Seminar „Werkzeuggestützte Modellierung des Tamagotchi“

mit ROOM/ObjecTime

Markus Fröhler (TUM), Michael Gnatz (TUM),
Marcus Lanzl (TUM)

1./2. Februar 1999

Universität Kaiserslautern
AG-SE
Fachbereich Informatik

Fraunhofer IESE
Einrichtung Experimentelles Software Engineering

TUM
TECHNISCHE UNIVERSITÄT MÜNCHEN

ROOM = **R**eal-Time **O**bject-**O**riented **M**odeling

Autoren: Garth Gulleskon, Bran Selic, Paul Ward

Zweck: Unterstützung aller Phasen in der Entwicklung von Echtzeit-Systemen

Anforderungen: nebenläufig, verteilt, verzögerungsfrei

Features von ObjecTime

- inhärente Objektorientierung
- auf Echtzeit-Umgebungen zugeschnitten
- Export der Architektur zur Dokumentation
- ausführbare Modelle auf allen Abstraktionsebenen

  → frühe Fehlererkennung
- schrittweiser, iterativer Entwicklungsprozeß

  → keine Sprünge im Modell

Konzept der Notation

- Structure-Sicht
- Behavior-Sicht

} Actor

- grundsätzlich beliebig tiefe Rekursion
- Kommunikationsprotokolle
- Message sequence charts
- Vererbung von Aktoren und Protokollen
- unabhängige Ausführbarkeit der Aktoren

Methodik der Modellierung

- Grobstruktur → Feingranularität
- Definition von Aktoren
- Definition von Kommunikationsprotokollen
- Definition von Ports, Kanälen
- Zuordnung: Kanal → Protokoll
- Erweiterte Zustands-Automaten

Komponenten von ObjecTime

- Model-Management
- Model-Editor
- Model-Validator
- Model-Compiler
- Model-Execution
- Cross Reference, Navigation

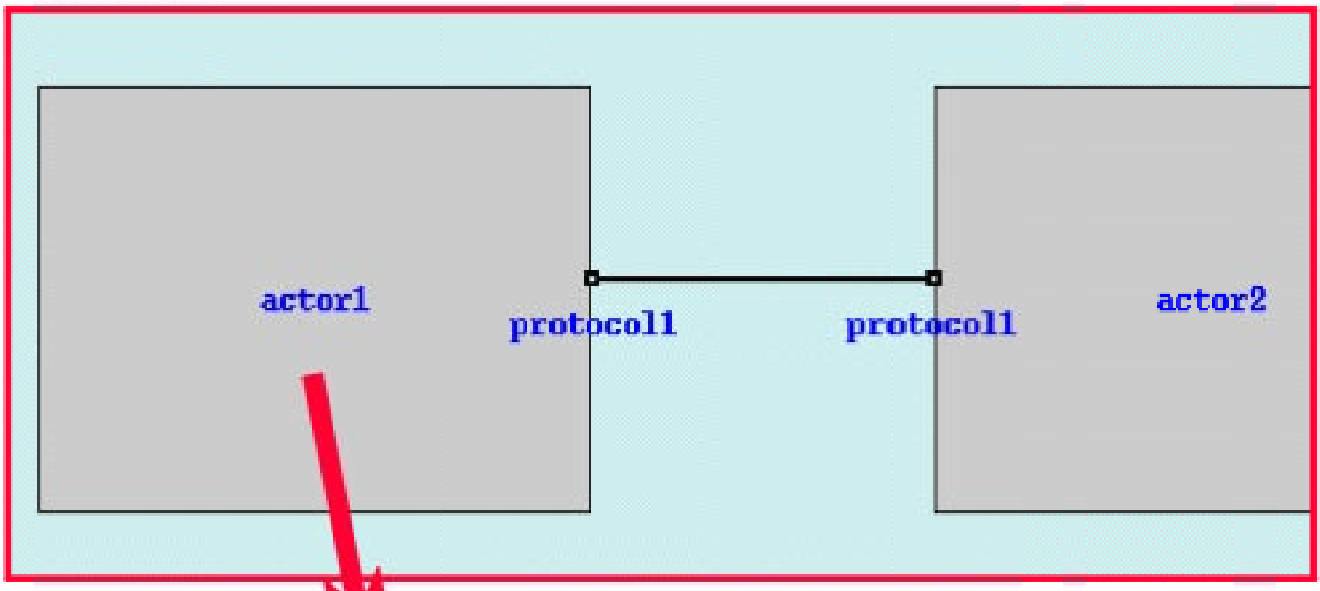
Modellierung im Team

- Vorgehen gemäß der Empfehlungen (Folie 5)
- kein verteiltes Arbeiten/Versionsmanagement
- Modellierung aller wesentlichen Aspekte
- Simulation ab ca. 50% der Modellierung
- Größe der Spezifikation:

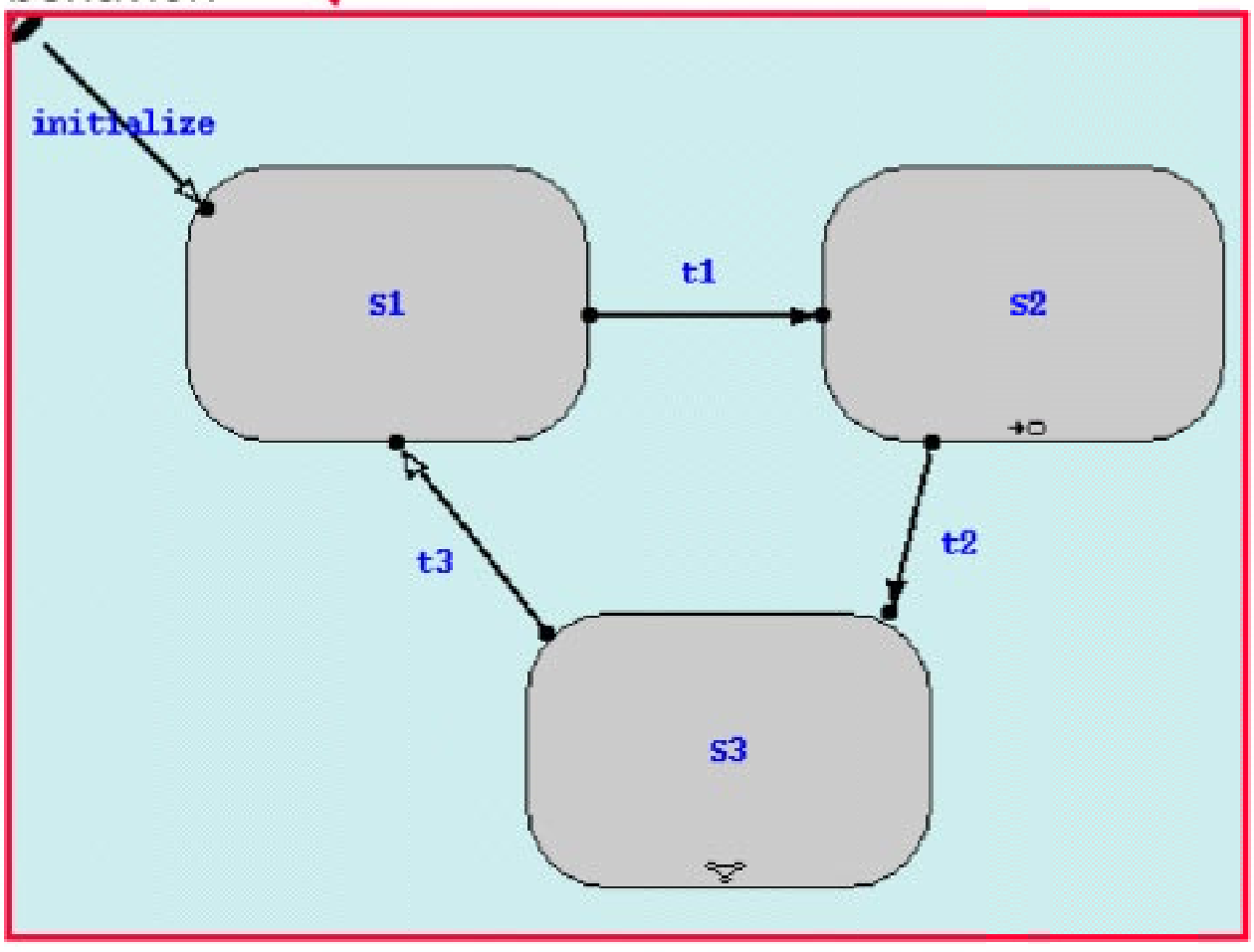
  25 Diagramme $\rightarrow$ 5500 Lines of C++-Code

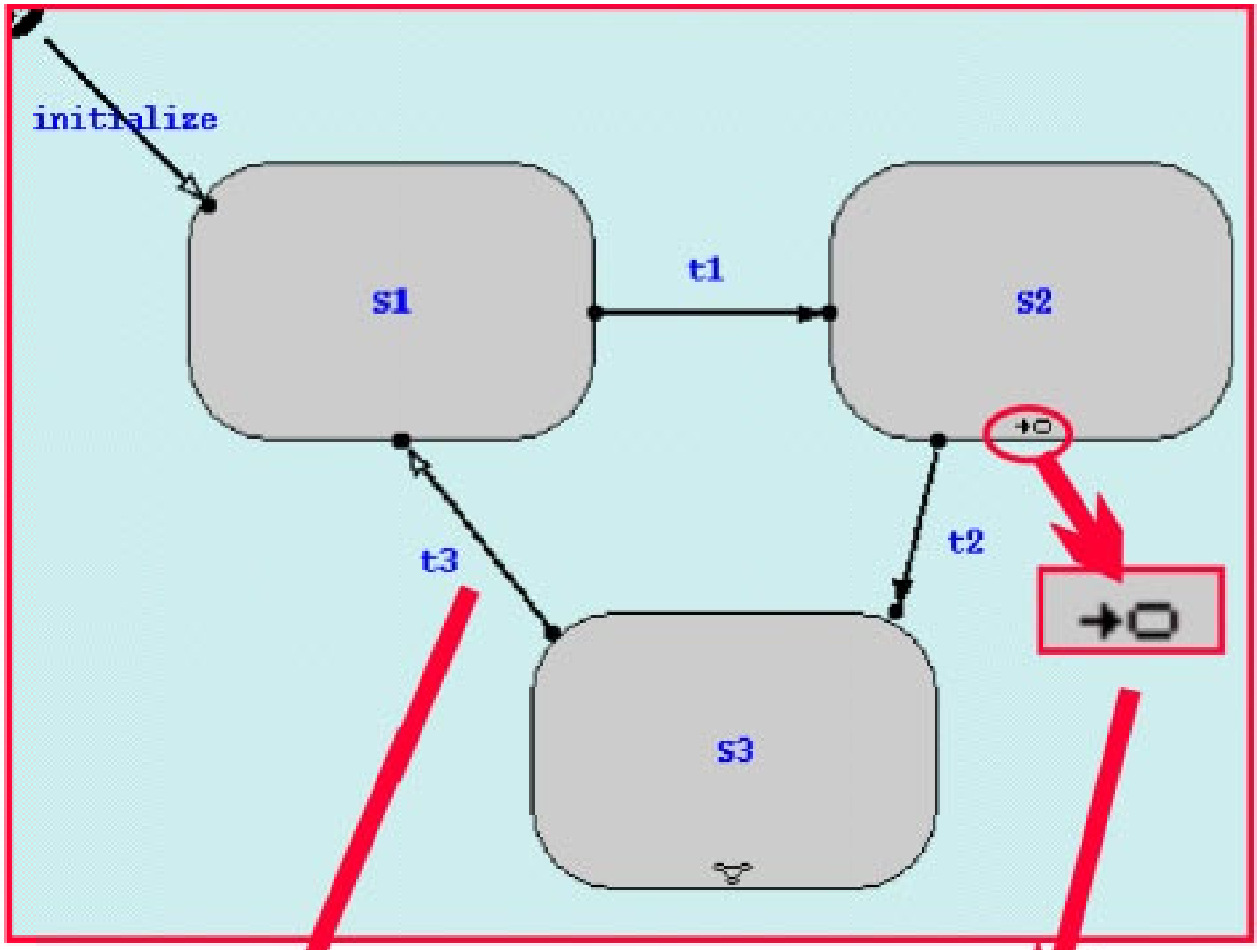
Überblick über die Notation

structure:



behavior:





Entry-Code:

C++ ->(Actor1:S2 | Code View

```
/* ***********************
   beliebiger C++-Code
   als Entry/Extry-Code
   ***********************/
```

Transitions-Code:

C++ --> Actor1:t1 | Cod

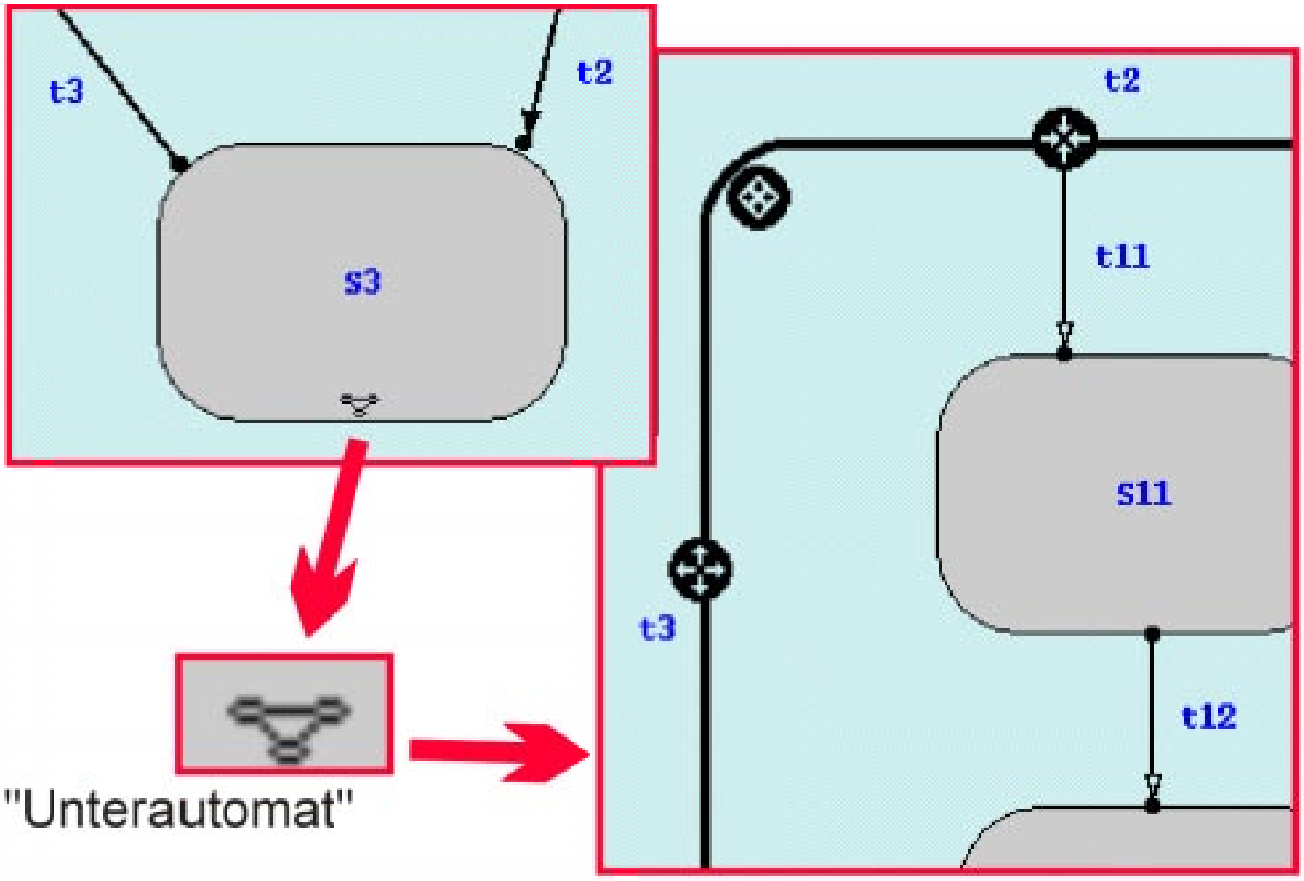
```
Protocol1.send(testSignal);

/* ***********************
   beliebiger C++-Code
   ***********************/
```

Signal versenden

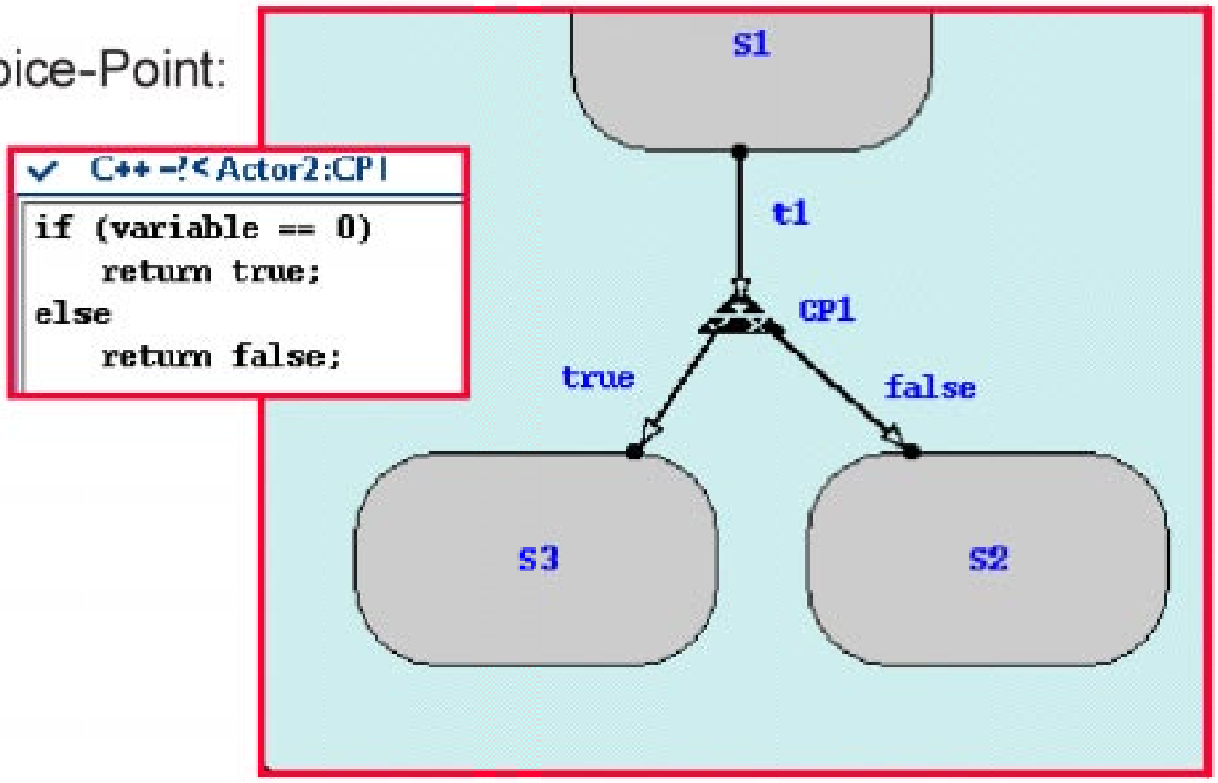
Signale timergesteuert erzeugen:

C++ --> Actor1:t2 | Code View

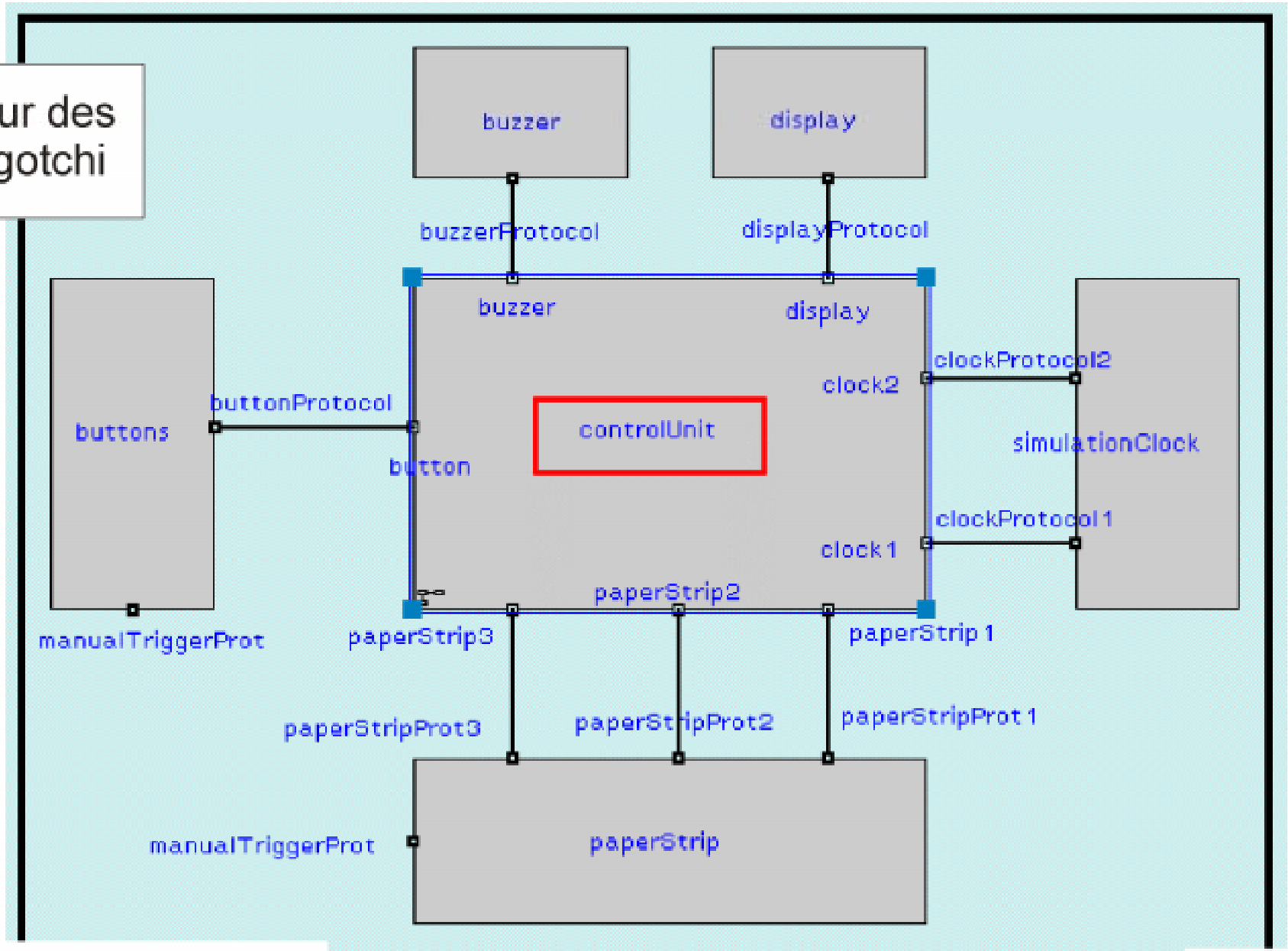
```
timerId = timingSAP.informIn(intervall);
```

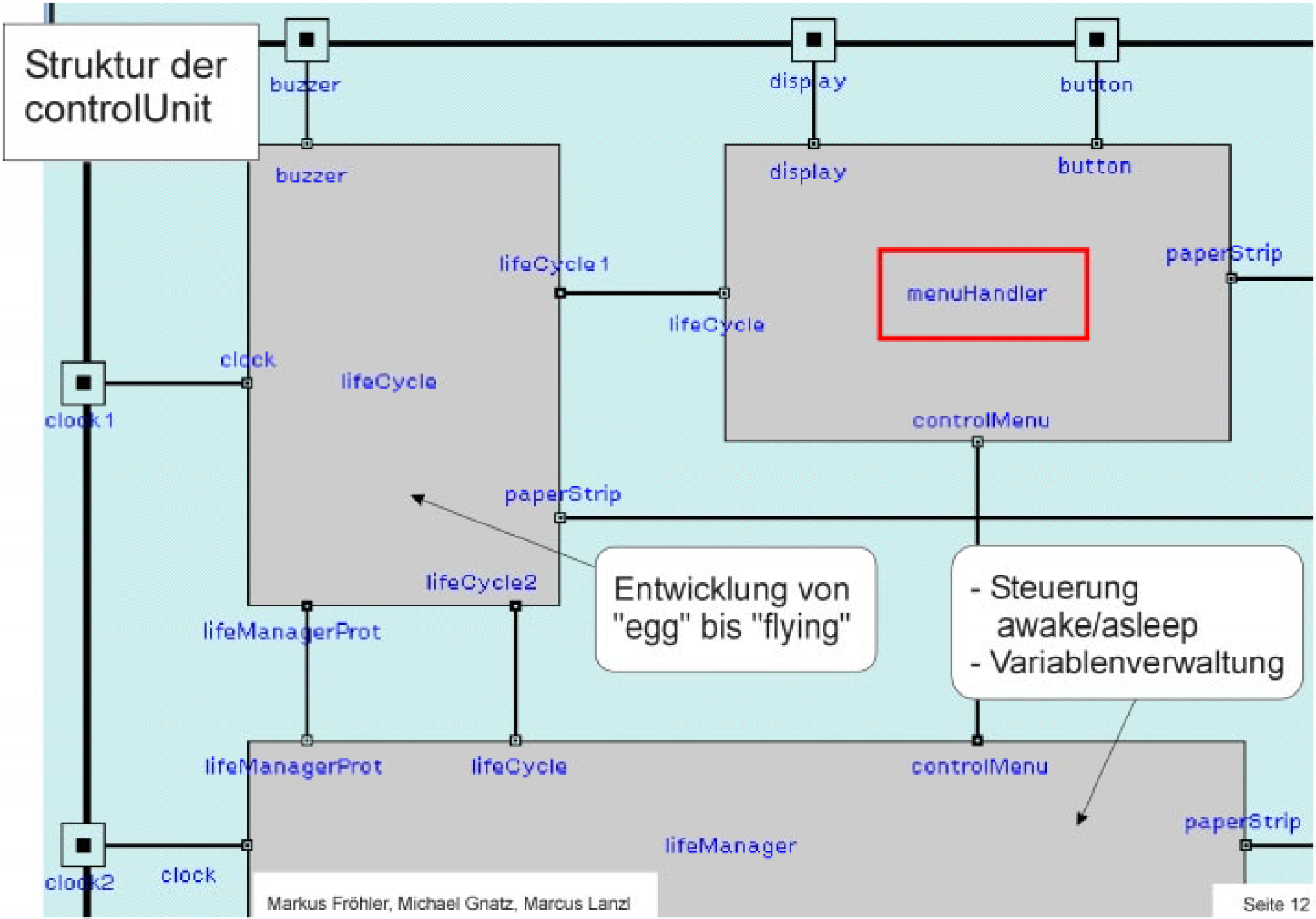


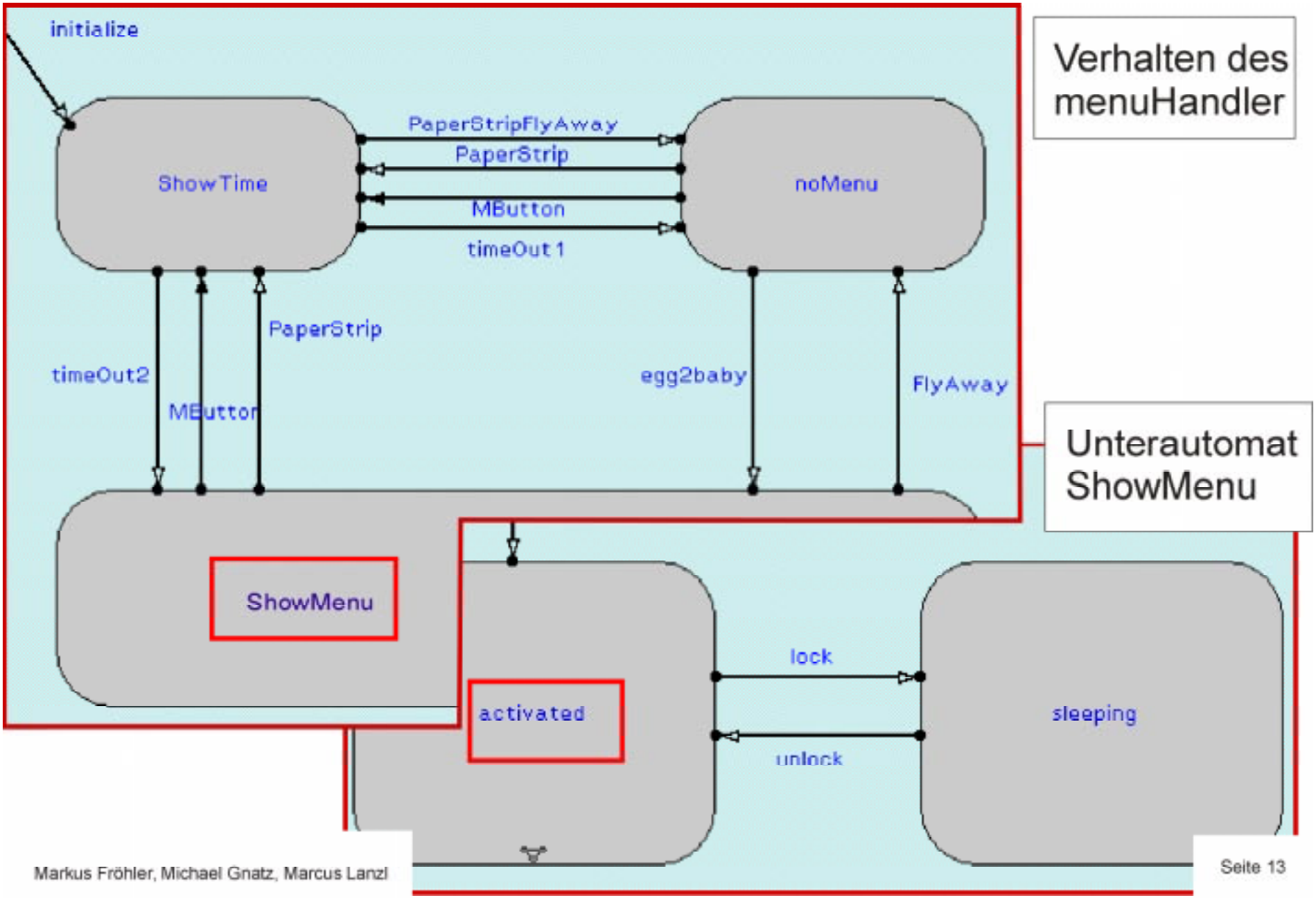
Choice-Point:



Struktur des Tamagotchi

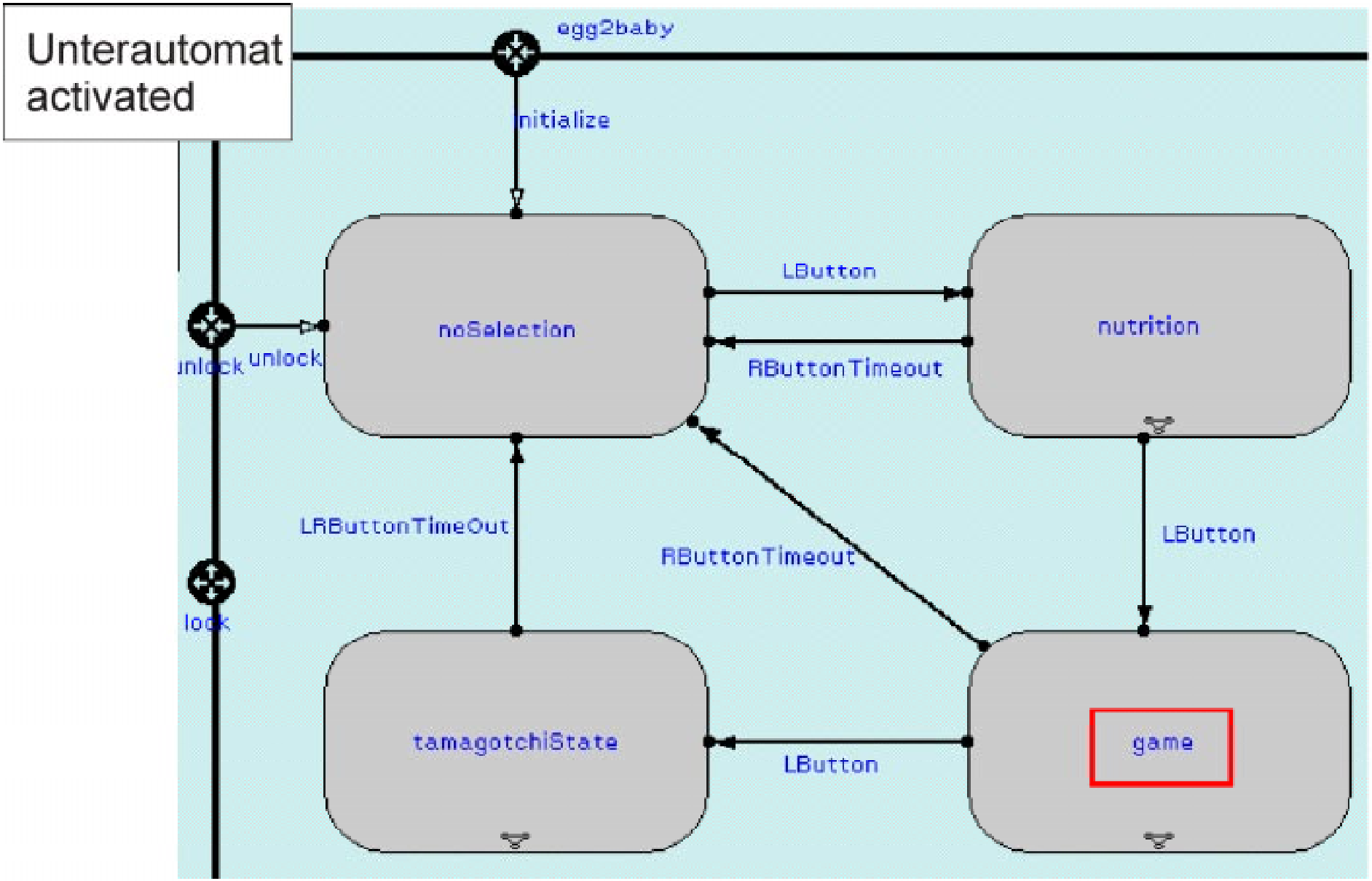


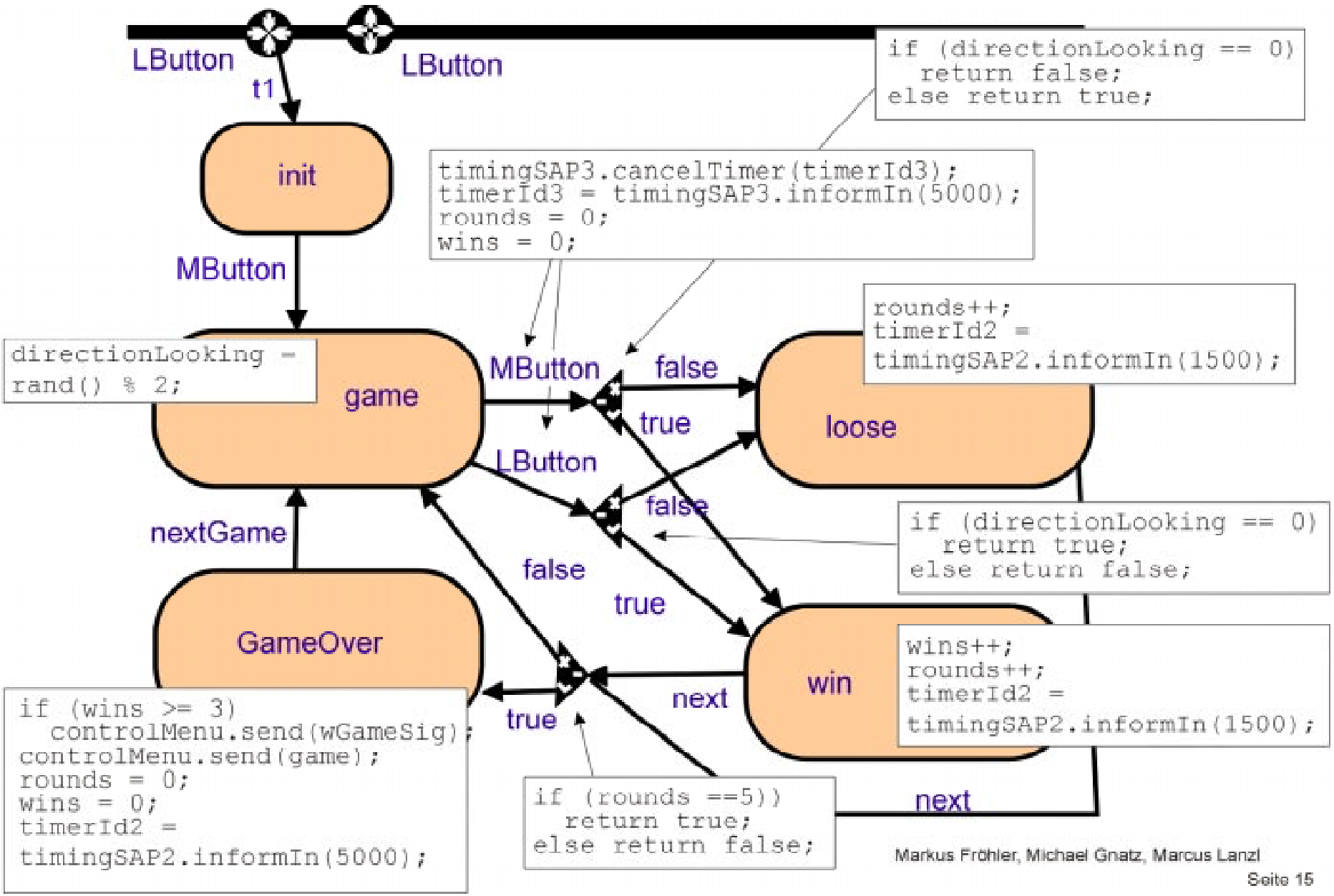

Struktur der controlUnit
buzzer
display
button
buzzer
display
button
lifeCycle1
lifeCycle
menuHandler
paperStrip
clock
lifeCycle
clock1
controlMenu
paperStrip
lifeCycle2
Entwicklung von "egg" bis "flying"
- Steuerung awake/asleep
- Variablenverwaltung
lifeManagerProt
lifeManagerProt
lifeCycle
controlMenu
paperStrip
lifeManager
clock2
clock
Markus Fröhler, Michael Gnatz, Marcus Lanzl
Seite 12



Verhalten des menuHandler

Unterautomat ShowMenu


Unterautomat activated
egg2baby
initialize
noSelection
LButton
nutrition
RButtonTimeout
unlock
unlock
LRButtonTimeOut
RButtonTimeout
LButton
lock
tamagotchiState
LButton
game


LButton
t1
LButton
init
MButton
game
directionLooking =
rand() % 2;
timingSAP3.cancelTimer(timerId3);
timerId3 = timingSAP3.informIn(5000);
rounds = 0;
wins = 0;
if (directionLooking == 0)
return false;
else return true;
MButton
false
true
loose
rounds++;
timerId2 =
timingSAP2.informIn(1500);
LButton
false
nextGame
if (directionLooking == 0)
return true;
else return false;
false
true
GameOver
win
wins++;
rounds++;
timerId2 =
timingSAP2.informIn(1500);
next
true
if (wins >= 3)
controlMenu.send(wGameSig);
controlMenu.send(game);
rounds = 0;
wins = 0;
timerId2 =
timingSAP2.informIn(5000);
if (rounds ==5))
return true;
else return false;
next

Einarbeitung

- Aufwand ca. 50 Stunden (incl. Reaktor)
- „Tutorial“ reine Bedienungsanleitung
- Online-Hilfe/Handbücher lückenhaft
- zusätzliche Literatur nötig (ROOM)

Modellierung

- Aufwand ca. 110 Stunden
- Notation sehr mächtig
- Gleichzeitigkeit von Signalen fehlt
- Notation vom Tool voll unterstützt
- jedoch Mängel in der Implementierung

Simulation

- Verständlichkeit:
  - 👍 graphische Darstellung
  - 👎 versteckte Codestücke
- Verfolgbarkeit:
  - 👍 Stichwortsuche und Navigation
  - 👍 Simulationsmöglichkeiten
  - 👎 hohe Schachtelungstiefe

Simulation

- Überprüfbarkeit:
  - 👍 hilfreiche Fehlermeldungen vom Tool
  - 👍 Konsistenzprüfungen beim Editieren
  - 👎 Umfang
  - 👎 Fehlermeldungen des Compilers

Review

- Viele Ähnlichkeiten mit AutoFocus festgestellt

Bewertung Modellierung

- sehr gut geeignet für verteilte Systeme
- berücksichtigt Zeitaspekte
- nicht ideal für das Tamagotchi

Bewertung Tool

- Einsparung von Tipparbeit
- Fehlererkennung oft schon beim Eingeben
- gute Visualisierung

Bewertung allgemein

- paralleles Arbeiten schwer möglich
- Stärken
  - 👍 keine Abstürze des Tools
  - 👍 intuitiver Aufbau
  - 👍 gute Simulationsmöglichkeiten
  - 👍 einfaches Timerkonzept

allgemeine Bewertung

- Schwächen
  - zu viele offene Fenster
  - schlechte Dokumentation
  - Implementierungsfehler

ObjecTime ist noch nicht perfekt, aber ein sinnvolles Werkzeug, das die Entwicklung in vielfältiger Weise unterstützt.