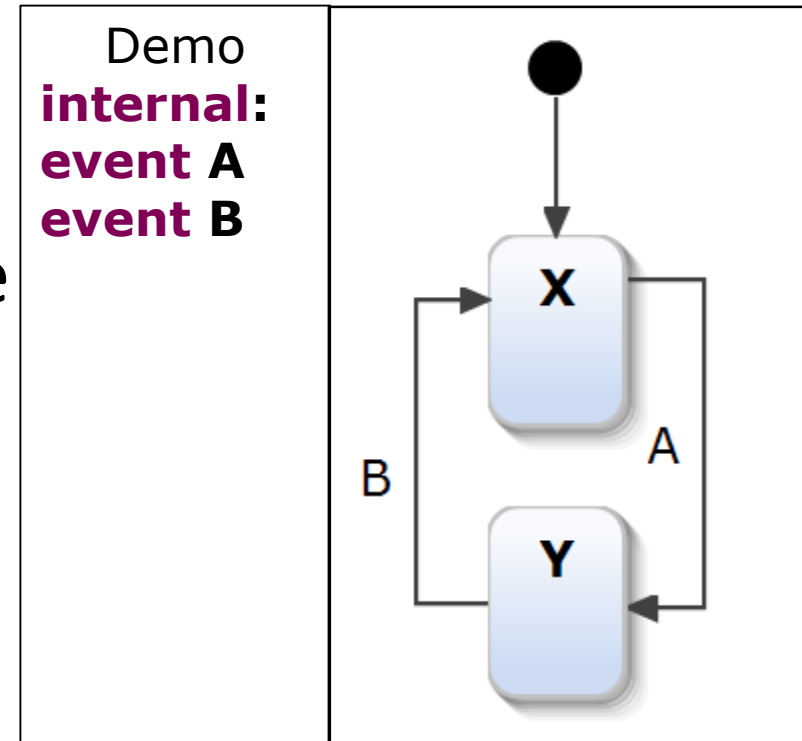
- **Aufgabe:** automatische Generierung von sich entsprechend verhaltenen Modellen
- Mehrere Möglichkeiten → Entwurfsentscheidung
 - **Interpretiert:** Modell wird eingelesen und ausgeführt
 - oder **Programmcode:** Quellencode wird generiert
 - **Programmiersprachen:** Java, C, C++, ...
 - **Optimierung:** Speicher vs. Prozessor
Beobachtbarkeit vs. Performanz
 - Wie kann der generierte Code mit eigenen Code ergänzt werden?
- **Codegenerator:** parametrisierbar + ergänzbar

Kodgeneratoren – ein Beispiel

- **Aufgabe:**
Generiere C-Kode für eine Yakindu Zustandsmaschine

- Schreibe eine Funktion, die:
→ ein Modellobjekt nimmt
← einen Text zurückgibt

- Der Text wird in eine Datei „Demo.c“ geschrieben
- Es wird von einem Compiler übersetzt



Vorlagenbasierter Codegenerator (Xtend)

- Ziel: Zustände → Enum

Die Ausgabe wird in ein char* gesammelt, anstatt %s werden die Namen X,Y geschrieben

- Lösung: C program

```
sprintf(result,  
        "enum states {\n\tState%s,\n\tState%s\n};",  
        state1->name,  
        state2->name);
```

```
enum states {  
    StateX,  
    StateY  
};
```

😊 Funktioniert gleich!
☹ Unlesbar

- Vorlage (Xtend):

```
'''  
enum states {  
    State«state1.name»,  
    State«state2.name»  
}'''
```

Die variablen Stellen werden angegeben

Die Vorlage wird geschrieben

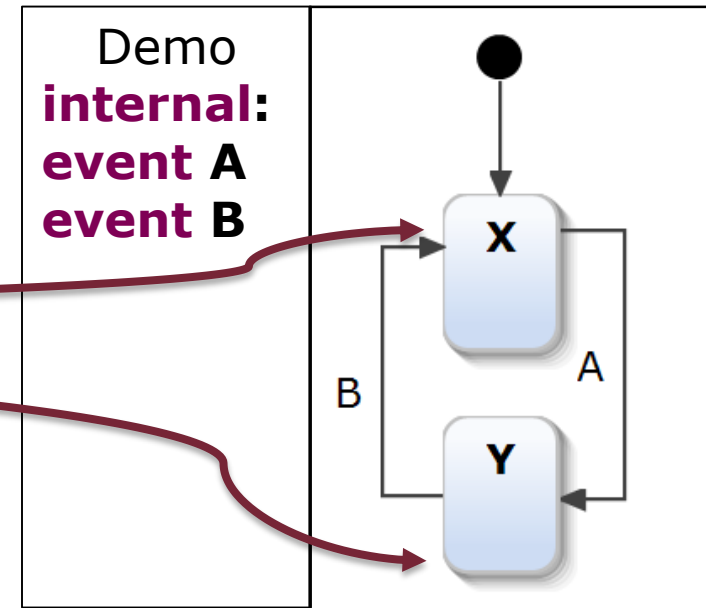
😊 Einfacher zu schreiben
😊 Übersichtlich
😊 Leicht zu modifizieren
☹ +1 Technologie

Kodegenerator Beispiel – Zustände

■ Erwarteter C-Kode:

```
//States of the statemachine  
enum states {  
    StateX,  
    StateY  
};
```

Mögliche Zustände:
Als Enum aufgezählt



■ Vorlage:

```
//States of the statemachine  
enum states {  
«FOR state : states»  
    State«state.name»,  
«ENDFOR»  
};
```

1. Wir iterieren durch alle Zustände
2. Ihre Namen werden mit Komma
getrennt ausgeschrieben:

State«Name» Pl: StateX

Kodegenerator Beispiel – Anfangszustand

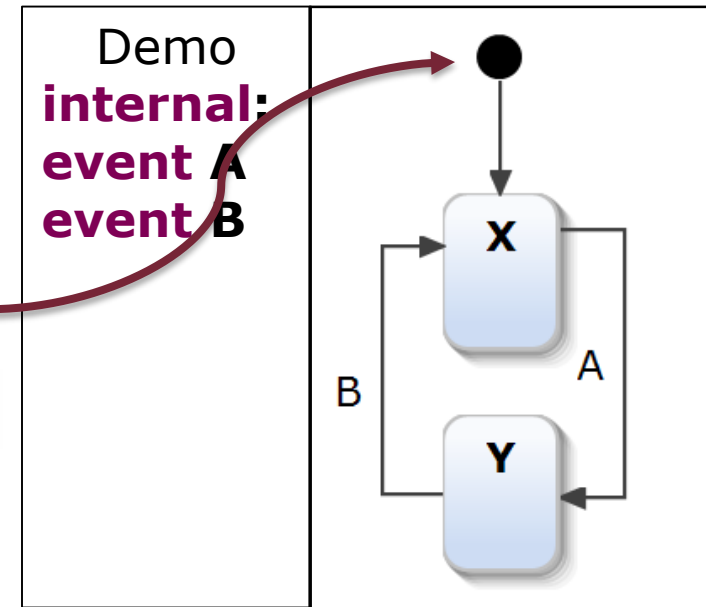
■ Erwarteter C-Kode:

```
// The current state  
// First = entry state.  
enum states actualState = StateX
```

Aktueller Zustand = Anfangszustand

■ Vorlage:

```
// The current state  
// First = entry state.  
enum states actualState = State«findEntry(states).name»
```



1. Wir suchen das Anfangselement
2. Sein Name wird ausgeschrieben

Kodegenerator Beispiel – Zustandsübergänge

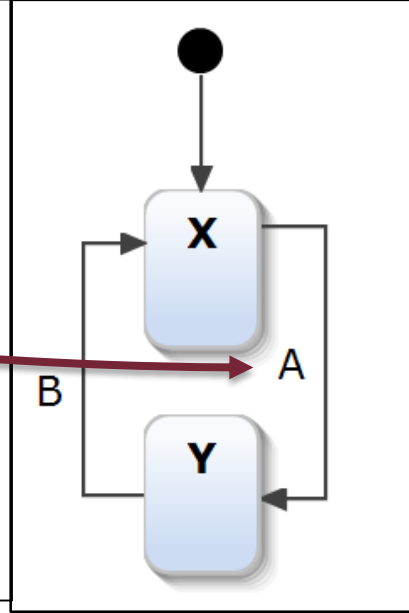
■ Erwarteter C-Kode:

```
// Execute "A" event  
void doA{  
  switch(actualState) {  
    case StateX:  
      actualState = StateY;  
      break;  
    case StateY:  
      break;  
  }  
}
```

A / X → Y

A / -

Demo
internal:
event A
event B



■ Vorlage (skizziert):

1. Für jedes Ereignis eine Funktion `do«NameDesEreignisses»`
2. Der Funktionsinhalt entspricht den Transitionen

**Für eine (einfache)
Zustandsmaschine ist der
Kodegenerator nur soviel!**

Kodegenerator – Zusammenfassung

- Kodegenerierung = Übersetzer
- Gleiche Schritte:



- Lösung in der Sprache des Problems: **Produktivität ++**
- Viele langweilige, komplizierte Kodierungsarbeit automatisiert **Leistung ++**
- Überprüfung in der Sprache des Problems: **Zuverlässigkeit ++**
- Projekte an unserem Lehrstuhl: **bis zu 95% generierter Code**

Funktionen

Kodegenerierung

Eclipse-basierte Umgebungen

ECLIPSE-BASIERTE MODELLIERUNGSUMGEBUNGEN

EclipseLink: Java Objekte ↔ Datenbanken

- **Aufgabe:** Speicherung der Objekte in einer Datenbank
- **Lösung:** Modellierung der Verbindungen mit Annotationen, automatischer Verwaltung

```
@Entity
@Table(name="Car_Table")
public class Car {
    @Id
    long identifier;
    String numberPlate;
    Person owner;
}
```



Car_Table		
Identifier	NumberPlate	Ovner_ID
1	EGG153	17
2	KSS122	55
3	KAF984	56



XMind: Mindmap Editor

The screenshot displays the XMind 2007 application window. The main workspace shows a mind map with a central node 'Sitemap' and several branches:

- Home**: What's New
- Products**: Overview, Screenshots, Tech Notes, Downloads, Purchase
- Features**:
 1. Structure for human brains
 2. Rich expression
 3. Boundary and Relationship
 4. Markers and Legend
 5. Strong auto-numbering
 6. Customized Templates
 7. Open Eclipse Plug-in Platform
 8. Filter and delamination
 9. Powerful workbook and assoc
 10. Seamless integration with of
 11. Import other mindmaps to sa
- Support**: FAQ, Install, Forum
- About**: Pictures, Map, Privacy
- Solutions**: Individual (Presentation, Decision Maker, Writing Helper, Time Management), Education (Innovation, Teaching, Key Notes), Teamwork (Brainstorming)
- Analytics**: Pageviews, Visits, P/V
- Next Version**: Solutions, Tech Notes, Downloads, Cases
- Four Special Features**: (with a smiley icon)

The interface includes a menu bar (File, Edit, View, Insert, Modify, Map, Window, Help), a toolbar, and several panels on the right: Outline (showing a tree view of the map), Notes, Markers (with a filter and 11 markers), and Filter. The bottom of the window shows a 'Properties' and 'Templates' section with six template options: Default Template, XMIND Classic, XMIND Simple, XMIND Business, XMIND Academese, and XMIND Comic. The status bar at the bottom indicates '1 topic (" Sitemap ") selected.'

Bonita: Modellierung von Geschäftsprozessen

The screenshot displays the Bonita Studio interface for modeling a business process. The main workspace shows a BPMN diagram for the process 'Fibra Spenta Richiesta (1.1)'. The process starts with a start event leading to the 'Inserimento Richiesta' task. This is followed by 'Mail Richiesta Studio Fattibilità', 'Richiesta Fattibilità Fibra', and 'Mail Studio Fattibilità Completato'. A decision gateway 'Gate1' follows, with two outgoing flows: 'Fattibilità KO' leading to 'Chiusura Fattibilità KO' and 'Fattibilità OK' leading to 'Produzione Offerta Commerciale'. The 'Produzione Offerta Commerciale' task leads to 'Attesa Risposta Cliente', which then leads to another decision gateway 'Gate3'. From 'Gate3', one flow leads to 'Chiusura Rifiuto Cliente' and another leads to 'Chiusura Fattibilità KO'. The 'Chiusura Rifiuto Cliente' task leads to 'Cliente Rifiuta', which then leads to 'Chiusura Fattibilità KO'. The 'Chiusura Fattibilità KO' task leads to a final end event labeled 'Fine Richiesta NON Fattibile'. The interface includes a menu bar (Process, Edit, Run, View, Extensions, Look'n'feels, Data types, Connectors, Contexts, Forms, BAM, Simulation, Repository, Help), a toolbar with icons for New, Open, Save, Print, Import, Export, Copy, Paste, Run, Debug, User XP, Application Developer, User guidance, Preferences, Help, and Welcome, and a left sidebar with a palette of BPMN elements and a zoom tool. At the bottom, there is an 'Overview' window showing a smaller version of the diagram and a 'Properties' window for the selected pool 'Fibra Spenta Richiesta', showing its name and version (1.1).

Kalypso: Hydrologische Planung + Simulation

Hochwasser Vorhersage Team Synchronizing Kalypso Modeler

Navigator Style Editor

Style: Subcatchments

Regel: + + -

Subcatchments Subcatchments-Numt

Titel: Subcatchments

MinDenom: 0.0

MaxDenom: 77277.838217127

Symbolizer: Polygo Ort

Legende:

Polygon Line Point Text

Füllung

Fill-Farbe:

Fill-Onaricity:

Outline

- + Legende
- + Knoten
- + VGewaesser
- + KMGewaesser - aktiv
- + RHBGewaesser
- Teilgebiete
- + Subcatchments
- Hydrotupe
- + hydrotop

modell.gmv

NaModell0

- CatchmentCollectionMember
 - CatchmentCollection1
 - catchmentMember
 - Catchment100
 - Catchment101
 - Catchment102
 - bodenkorrekturmember
 - grundwasserabflussMember
 - entwaesserungsStrangMemb
 - KMChannel101
 - Catchment103
 - Catchment104
 - Catchment105
 - Catchment601
 - Catchment603
 - Catchment604
 - Catchment605
 - Catchment606
 - Catchment607
 - Catchment608

*Tabell... Tabelle... *Modell... *Karten... »1

*Tabelle_Teilgebiete.gtt

TG-Nummer (in...	Versiegelu...	Anstie...	TG-Flaeche (fla...	Faktor ...	Anfangsinhalt...
100	0.188		1123080		
101	0.0080				
102					

<http://www.kalypso-simulation-platform.org>