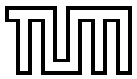
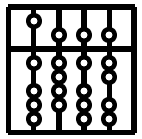


Technische Universität München,

Institut für Informatik



Hauptseminar

Werkzeuggestützte Modellierung des Tamagotchi mit AutoFocus

Bearbeitung Alexander Seitz,
Thomas Seliger und
Florian Kunkel

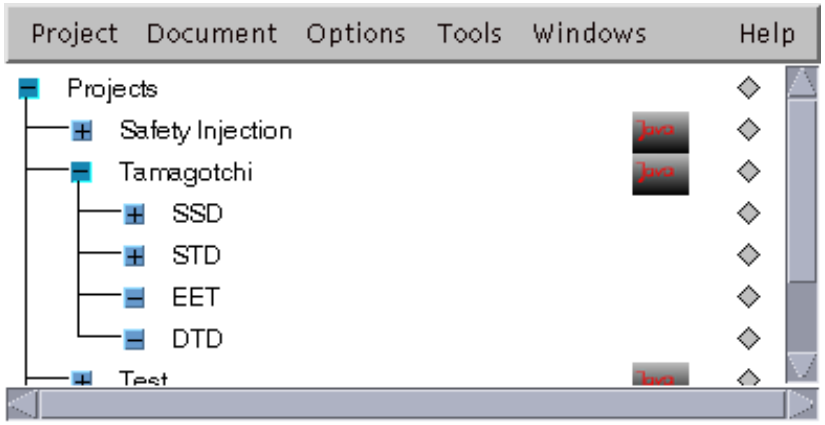
Betreuung Dr. Bernhard Schätz

Aufgabenstellung Prof. Dr. Manfred Broy

Vortragstermin 02.02.1999

Überblick über AutoFocus

- entwickelt am Lehrstuhl Broy (Softwaretechnikpraktikum 1996, Mitarbeiter, studentische Hilfskräfte)
- baut auf der formalen Beschreibungstechnik **Focus** auf
- Werkzeug zur Erstellung verteilter, eingebetteter Systeme unter graphischer Oberfläche mit Simulationsumgebung
- „Modellierungszentrale“:



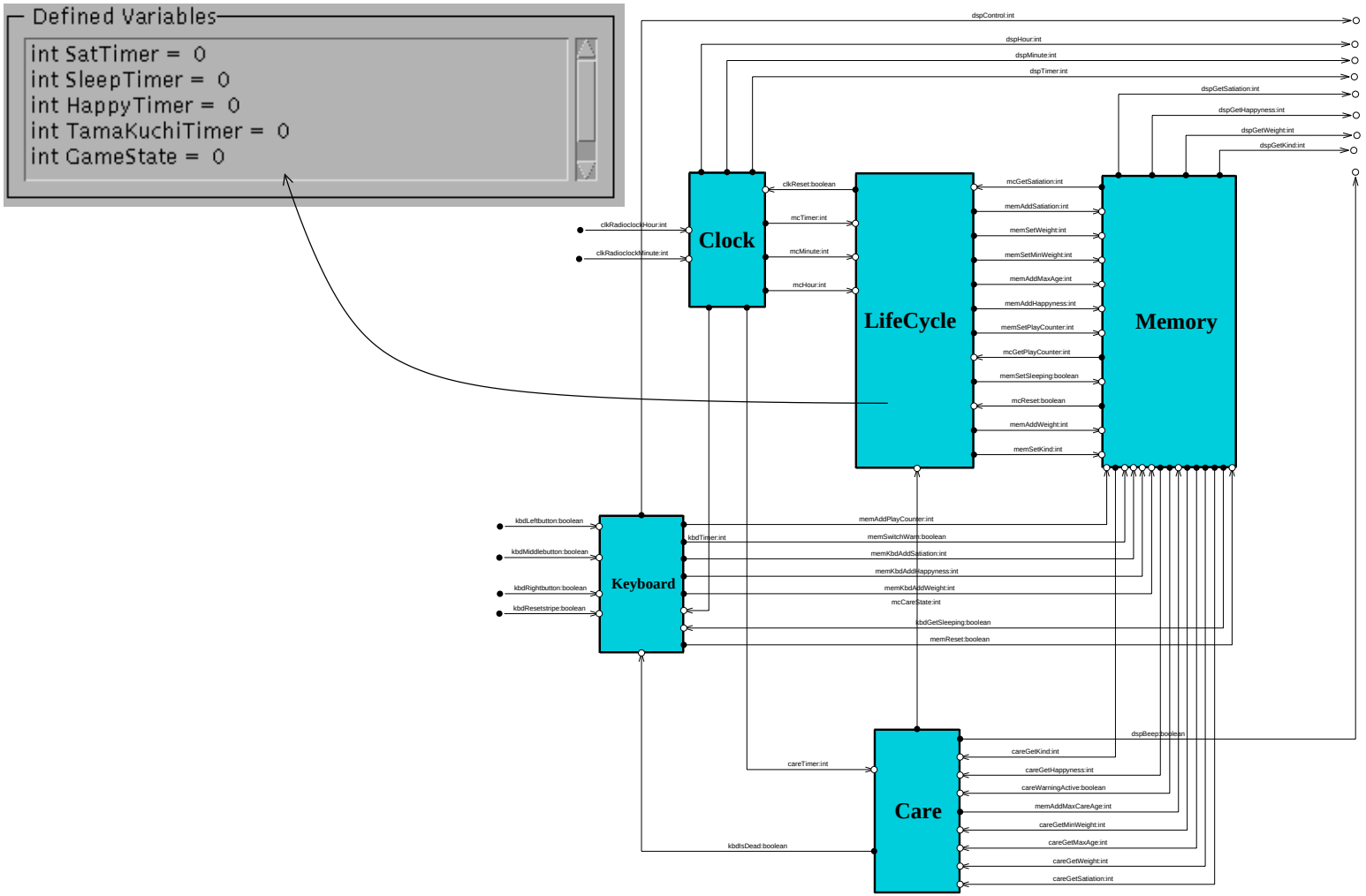
Konzept der Notation

- **Sichten:**

SSD	System Structure Diagram — Systemstrukturdiagr. *Statische Sicht mit Komponenten und Kanälen*
STD	State Transition Diagram — Zustandsübergangsdiagr. *Dynamisches Verhalten der Komponenten*
DTD	Data Type Definition — Datentypdef. *z.B. Art der Kanäle zwischen Komponenten*
EET	Extended Event Trace — Erweitertes Ereignisdiagr. *zum Verständnis des Verhaltens eines SSDs*

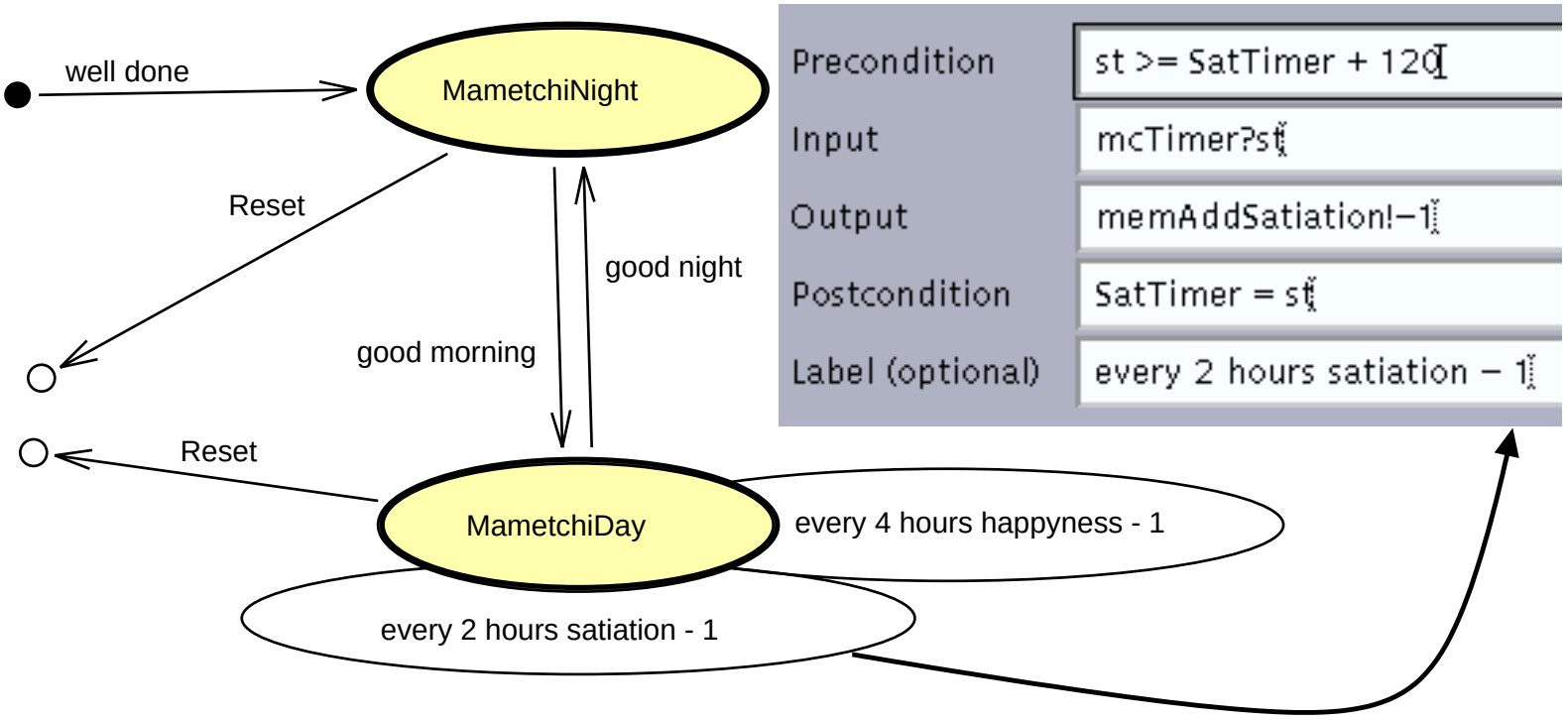
- **Hierarchische Entwicklung:**
  SSDs und STDs können Unterstrukturen gleicher Art haben.

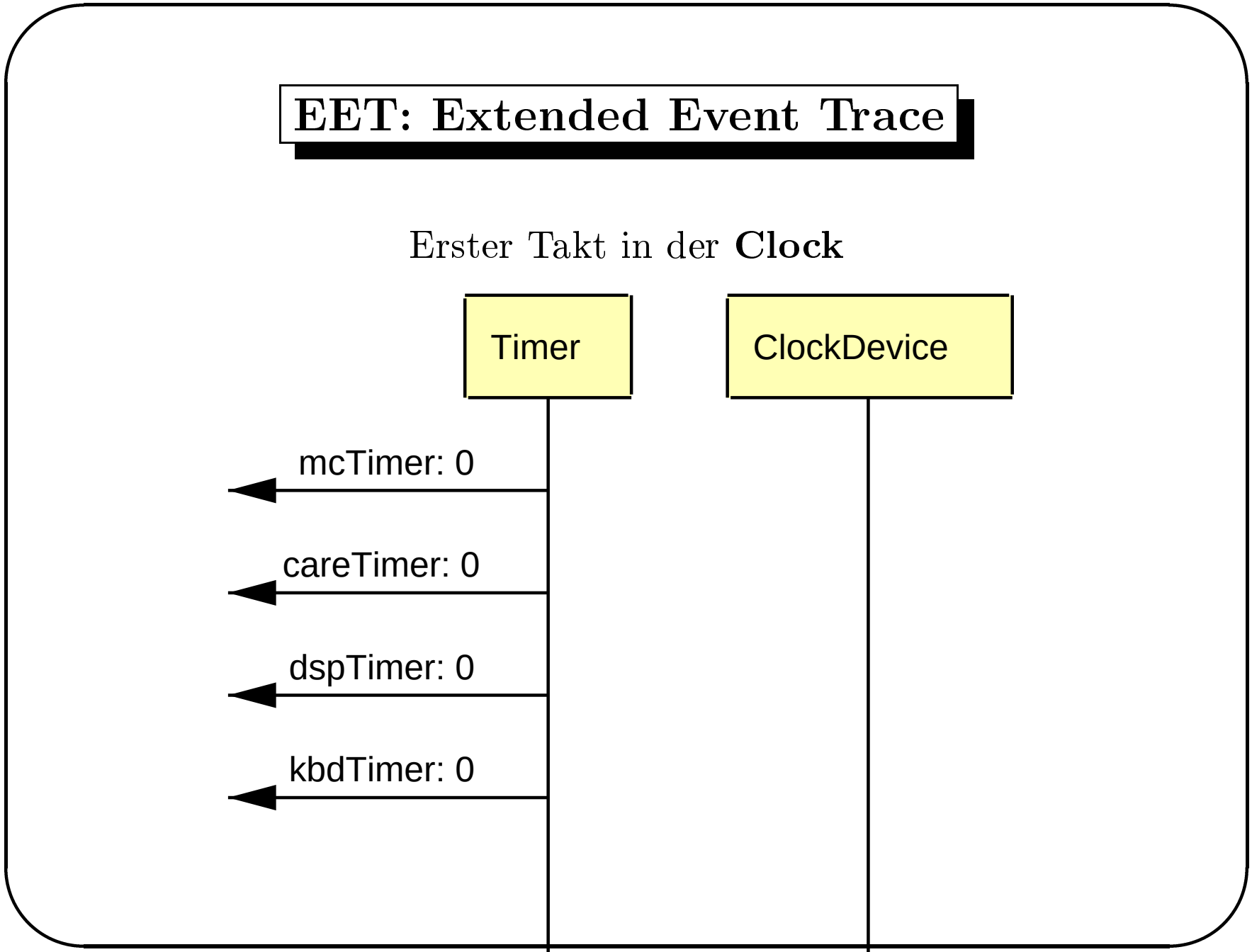
SSD: System Structure Diagram



STD: State Transition Diagram

wachendes und schlafendes Mametchi in LifeCycle




EET: Extended Event Trace
Erster Takt in der Clock
Timer
ClockDevice
mcTimer: 0
careTimer: 0
dspTimer: 0
kbdTimer: 0

Vorgehensweise

- Reihenfolge der Erstellung der verschiedenen Sichten nicht vorgeschrieben
- Hierarchiekonzept:

Top	$\rightarrow$	**Down**
Komponente	$\rightarrow$	Unter-SSD
Zustand	$\rightarrow$	Unter-STD

- sinnvolle Reihenfolge:
  - SSD $\rightarrow$ STD
  - DTD zwischendurch
  - EET v.a. bei Simulation der Unterkomponenten

Handbuch, Features

- beiliegende Texte liefern keine Richtlinien zur Abbildung der realen Welt in das System
- Features
  - Simulation (erzeugt Java-Code)
  - Konsistenzprüfung
  - Schnittstelle nach außen
  - Java => Plattformübergreifend
  - Client-Server-Architektur => Mehrbenutzerbetrieb

Konkrete Vorgehensweise bei Modellierung

- Besprechung der Komponentenübersicht
  => Skizze auf Papier
- Entwicklung des **LifeCycles** und der **Clock**
  (parallel zur Einarbeitung)
  => Anforderungen an die Memory (Speicherbausteine, Kanäle)
- Entwicklung der **Memory**-Komponente
- Entwicklung der **Care**-Komponente
- Entwicklung der **Keyboard**-Komponente

Aufteilung der Anforderungen, Schnittstellen

- Aufteilung:

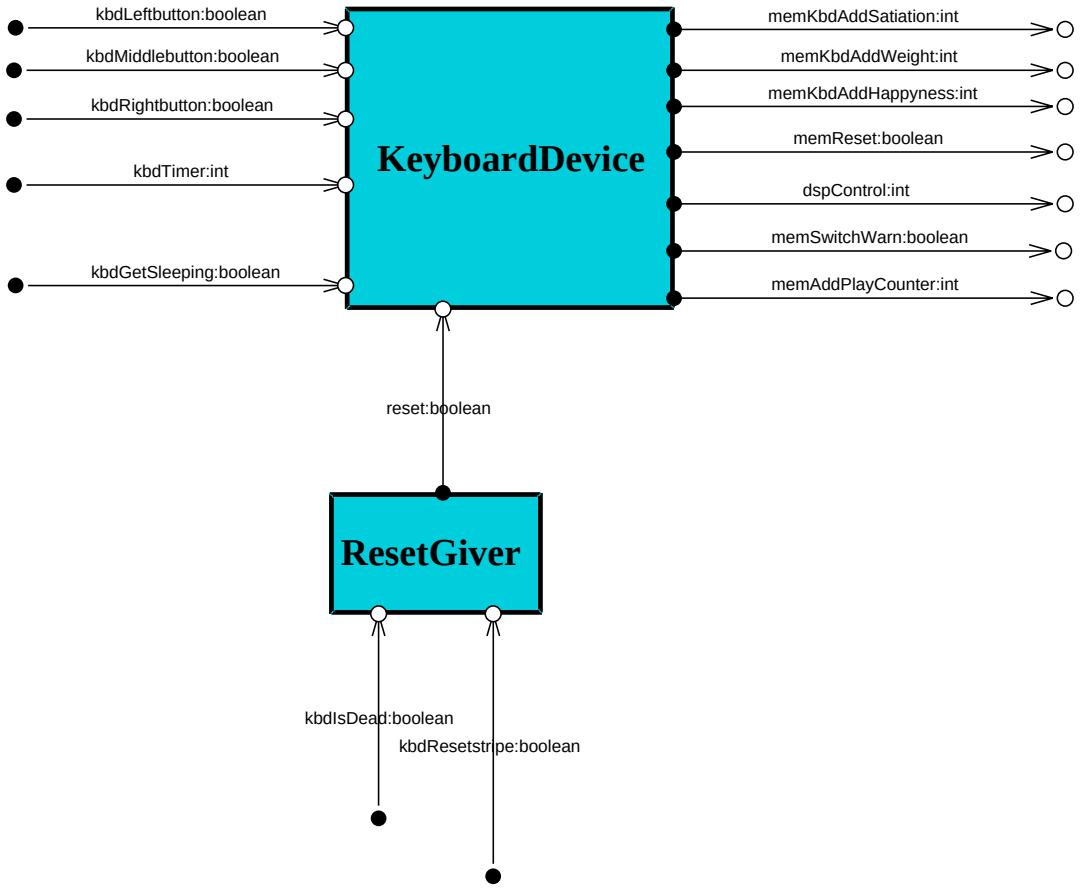
Alexander Seitz	LifeCycle
Thomas Seliger	Clock
	Keyboard
Florian Kunkel	Memory
	Care

- Schnittstellendefinition:
  - Vorgehensweise bei Besprechung der Komponentenübersicht
  - Bei Entwicklung von **LifeCycle** und **Clock** Ansammlung an Schnittstellenbedarf (v.a. an **Memory**)

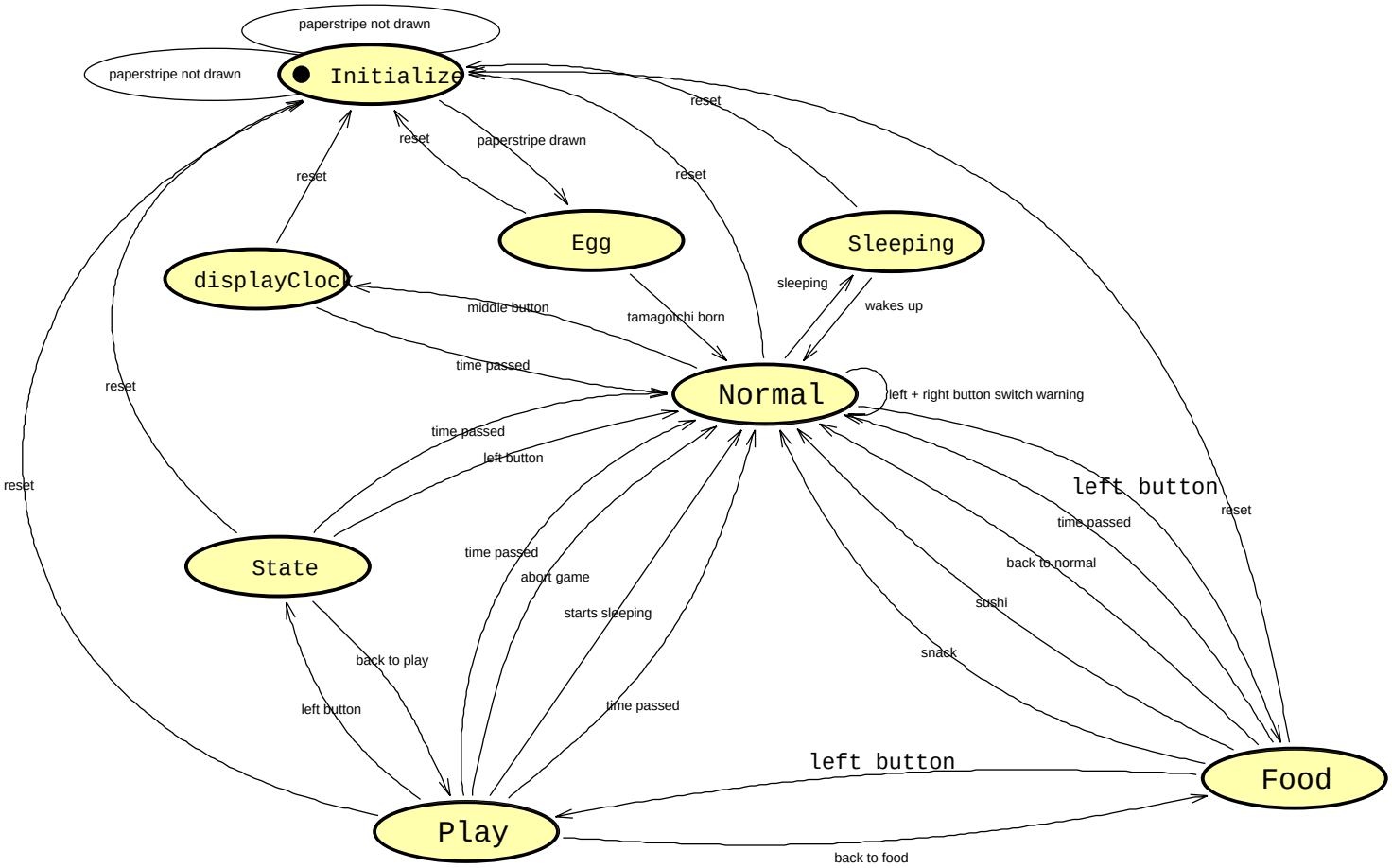
Anmerkungen zur Modellierung

- noch zu verbessern:
  - Display fehlt,
  - keine Überprüfung der Wertebereiche,
  - kann nicht verhungern und
  - piept jede Minute, wenn ein Wert nicht optimal ist
  - nicht jeder Zustandsübergang deterministisch
- Vereinfachung: Zeitverhalten $\rightarrow$ Minutentakt
- Unterkomponenten wurden während Modellierung simuliert.
- Anzahl der Diagramme: 6 SSDs, 33 STDs

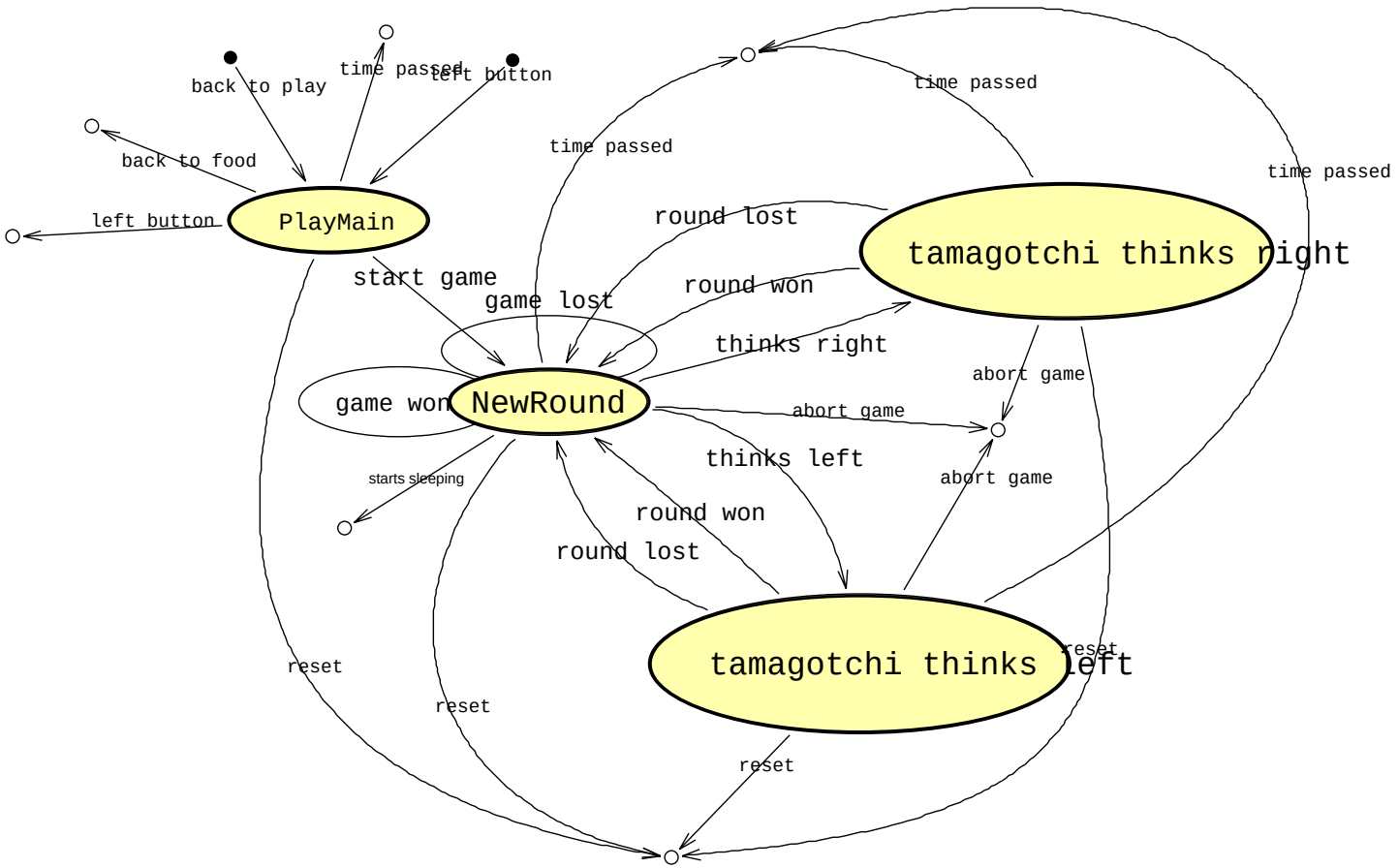
Sub-SSD Keyboard



STD Keyboard



Sub-STD Play


back to play
time passed
left button
back to food
left button
PlayMain
start game
game lost
game won
NewRound
starts sleeping
time passed
time passed
time passed
round lost
round won
tamagotchi thinks right
thinks right
abort game
abort game
thinks left
abort game
round won
round lost
tamagotchi thinks left
reset
reset
reset
reset

→ NewRound →

start game	
Input	reset?, kbdMiddlebutton?true, kbdTimer?x
Output	dspControl!41[1]
Postcondition	Round = 0; RoundsWon = 0; Savetimer = x

thinks left	
Precondition	Round < 5
Input	reset?, kbdGetSleeping?false
Output	dspControl!41[1]

thinks right	
Precondition	Round < 5
Input	reset?, kbdGetSleeping?false
Output	dspControl!41[1]

[1] 41 = Playing Tamagotchi

tamagotchi thinks left → NewRound

round won	
Input	reset?, kbdMiddlebutton?true
Output	dspControl!44[2]
Postcondition	Round++; RoundsWon++

round lost	
Input	reset?, kbdLeftbutton?true
Output	dspControl!45[3]
Postcondition	Round++

[2]44 = Playing Tamagotchi looking left (happy)

[3]45 = Playing Tamagotchi looking left (sad)

NewRound → NewRound

game won	
Precondition	RoundsWon > 2 && Round == 5
Input	reset?, kbdTimer?x, kbdGetSleeping?false
Output	dspControl!41[4], memKbdAddHappyness!1, memKbdAddWeight!-1, memAddPlayCounter!1
Postcondition	Savetimer = x; Round = 0; RoundsWon = 0

game lost	
Precondition	RoundsWon < 3 && Round == 5
Input	reset?, kbdTimer?x, kbdGetSleeping?false
Output	dspControl!41[4], memKbdAddWeight!-1, memAddPlayCounter!1
Postcondition	Savetimer = x; Round = 0; RoundsWon = 0

[4] 41 = Playing Tamagotchi

Informelle Anforderung - Umsetzung

- Grobgliederung der verschiedenen Komponenten durch (Sub)SSDs
- Funktionalität der Komponenten wird in den (Sub)STDs spezifiziert
- Zustände des Tamagotchi werden durch Zustände in den STDs repräsentiert
- Timer wird durch Hochzählen einer Variablen implementiert
- Im Speicher abgelegte Werte werden in jedem Takt neu ausgegeben

Einarbeitung (Safety-Injection-System)

- ca. 16 Stunden Einarbeitung und Erstellung
- Erlernbarkeit
  - Nur kurze Einführung vorhanden, keine genaueren Beschreibungen oder Tutorials
  - Erster Einstieg relativ schnell und problemlos, aber umständliche Bedienung
  - Keine Beschreibung der (aktuellen) Funktionalität des Programms (was funktioniert, was nicht)
  - Anfängliche Schwierigkeiten mit dem Takt- und Channelkonzept

Tamagotchi Gesamt-Aufwand

- ca. 140 Stunden Gesamtaufwand
- Grobgliederung des Systems und Schnittstellendef.; ca. 15%
- Erstellen der Module und Automaten; ca. 25%
- Probleme durch AutoFocus (Bugs, Bedienung, fehlende Dokumentation, etc.); ca. 50%
- Test und Debugging; ca. 10%

Tamagotchi Modellierung

- Notation
  - Übersichtliche Darstellung der SSDs und STDs
  - Gliederung in Sub-Diagramme (SSD und STD)
  - Die vielen Kanäle zwischen den SSDs verwirren ein wenig
  - Namen von Ports / Channels geraten bei Änderungen leicht durcheinander
- Anfangsschwierigkeiten
  - Schwierigkeiten bei der Kombination von Ports und Variablen in den Transitions-Attributen
  - Sichtbarkeit der Variablen unklar (Sub-SSDs)

Tamagotchi Modellierung

- Tool
  - Simulation auch von einzelnen Komponenten
  - Consistency-Checks
  - Nachträgliche Änderungen sind kompliziert durchführbar
- Anfangsschwierigkeiten
  - Fehler werden erst bei der Simulation sichtbar, keine Überprüfung der Daten bei der Eingabe
  - Fehler schwer lokalisierbar (keine direkten Fehlermeldungen des Tools, sondern indirekt durch den Java-Compiler)
  - Bei Änderung einer Automatenübergreifenden Transition müßen die Atrribute an drei Stellen geändert werden

Vergleich mit ObjecTime

- Ähnliche Konzepte wie bei AutoFocus
- Durch Gliederung und Substrukturen gute Übersicht
- Übersichtlichere Darstellung der Kanäle
- Es gibt kontextsensitive Menues

Teamarbeit unter AutoFocus

Gute Parallelisierung der Modellierung möglich:

- Client Server Architektur des Tools
- Aufteilung der Modellierung in Subkomponenten die getrennt voneinander bearbeitbar sind
- Simulation von Unterkomponenten möglich, Möglichkeit der Fehlersuche in Unterkomponenten vor Test des Gesamtsystems

Stärken von Notation/Methode/Tool

- Übersichtliche, leicht begreifbare Darstellung
- Automaten mit Zustandsübergängen (STDs)
- Strukturdiagramme mit Nachrichtenkanälen (SSDs)
- Möglichkeit ein Spezifikation erst oberflächlich zu bearbeiten und später genauer zu spezifizieren
- Unterstützung von Gruppenarbeit (Client/Server)

Schwächen des Tools

- Wird ein Signal von mehreren Komponenten benötigt, muß zu jeder Komponenten ein Channel gezogen werden
- Komplizierte Menuführung und Oberfläche
- Fehlerhaftes 'cut and paste'
- Änderungen von Strukturen mit Substrukturen umständlich

Erweiterungen / Verbesserungen

- Erweiterung der Methode
  - 'Reset-Signal' für Automaten
  - Busobjekt um Signale an mehrere Komponenten weiterzureichen
- Verbesserungsvorschläge zum Tool
  - Brauchbare Dokumentation mit Tutorials
  - Objektorientierte statt dokumentenbasierte Datenverwaltung
  - Fehlerüberprüfung bei der Eingabe (z.B. Variablennamen)
  - Möglichkeit Substrukturen nachträglich zuzuordnen