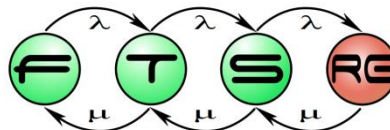


Modellierungsumgebungen, Kodegenerierung

**Budapest University of Technology and Economics
Fault Tolerant Systems Research Group**



Inhalt

Funktionen



Modellierungsumgebungen



Kodegenerierung

Funktionen

Umgebungen

Kodegenerierung

DIE FUNKTIONEN EINER MODELLIERUNGSUMGEBUNG

Die Semantik des Codes

Beispiel: Hausaufgabe (Schachuhr)

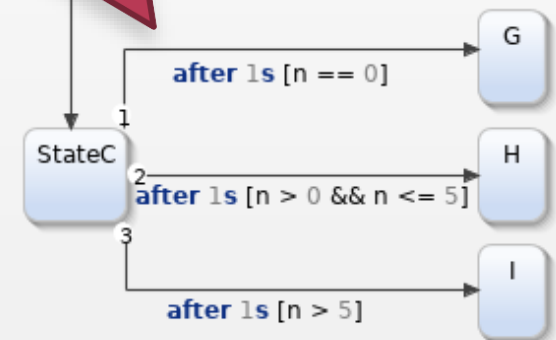
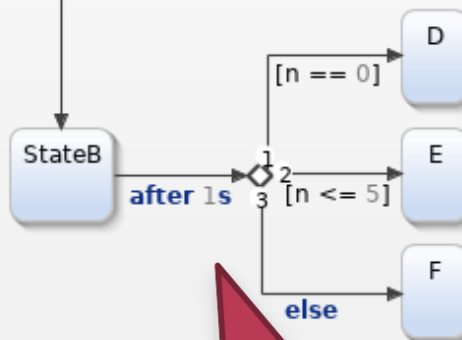
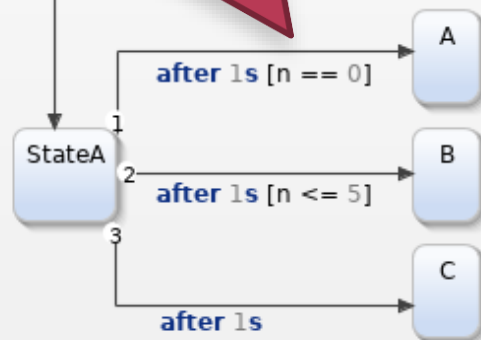
- „funktioniert wegen eines Programmierfehlers unter Umständen nicht richtig, deshalb zeigt dann die Bewertung falsche Fehler an“
- (Teil-)Aufgabe: Die richtige Behandlung von verschiedenen Übergängen, die
 - von dem gleichen Ereignis ausgelöst werden,
 - Miteinander überlappende Wächterkriterien haben ($x=1$, $x<5$),

Mögliche Lösungen

main region

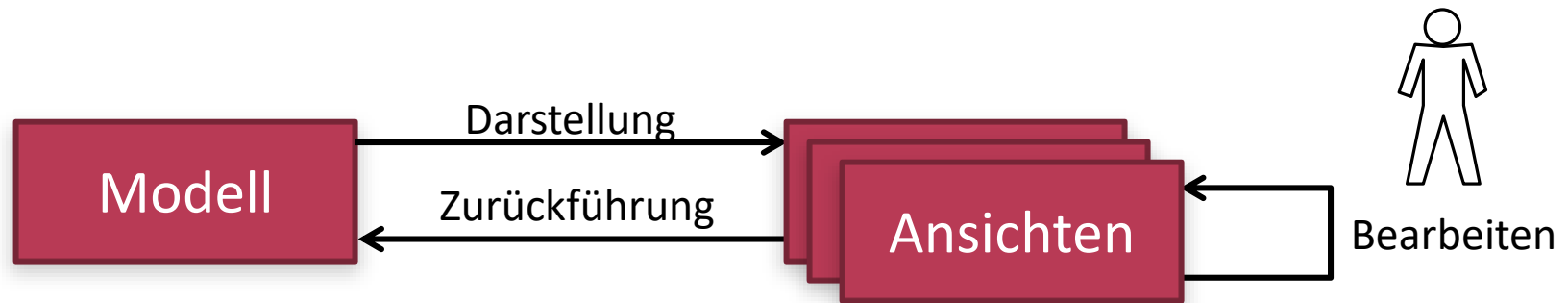
Prioritäten → Beeinflusst die Auswertung der Wächterkriterien!

Gegenseitig ausschliessende Wächterkriterien

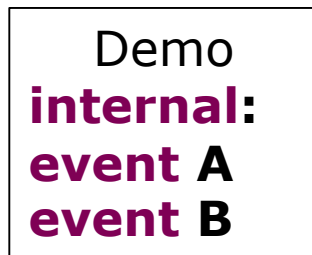


Choice + default Zweig

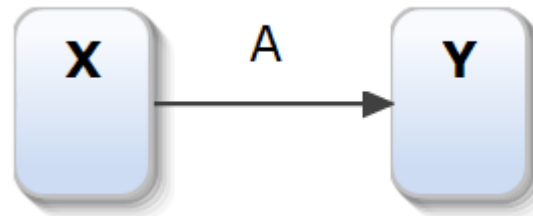
Funktionen einer Modellierungsumgebung



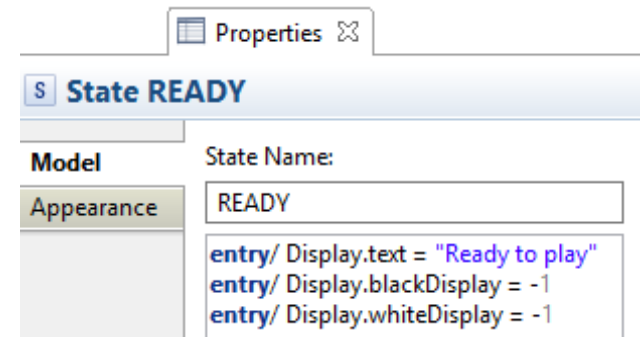
Textuell



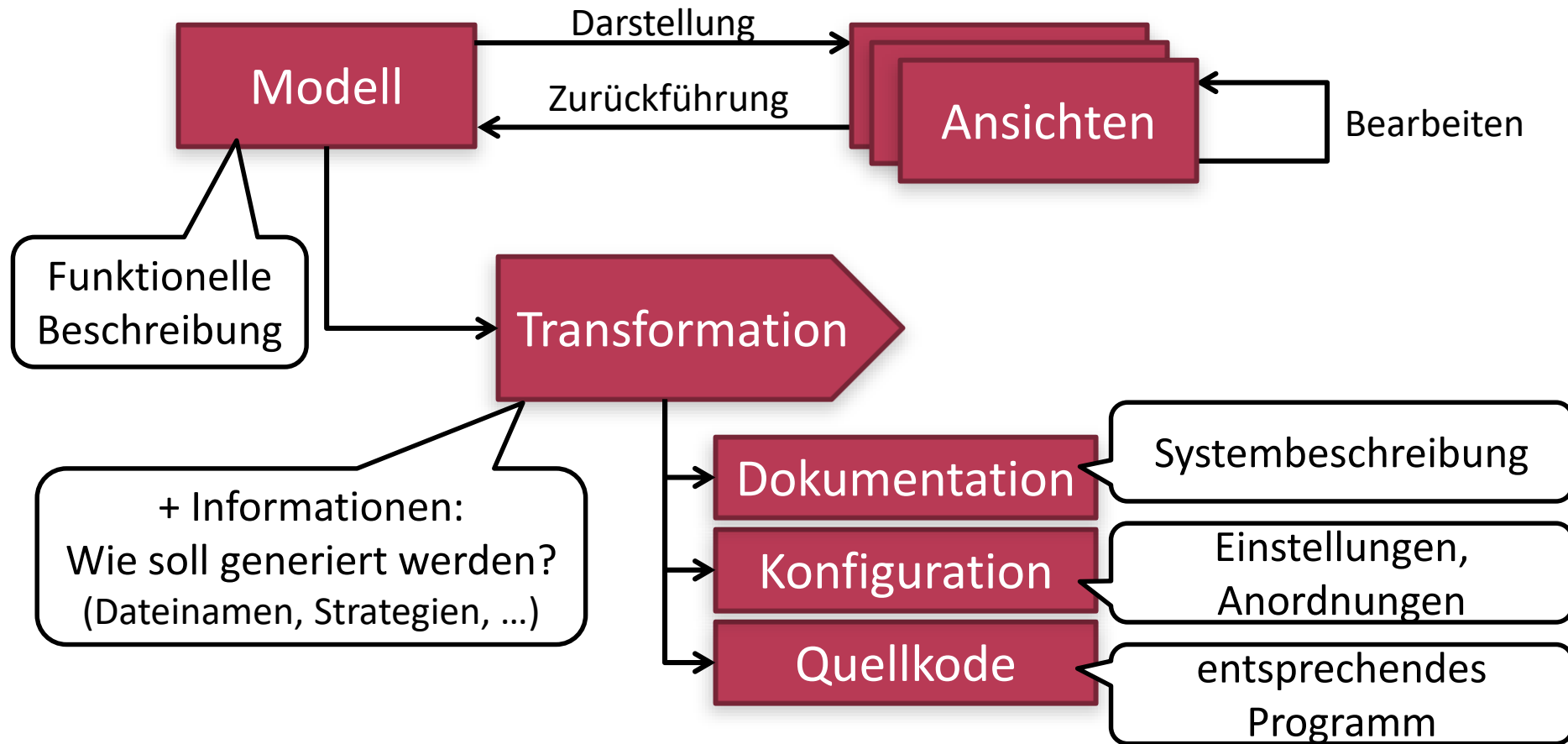
Graphisch



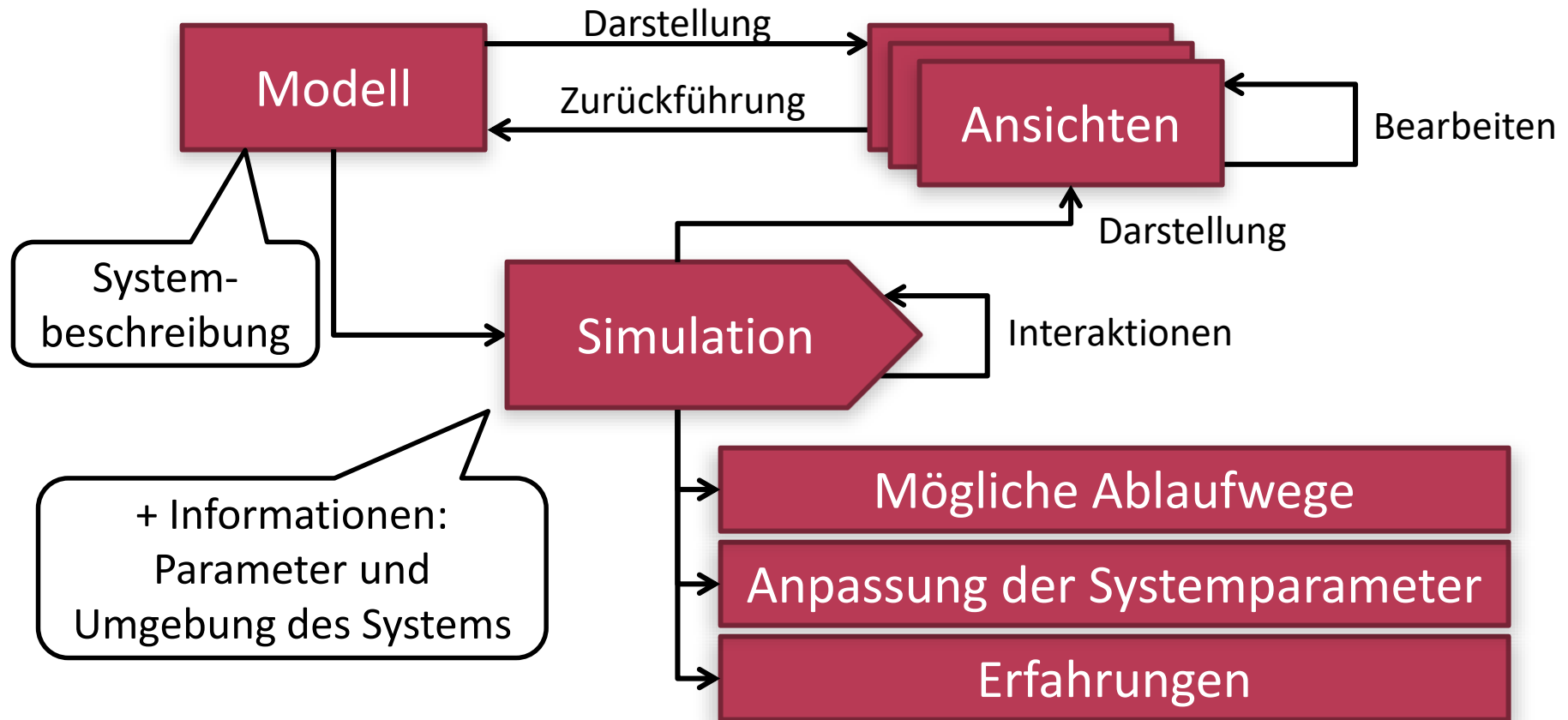
Strukturierte Schnittstellen



Funktionen einer Modellierungsumgebung



Funktionen einer Modellierungsumgebung



Funktionen

Umgebungen

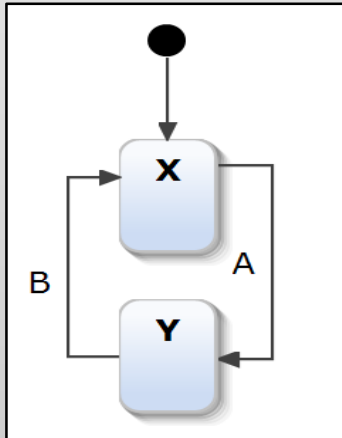
Kodegenerierung

MODELLIERUNGSUMGEBUNGEN

Modellierungsfunktionen von Yakindu

Konkrete Syntax
(für den Benutzer)

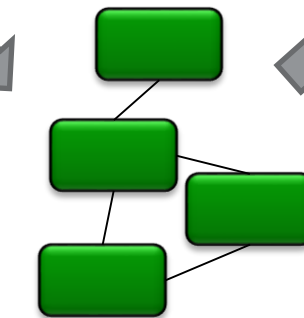
Graphisch:



Textuell:

Demo
internal:
event A
event B

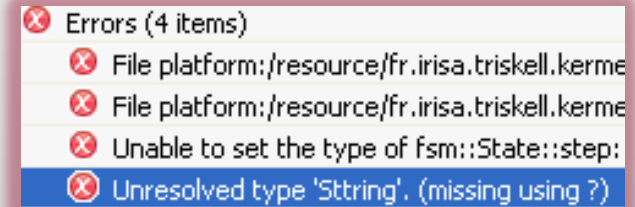
Syntax → Semantik



Modell
(Abstrakte Syntax)

Modellierungsfunktionen

Modellüberprüfung



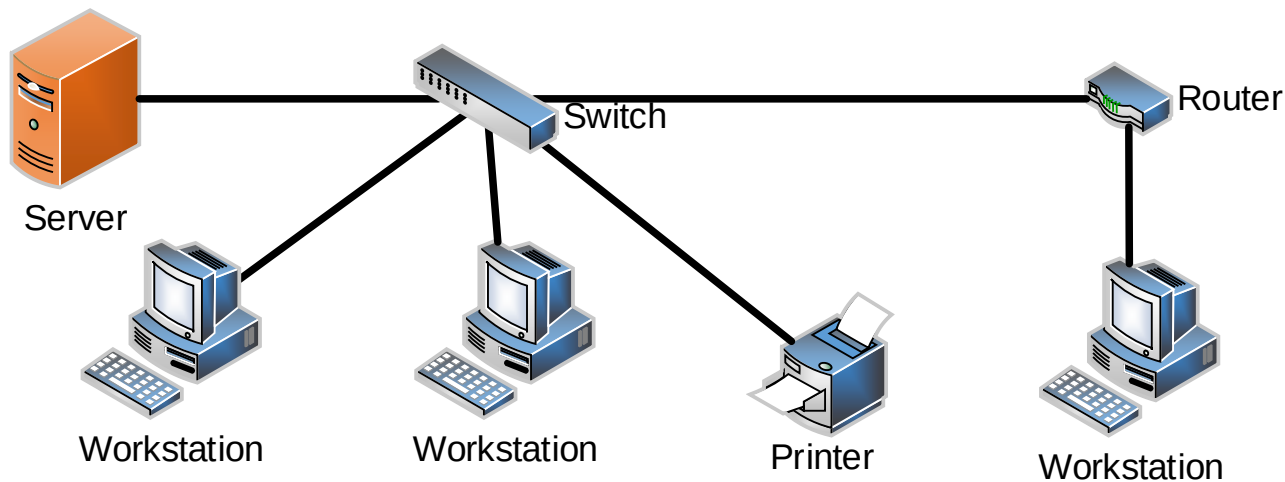
Kodegenerierung

```
</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" />
```

(Quellenkode, Dokumentation,
Konfiguration)

Abstrakte Syntax

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

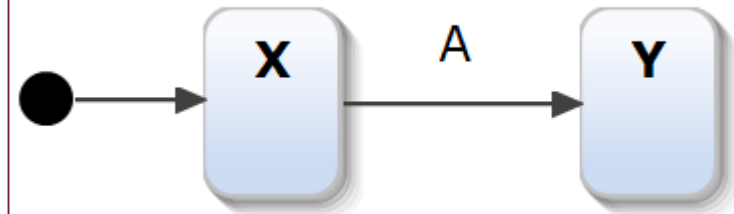


Beispiel – Abstrakte Syntax: Yakindu

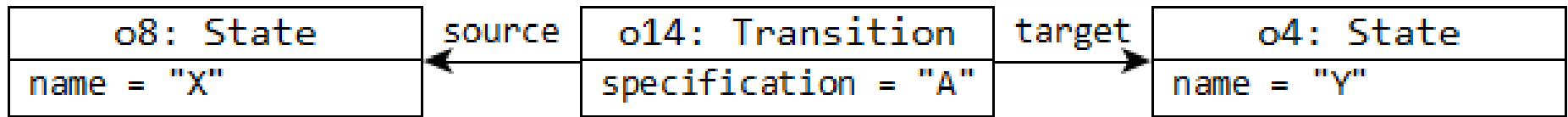
Frage:

Wie würden wir eine Modelleirungsumgebung implementieren?

Beispiel: Yakindu Modell



Abstrakte Syntax



Beispiel – Abstrakte Syntax: Yakindu

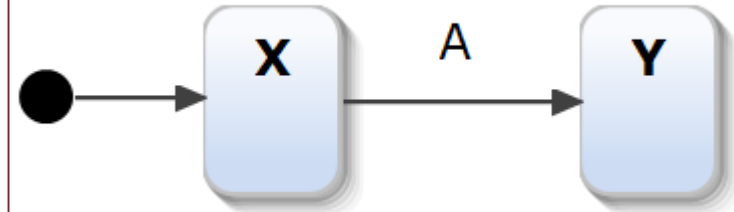
Frage:

Wie würden wir eine Modelleirungsumgebung implementieren?

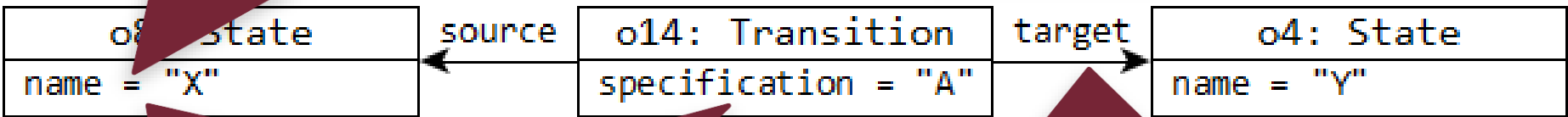
Namen werden als String gespeichert

name = "X"

Beispiel: Yakindu Modell



Abstrakte Syntax



Modellelemente als Objekte

Relationen als Referenzen

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

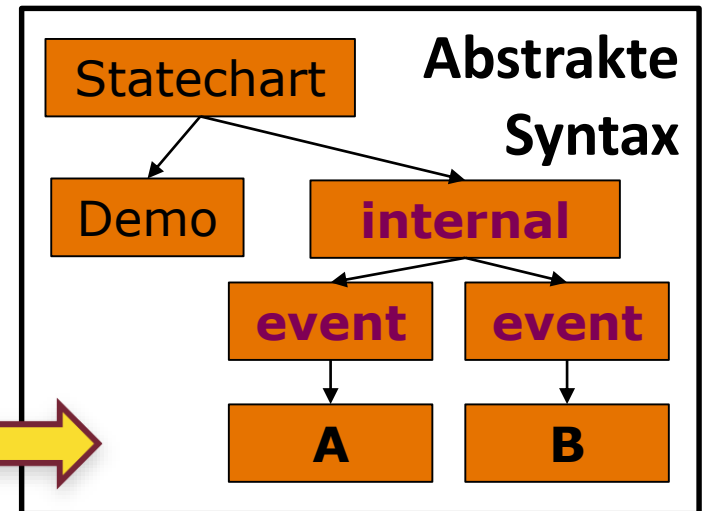
Konkrete Syntax: Textuelle Syntax

- **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

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

- 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 $\leftrightarrow$ 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 creating and editing diagrams. Below the toolbar is a diagram area containing two states: 'X' and 'Y'. State 'X' is a simple blue rectangle with the letter 'X' inside. State 'Y' is a blue rectangle with the letter 'Y' inside, surrounded by a dashed black border, indicating it is selected. Below the diagram area is a panel with two tabs: 'Properties' and 'Problems'. The 'Properties' tab is active, showing a table of properties for 'State Y'.

Property	Value
State Y	
Composite	☒ false
Documentation	
Incoming Transitions	→ X -> Y (A)
Leaf	☒ true
Name	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

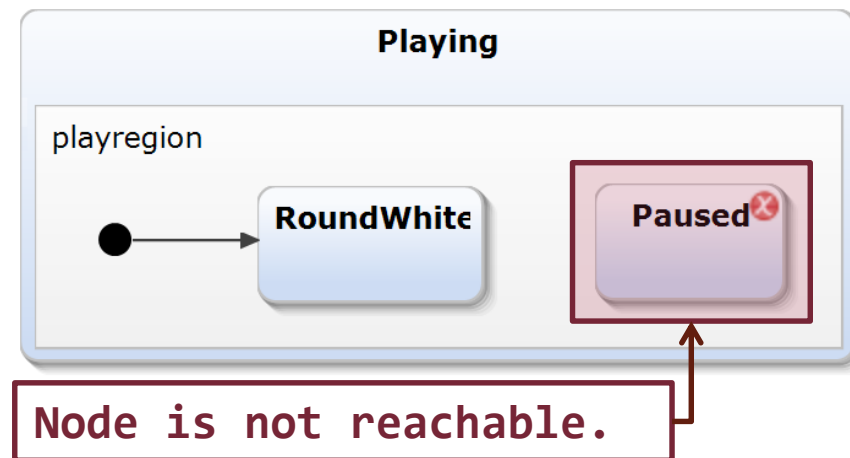
after 1 s [clock>0]/ clock=1



- 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

Umgebungen

Kodegenerierung

KODEGENERIERUNG

Motivation

- Abkürzung der Entwicklungszeit:
automatische Generierung der
 - Dokumentation
 - Quellencode
 - Konfigurationanhand der Modelle/Anforderungen/Pläne
- Beispiele: Systemmodellierung Hausaufgabe
 - Aufgabe: Parameter → Aufgabenbeschreibung
(in verschiedenen Sprachen), Testfälle
 - Yakindu: Modell → Kode

Beispiel: Generierung der Aufgabenbeschr.

- Beispiel: Anfangsbedenkzeit der Spieler
- Parameter:
 - Anfangswert der Anfangsbedenkzeit
 - Änderung der Anfangsbedenkzeit
 - nicht möglich
 - gleichzeitig für beide Spieler
 - getrennt für die zwei Spieler
- LaTeX Kode:

Am Anfang sind die die Bedenkzeit messenden Uhren beider Spieler auf die Anfangsbedenkzeit eingestellt. Der Wert der Anfangsbedenkzeit ist `\ifbool{setupInitialTimeSettable}{beim Einschalten der Schachuhr}\setupInitialTimeDefault{}` Sekunden für beide Spieler `\ifbool{setupInitialTimeSettable}{}`, er ist aber `\ifbool{setupInitialTimeIndividual}{für die zwei Spieler getrennt}` einstellbar (siehe Kapitel `\refstruc{sec:settings-details}`).

Beispiel: Generierung der Aufgabenbeschr.

■ LaTeX Kode:

Am Anfang sind die die Bedenkzeit messenden Uhren beider Spieler auf die Anfangsbedenkzeit eingestellt. Der Wert der Anfangsbedenkzeit ist `\ifbool{setupInitialTimeSettable}{beim Einschalten der Schachuhr}{\setupInitialTimeDefault}` Sekunden für beide Spieler `\ifbool{setupInitialTimeSettable}{, er ist aber \ifbool{setupInitialTimeIndividual}{für die zwei Spieler getrennt}{einstellbar (siehe Kapitel \refstruc{sec:settings-details})}`.

■ Die generierte Aufgabenbeschreibung

Am Anfang sind die die Bedenkzeit messenden Uhren beider Spieler auf die Anfangsbedenkzeit eingestellt. Der Wert der Anfangsbedenkzeit ist beim Einschalten der Schachuhr 150 Sekunden für beide Spieler, er ist aber einstellbar (siehe Kapitel 2.4).

Am Anfang sind die die Bedenkzeit messenden Uhren beider Spieler auf die Anfangsbedenkzeit eingestellt. Der Wert der Anfangsbedenkzeit ist beim Einschalten der Schachuhr 180 Sekunden für beide Spieler, er ist aber für die zwei Spieler getrennt einstellbar (siehe Kapitel 2.4).

Programmierer vs. Kodegenerator

- Vorteile des **richtig geschriebenen** Kodegenerators
 - Generiert richtigen Kode → Zertifikation
 - Kode wird auf Knopfdruck generiert
 - Nach jeder Änderung gleich
 - Optimierbar auf Lesbarkeit, Effizienz, etc.
- Kodegenerator richtig zu schreiben ist schwierig
 - muss viel können, siehe oben

Aufgaben der Kodegenerierung

- **Aufgabe:** automatische Generierung von sich entsprechend verhaltenen Modellen
- Mehrere Möglichkeiten → Entwurfsentscheidung
 - **Interpretiert:** Modell wird eingelesen und ausgeführt
Programmcode: Synthese der Quellencode
 - **Programmiersprachen:** Java, C, C++, ...
 - **Optimierung:** Speicher vs. Prozessor
Beobachtbarkeit vs. Performanz
 - Verbindung des generierten Codes mit eigenem Code
- Kodegenerator: parametrisierbar + ergänzbar

Interpretieren vs. Generieren

- **Dynamischer Interpreter**
 - Schnelles Starten, Ergebnisse gleich
 - Im Allgemeinen Zusatzkosten in Zeit/Speicher
 - Abhängigkeiten zur Ausführungszeit
 - Kann Änderung des Modelles während der Ausführung unterstützen
 - Verhalten entspricht immer dem Modell
- **Kodegenerator**
 - Starten erst nach Generieren und Kompilieren
 - Optimierbar auf effizientes Ausführen
 - Nach Änderung muss neu generiert werden
 - Kode manuell veränderbar
→ *Wollen wir das?*

Kodgenerator

- Was ist ein Kodegenerator?

- Irrglaube:

Programm, welches Programme schreibt ...???

- Realität:

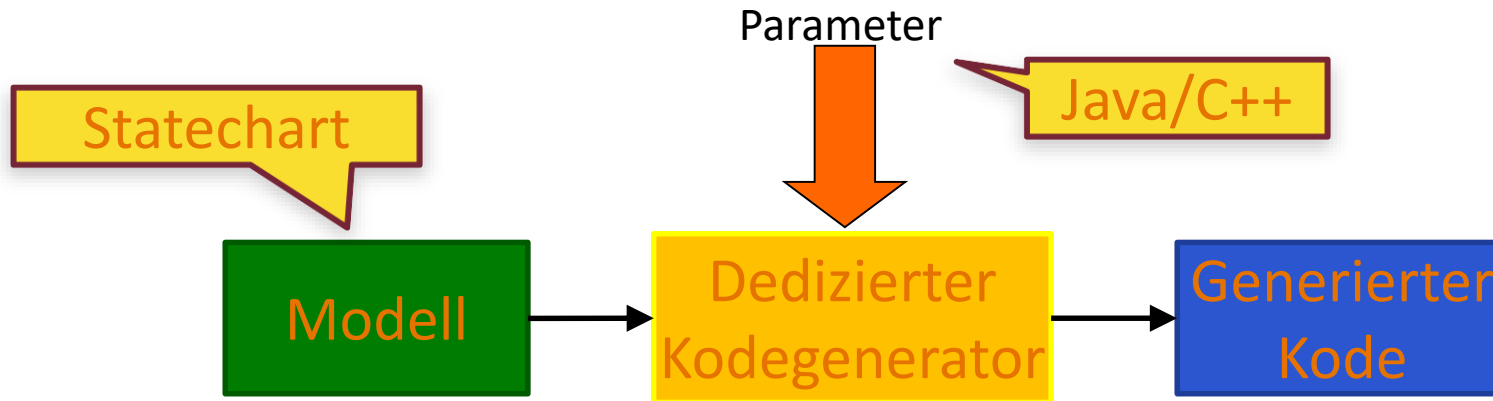
Einfaches Programm mit viel, viel Ausschreiben

```
sourceFile.write("    temp = ({AIDA_PARTITION_TYPE*} selfModule.partitions.elements);\n")
i = 0
for partition in partitions:
    numPorts = getNumberOfAllCommPorts_Partition(currModuleComm, interPartitionComm, partition.partitionName)
    sourceFile.write("    temp[" + str(i) + "].partition_id = " + str(partition.partitionID) + ";\n")
    sourceFile.write("    strcpy( &temp[" + str(i) + "].partition_name[0], \"" + str(partition.partitionName) + "\");\n")
    sourceFile.write("    temp[" + str(i) + "].ports.type = CONST_AIDA_PORTS_TYPE;\n")
    sourceFile.write("    temp[" + str(i) + "].ports.elements = &mem_ports_" + str(partition.partitionName) + "[0];\n")
    sourceFile.write("    temp[" + str(i) + "].ports.numOfElements = " + str(numPorts) + ";\n")
    sourceFile.write("\n")
    i = i + 1
## end for
sourceFile.write("\n")
```

- Kitekintés: *Quine* – Programm, welches seinen eigenen Kode ausschreibt

Kodegeneratortypen I.

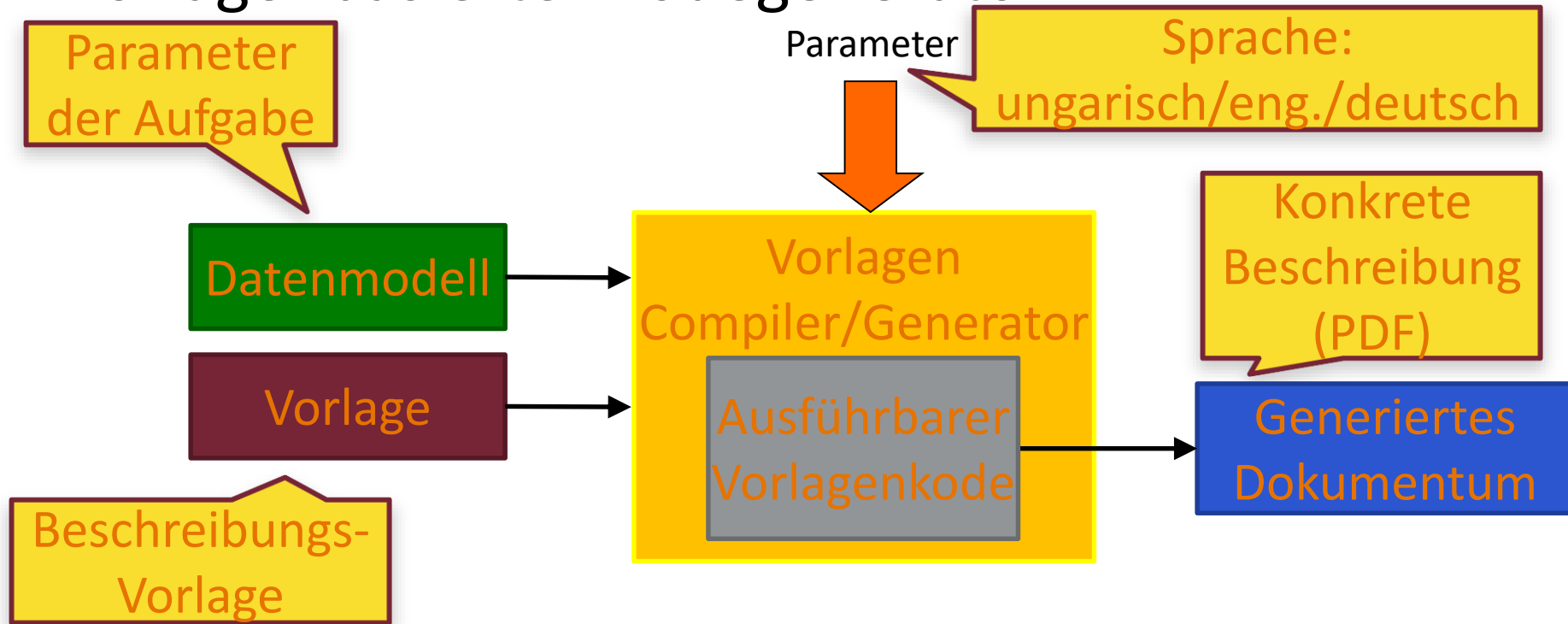
- Dedizierter Kodegenerator (z.B. Yakindu)



- Teil einer Modellierungsumgebung
- Zertifizierbar
- Kann nicht richtig angepasst werden

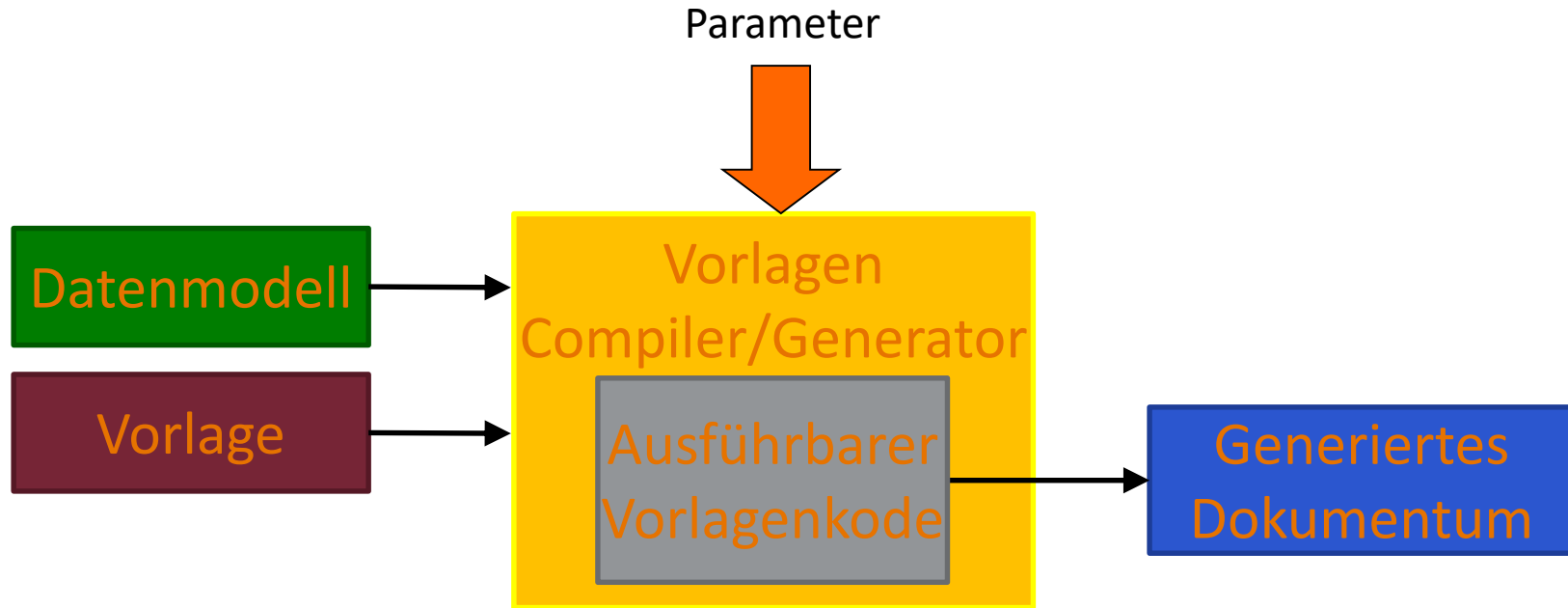
Kodegeneratortypen II.

■ Vorlagenbasierter Kodegenerator



Kodegeneratortypen II.

■ Vorlagenbasierter Kodegenerator



- Schnellere Entwicklung
- Komplexe Änderungen während des Lebenszyklus
 - Vorlage und Modell können unabhängig geändert werden

Funktionen

- Effizienz vs. Wiederverwendbarkeit, Lesbarkeit
 - Nachverfolgbarkeit
 - Welches Modellelement ist zu einem gegebenen Kodeteil geführt?
 - Unterstützung der *Inkrementalität*: der Code wird nur da geändert, wo das Modell geändert wird
- Effizientes Neugenerieren

Anpassung des generierten Codes

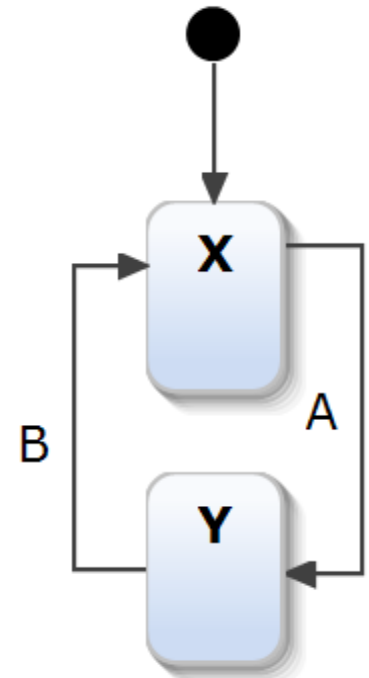
- Der vom Modell generierte Code ist nur ein Teil der Anwendung. *Wird er zu den anderen beigelegt. Wie?*
 - Generierter Code wird manuell nicht geändert
→ kann jederzeit neu generiert werden!
 - Im Modell/Vorlage würde es die Komplexität erhöhen
Passt einfach nicht da hinein ...
- Generierter und manuell geschriebener Code kommt in getrennten Dateien/Verzeichnisse
- Anpassung: *Klebekode* (glue code)
 - Aufruf des generierten Codes
 - Ableitung aus dem generierten Code

KODEGENERATOR – EIN BEISPIEL

Die Aufgabe des Kodgenerators

- **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

Demo
internal:
event A
event B



Vorlagenbasierter Kodegenerator (Xtend)

- Ziel: Zustände → Enum

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

- Lösung: C Programm

```
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 (*escape*)

Die Vorlage wird geschrieben

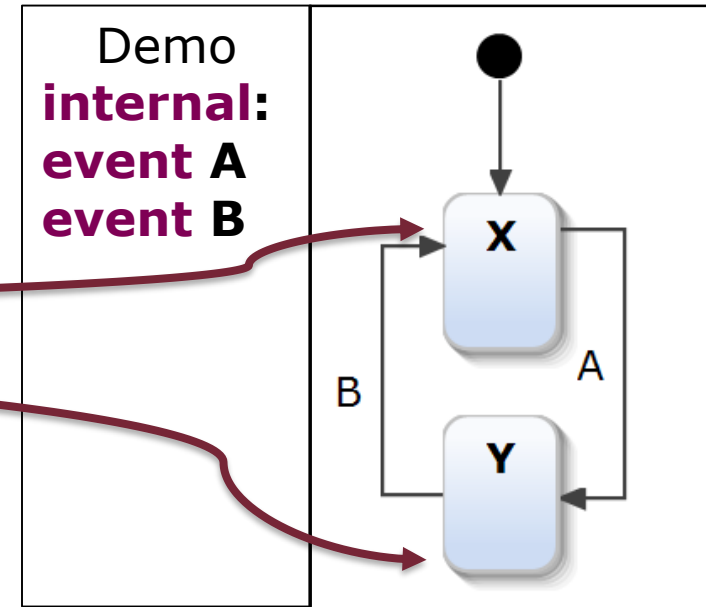
😊 Einfacher zu schreiben
😊 Übersichtlich
😊 Leicht zu modifizieren
☹ +1 Technologie

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» z.B.: StateX

Kodegenerator Beispiel – Anfangszustand

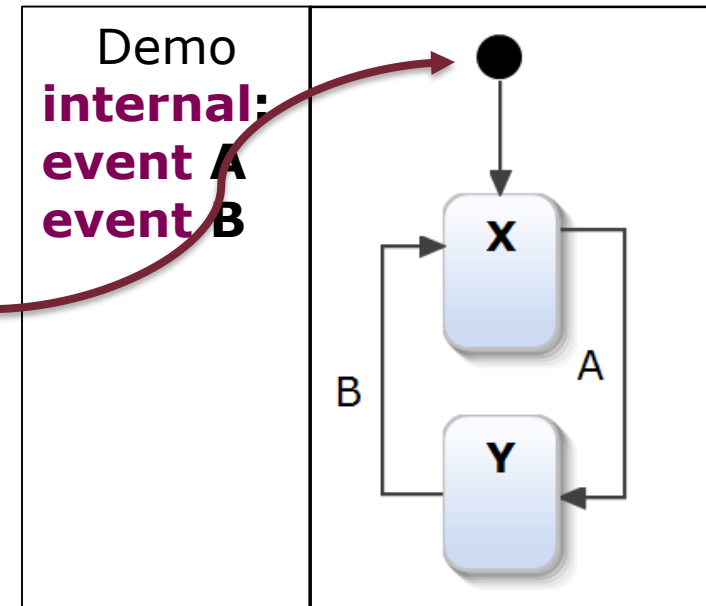
■ Erwarteter C-Code:

```
// 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 geschrieben

Kodegenerator Beispiel – Zustandsübergänge

■ Erwarteter C-Code:

```
// Execute "A" event  
void doA{  
  switch(actualState) {
```

```
    case StateX:
```

```
        actualState = StateY;
```

```
        break;
```

```
    case StateY:
```

```
        break;
```

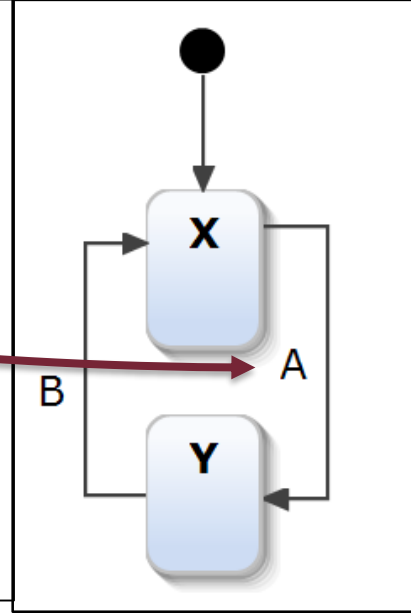
```
  }
```

```
}
```

A / X → Y

A / -

Demo
internal:
event A
event B



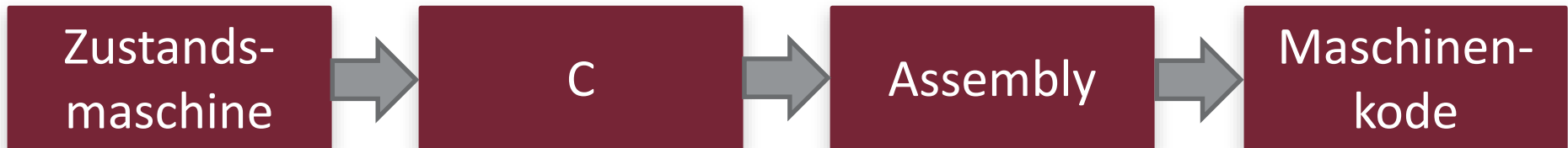
■ Vorlage (skizziert):

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 Kode

Kodegenerator – Zusammenfassung

- Kodegenerierung = Übersetzer
- Gleiche Schritte:

Prophezeiung:

Zustand
maschin

gestern Entwurfsmuster →

heute Funktion der Kodegeneratoren →

morgen Element der Modellierungssprachen

schinen-
kode

- 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 Kode