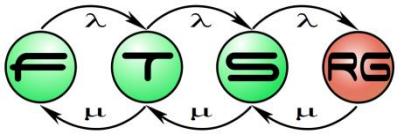
Modellierungsumgebungen, Kodegenerierung

Budapest University of Technology and Economics
Fault Tolerant Systems Research Group

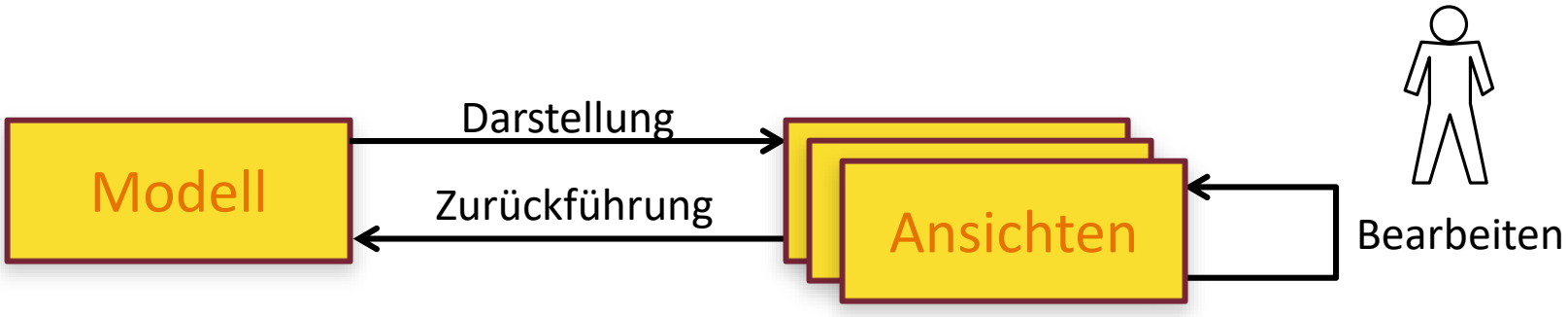


Inhalt

- **Funktionen**
- **Modellierungsumgebungen**
- **Kodegenerierung**

DIE FUNKTIONEN EINER MODELLIERUNGSUMGEBUNG

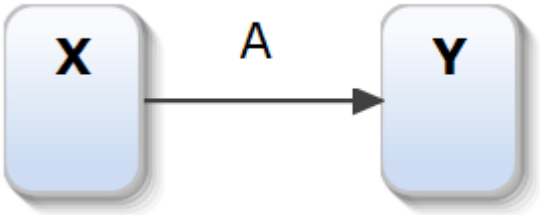
Funktionen einer Modellierungsumgebung



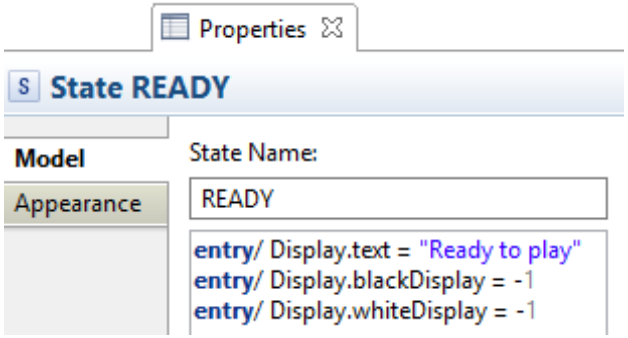
Textuell

Demo
internal: event A
event B

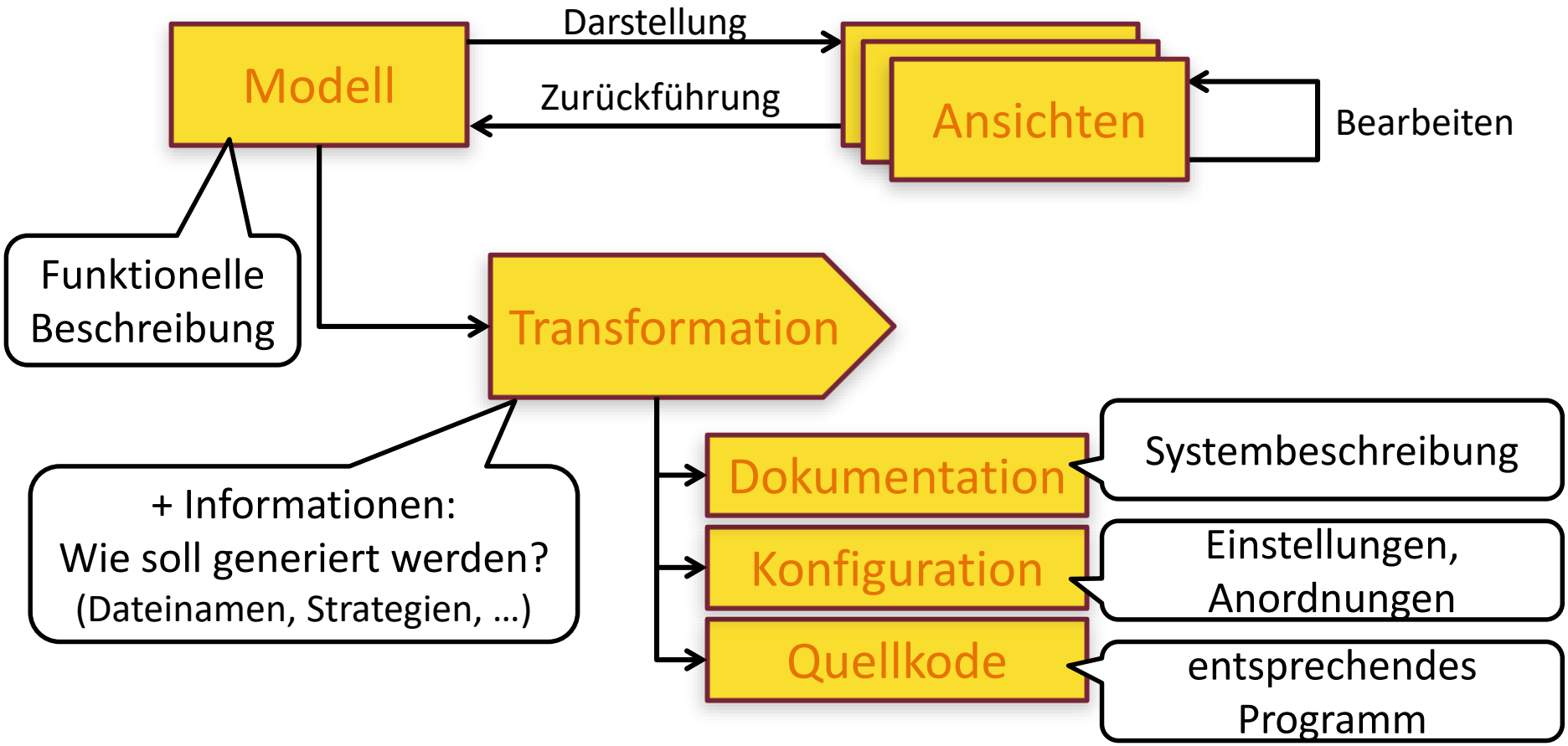
Graphisch



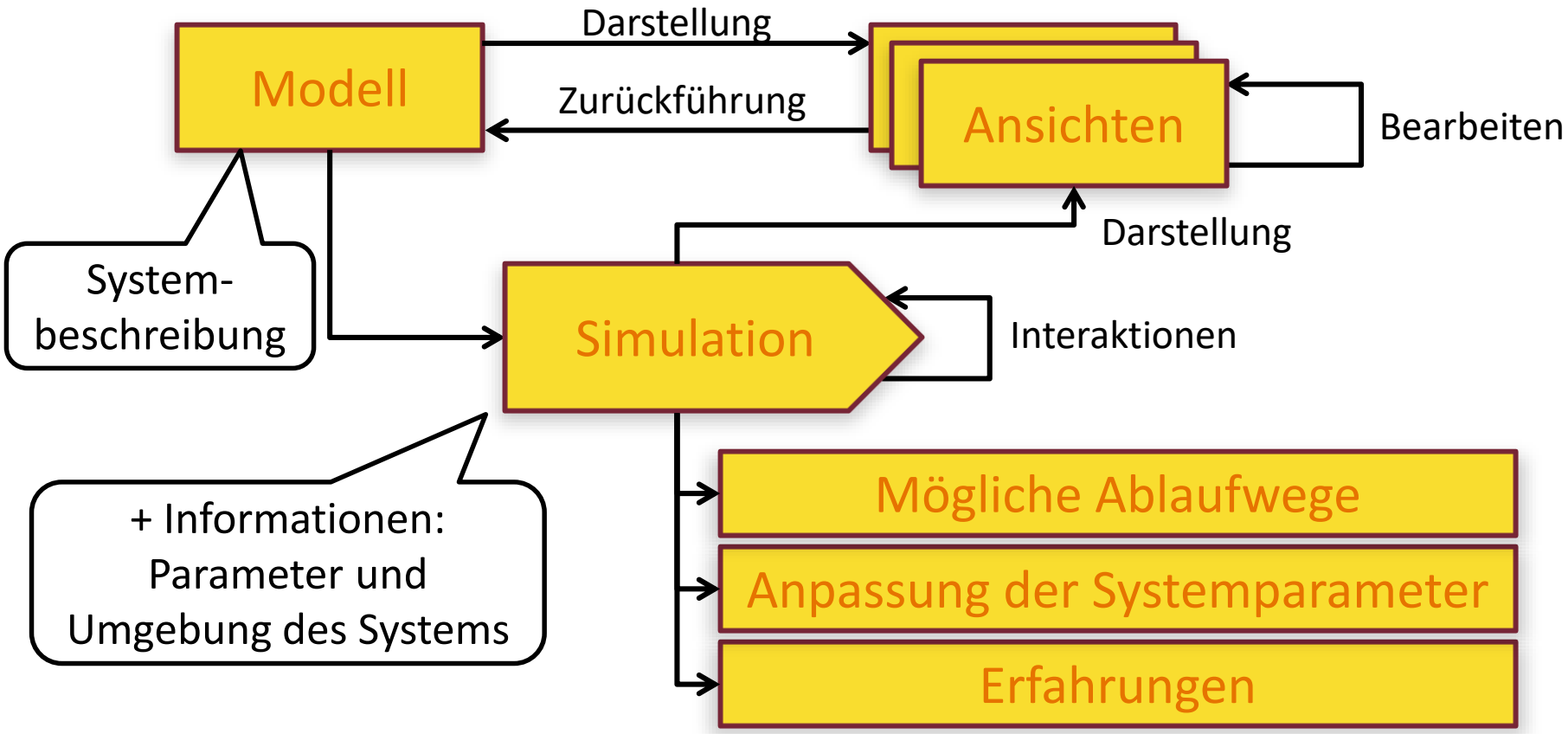
Strukturierte Schnittstellen



Funktionen einer Modellierungsumgebung



Funktionen einer Modellierungsumgebung

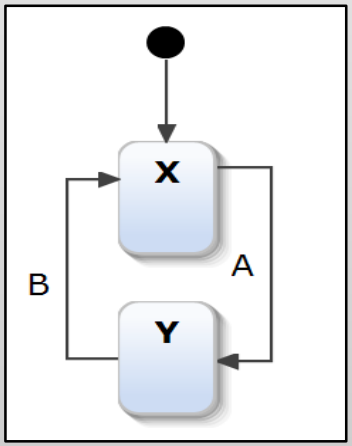


MODELLIERUNGSUMGEBUNGEN

Modellierungsfunktionen von Yakindu

Konkrete Syntax
(für den Benutzer)

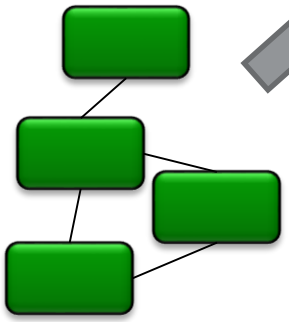
Graphisch:



Textuell:

Demo
internal:
event A
event B

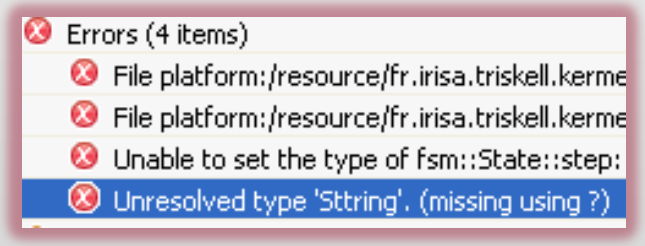
Syntax → Semantik



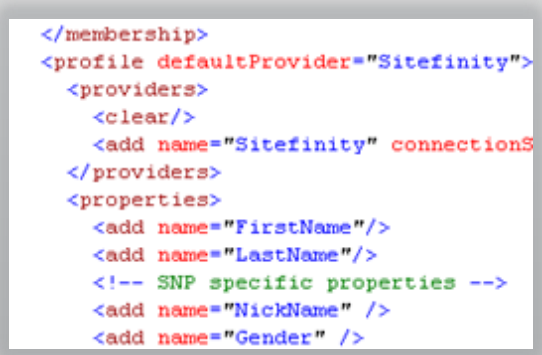
Modell
(Abstrakte Syntax)

Modellierungsfunktionen

Modellüberprüfung



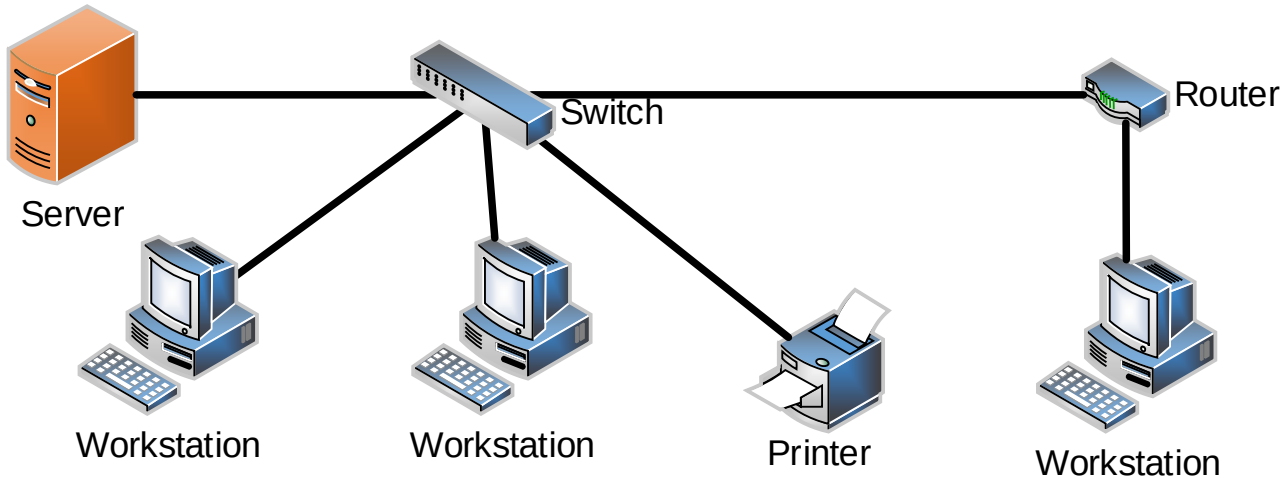
Kodegenerierung



(Quellenkode, Dokumentation,
Konfiguration)

Abstrakte Syntax

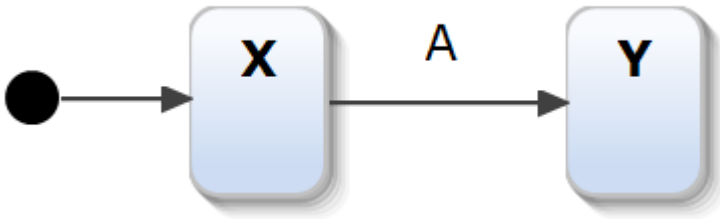
- **Definition:** Strukturelles Modell des zu editierenden Systemmodells
  - Strukturelles Modell eines Modells ???
- Wird von der Modellierungsumgebung verwaltet
- Zur Erinnerung: Strukturelles Modell = **Graph**
  - **Graph von: Knoten, Kanten, Eigenschaften**



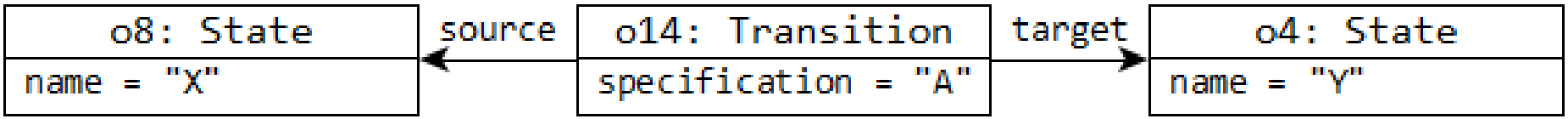
Beispiel – Abstrakte Syntax: Yakindu

Frage:
Wie würden wir eine Modelleirungsumgebung implementieren?

Beispiel: Yakindu Modell



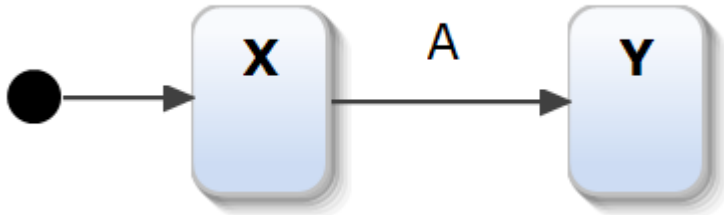
Abstrakte Syntax



Beispiel – Abstrakte Syntax: Yakindu

Frage:
Wie würden wir eine Modelleirungsumgebung implementieren?

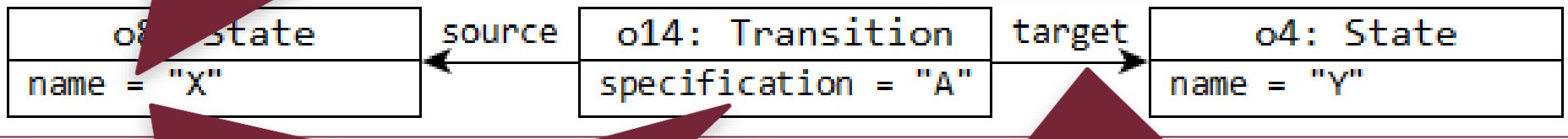
Beispiel: Yakindu Modell



Namen werden als String gespeichert

```
name = "X"
```

Abstrakte Syntax



Modellelemente als Objekte

Relationen als Referenzen

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

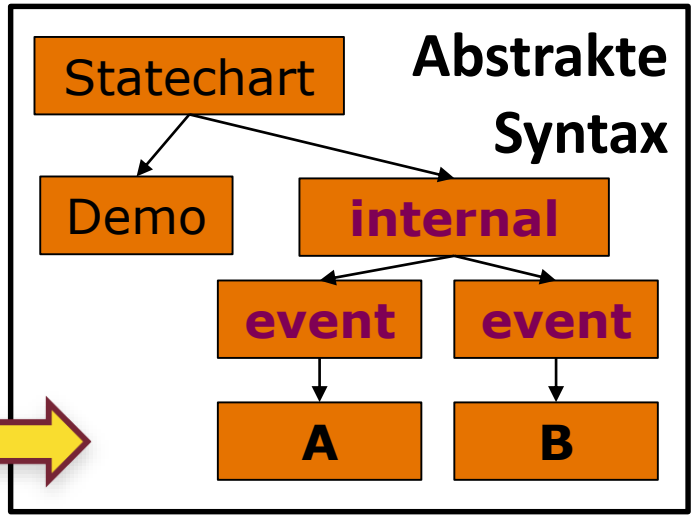
Konkrete Syntax: Textuelle Syntax

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

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

Grammatik

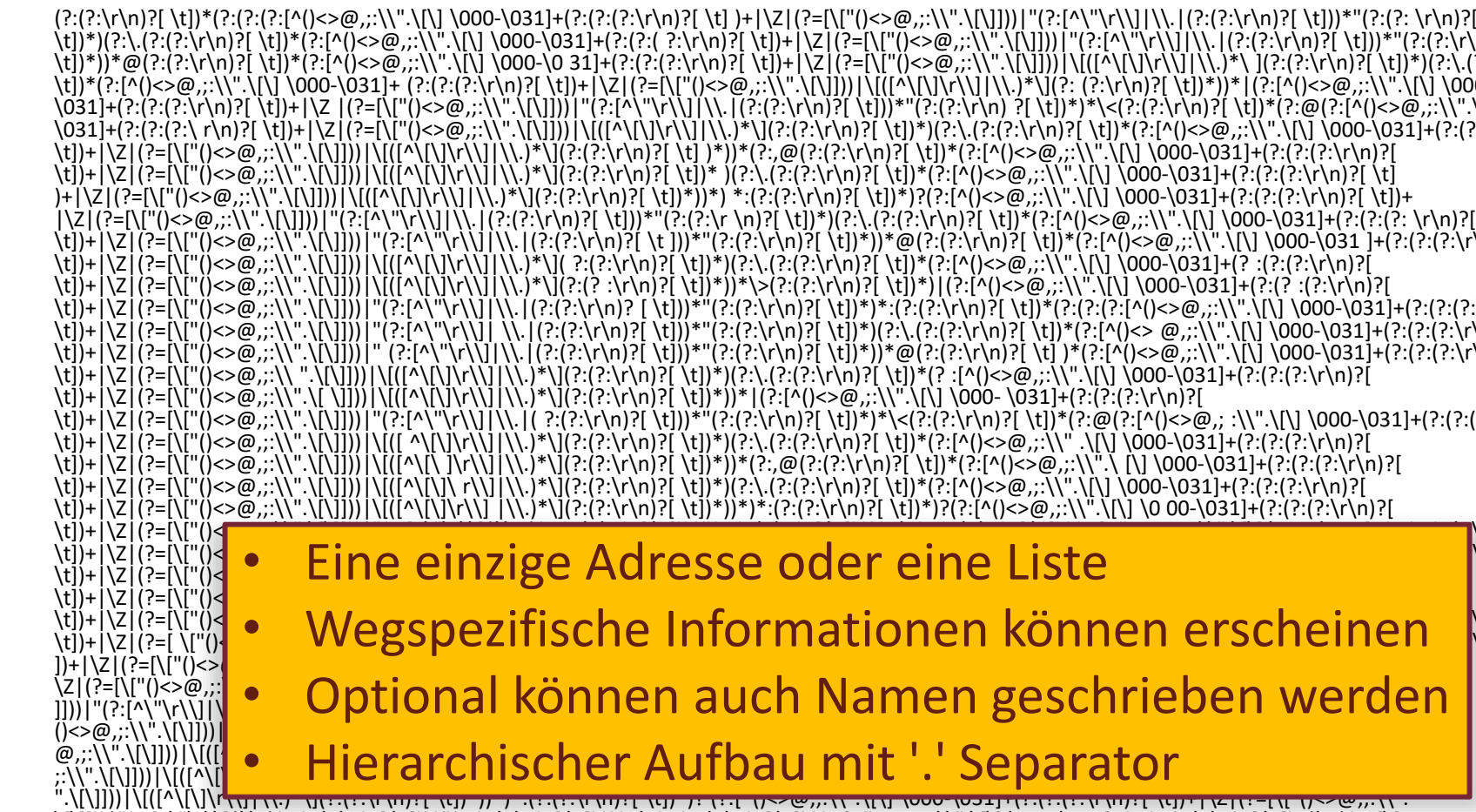
```
<Statechart>::= <Name> <Interface>*
<Interface> ::= ("internal" | <Name>)
                ":" <Event>*
<Event>     ::= "event" <Name>
<Name>      ::= …
```



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

Gegenbeispiel für „geeigneten Methoden”

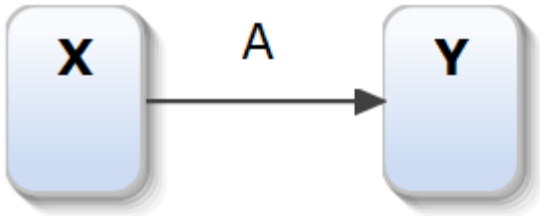
- **Aufgabe:** entscheiden ob ein Text eine Emailadresse ist



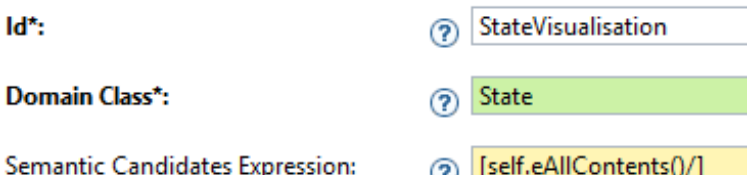
- Eine einzige Adresse oder eine Liste
- Wegspezifische Informationen können erscheinen
- Optional können auch Namen geschrieben werden
- Hierarchischer Aufbau mit '.' Separator

Konkrete Syntax: Graphische Syntax

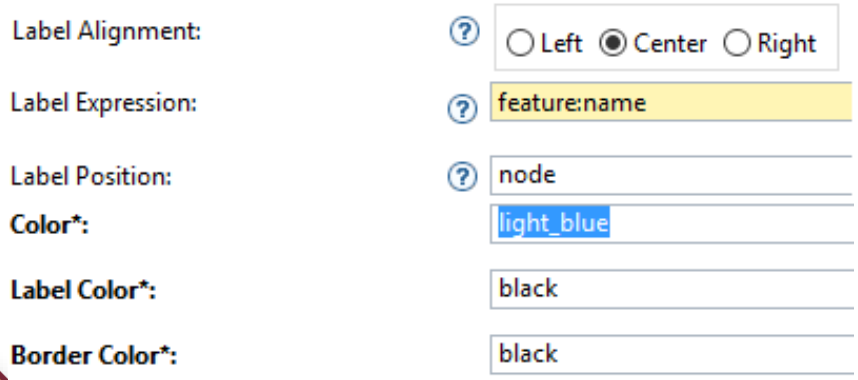
- **Ziel:** Repräsentation⇔ das Modell
- Graphische Syntax (z.B. Diagramm)
  - Aufgabe: Diagramm ↔ Modell
  - Übersichtlicher, schwieriger zu schreiben, regelbasiert



Kondition auf dem Modell



Anlegen des Diagrammelementes

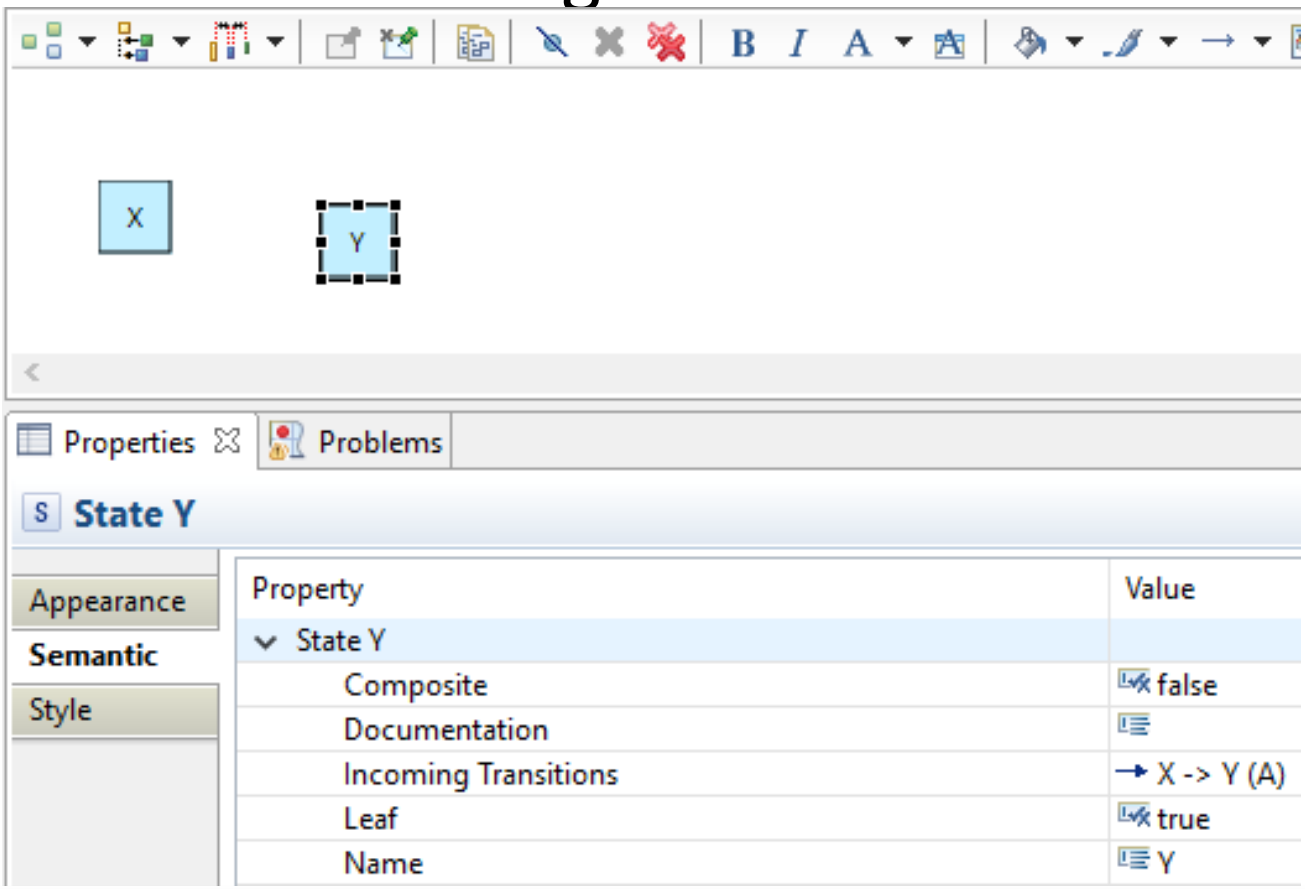


Kondition erfüllt → Diagrammelement wird angelegt

Diagramm wird geändert → Modell ändert sich auch

Konkrete Syntax: Graphische Syntax

Ergebnis:



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

Modellvalidierung: Syntaktische Überprüfung

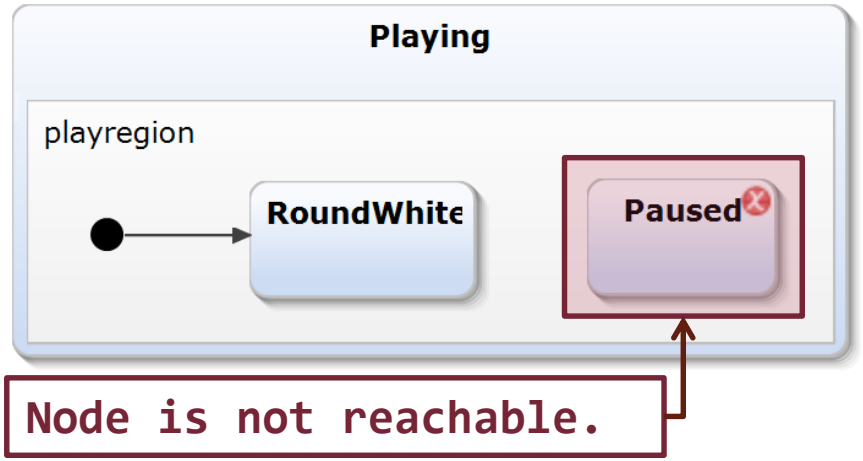
- Syntaktische Überprüfung: die Modellierungsumgebungen verbinden die logisch zusammengehörenden Elemente

- Syntaxgesteuerte Editoren
  - Fehler während Bearbeitung→ `Couldn't resolve reference`
  - Moderne Umgebungen: Vorschlagen der Kandidaten
- Kode+Diagram gemeinsam

- 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 Wertzuornungen, etc.

KODEGENERIERUNG

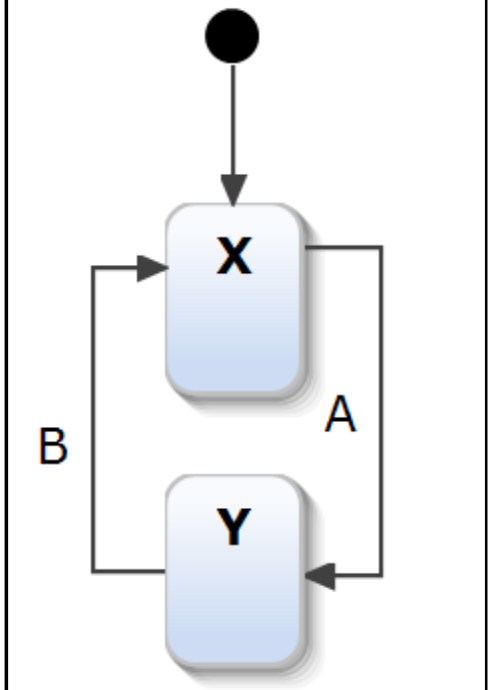
Aufgaben der Kodegenerierung

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

Kodgeneratoren – ein Beispiel

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

Demo
internal:
event A
event B



- 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 Kodegenerator (Xtend)

- Ziel: Zustände → Enum

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

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);
```

☺ Funktioniert gleich!
☹ Unlesbar

- Vorlage (Xtend):

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

Die variablen Stellen werden angegeben (*escape*)

Die Vorlage wird geschrieben

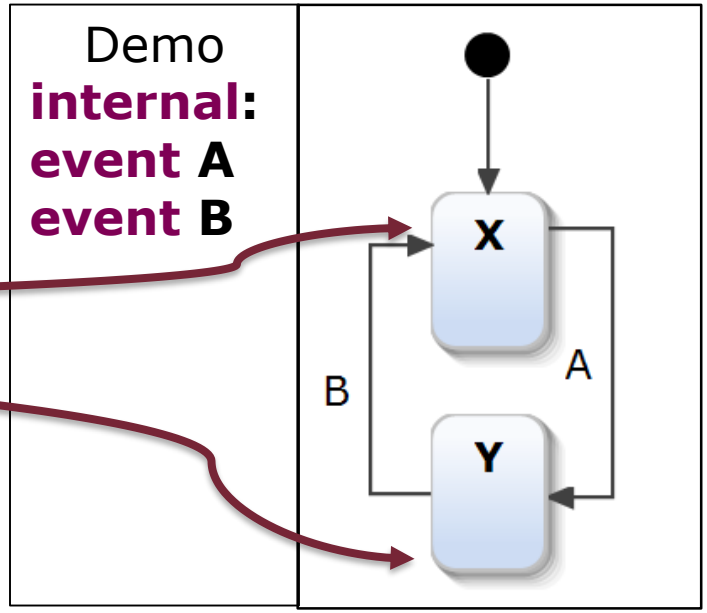
☺ Einfacher zu schreiben
☺ Übersichtlich
☺ Leicht zu modifizieren
☹ +1 Technologie

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»
};
```

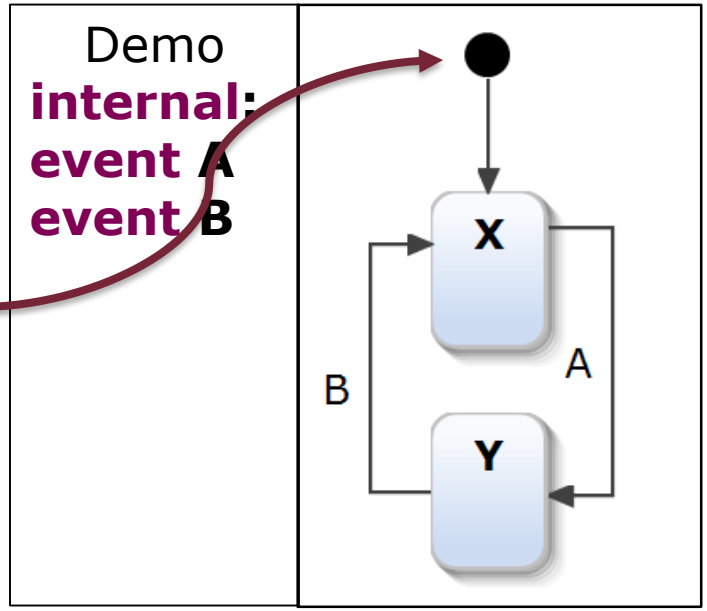
1. Wir iterieren durch alle Zustände
2. Ihre Namen werden mit Komma getrennt ausgeschrieben:
   `State«Name»` z.B.: `StateX`

Kodegenerator Beispiel – Anfangszustand

- Erwarteter C-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

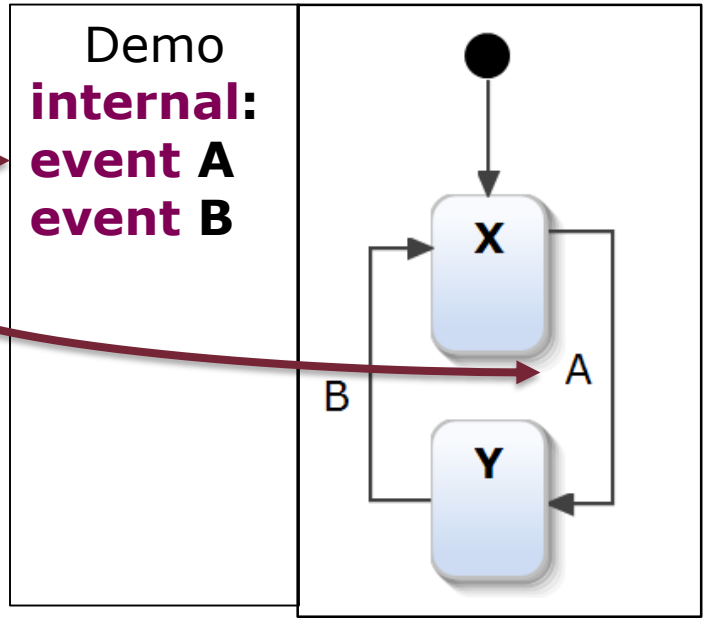
Kodegenerator Beispiel – Zustandsübergänge

- Erwarteter C-Kode:

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

A / X → Y

A / -



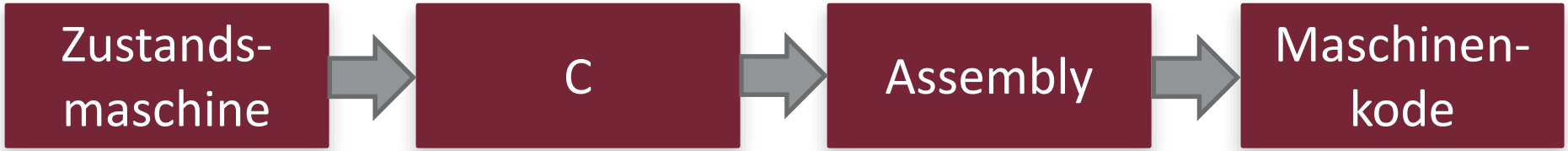
- Vorlage (skizziert):

1. Für jedes Ereignis eine Funktion `do«NameDesEreignisses»`
2. Der Funktioninhalt 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:

Zustands- schinen-
maschin- kode

Prophezeiung:
gestern Entwurfsmuster →
heute Funktion der Kodegeneratoren →
morgen Element der Modellierungssprachen

- 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