1 Statecharts language and Statemate Magnum

1. 1 Introduction

The language of Statecharts has been developed to deal with the problems of specification and design of large reactive systems. The basic foundation for the *Statechart* language is the *finite state machine*. David Harel extended this theory in [Harel87] by adding the following features:

- Clustering and refinement of states.
- A history mechanism for groups of states, which allow substates to have a memory.
- The possibility to deal with concurrency.
- Hybrid State Machines.
- The possibility to deal with conditioned events.

This extended theory is called *Harel machine*. Harels new concept allows the combination of *Moore-* and *Mealy Machines* thus leading to the expression of *Hybrid State Machines*.

Statecharts are used to depict the behavioral view of a system. Overall, there are three different views:

1. The *Module-charts* describe the interaction of the System with the environment. This is the structural view, which captures the „how" (Figure 1).
2. The *Activity-charts* are used to describe the functional view („what"). Different system activities and the information flow between them are shown (Figure 2).
3. The *Statecharts* describe the functionality of these activities with states and transitions. This behavioral view captures the „when" (Figure 3).

The usage of these languages will be explained with the Safety-Injection example. Figure 3 shows the Statechart for this system. Several states represent the different modes in which the system can be. Moving from one state to another is accomplished by different events that can cause various types of action. An action can take place when entering a certain state, leaving it, or during the whole time in that state. Rounded rectangles are used to denote states at any level. Arrows labeled with an event are used to denote transfers between states.
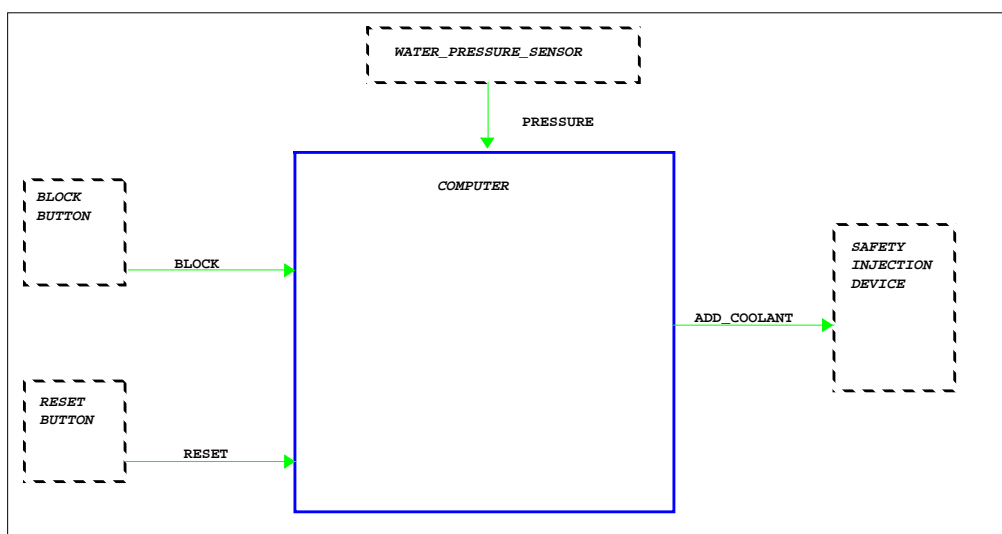

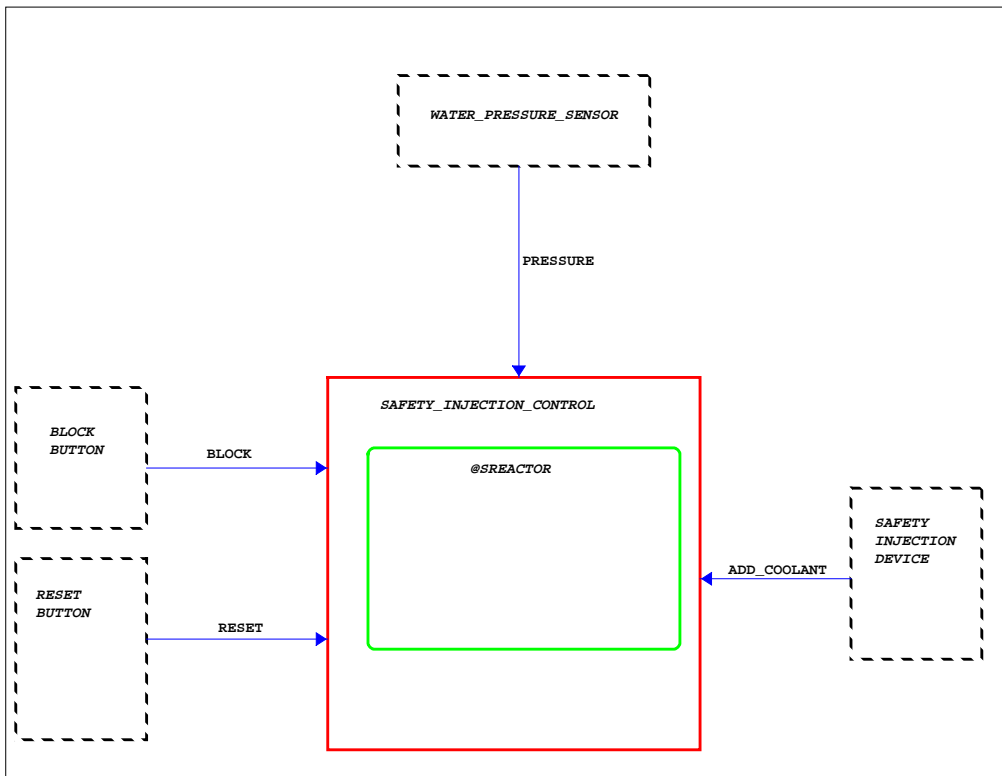Figure 1: Module-Chart of the Safety-Injection example

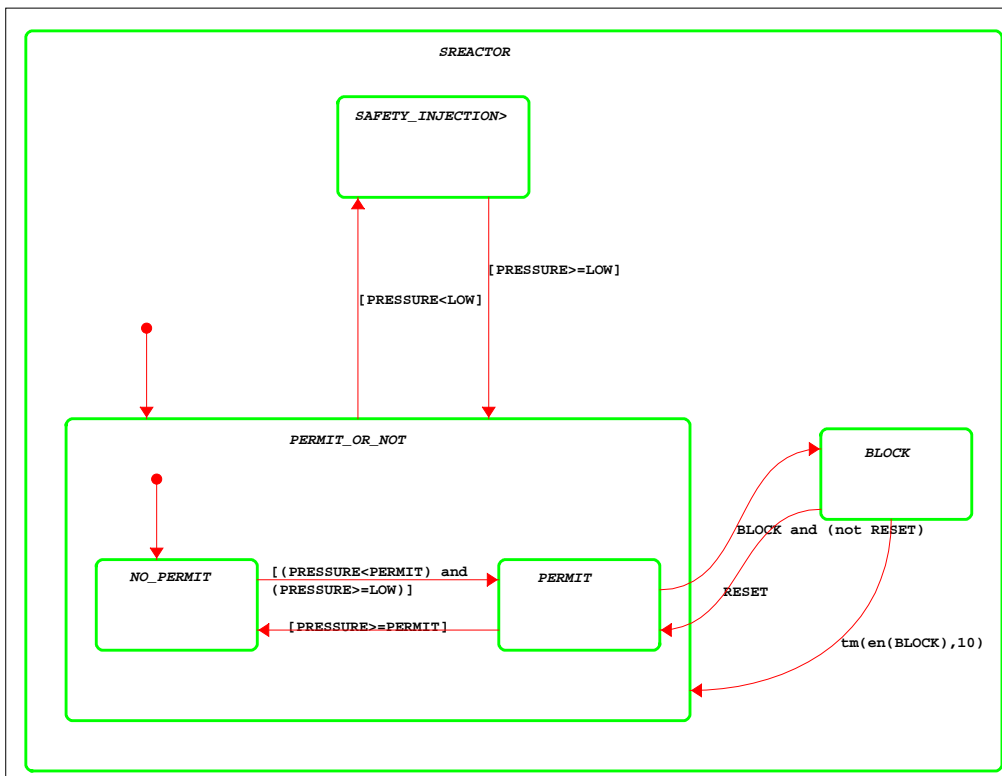Figure 2: Activity-Chart of the Safety-Injection example

Figure 3: Statechart of the Safety-Injection example

Datadictionary:

```
Information-flow   ADD_COOLANT
 Defined in chart: MREACTOR
 Contains: START_COOLANT
           STOP_COOLANT


Module   BLOCKBUTTON
 Defined in chart: MREACTOR
 Type: External


Module   COMPUTER
 Defined in chart: MREACTOR
 Type: Execution
 Purpose: Regular
 Described by activity-chart: AREACTOR


Data-item   LOW
 Defined in chart: SREACTOR
 Usage: Variable
 Data type: Integer length=8


State   NO_PERMIT
 Defined in chart: SREACTOR
 Type: Basic


State   PERMIT_OR_NOT
 Defined in chart: SREACTOR
 Type: Or


State   SAFETY_INJECTION
 Defined in chart: SREACTOR
 Type: Basic
 Static reactions:
 entering/START_COOLANT;;
 exiting/STOP_COOLANT


Activity   SAFETY_INJECTION_CONTROL
 Defined in chart: AREACTOR
 Type: Non-Basic
 Termination type: Reactive Controlled


Event   RESET
 Defined in chart: MREACTOR
 Usage: Variable
 Structure: Event
```

To specify time-related items, predefined time-out events are offered. These events automatically occur within a certain time after an other event took place. There is an unlimited number of self definable variables available to design data properties. By using superstates and substates the specification requirements can be divided into manageable subsystems. States and transitions are drawn in a graphical editor; events, actions, and variables are textually described (see Datadictionary). All activities may be decomposed into subactivities, until the system has been specified in terms of basic activities.

To verify a model each state (with its entry- and exit criteria and each transition) has to be checked or simulated.

There is no formal approach as how to transform given requirements into the three diagra types and how to model the behavior in the statecharts with states and transitions and the extended features of the *Harel machines*.

Also missing is a formal criteria telling you when you are finished with you description of the system; you have to check whether you modeled every feature action and reaction of the system.

The tool *Statemate Magnum* was developed by the company i-logics. As a founder of i-logics, David Harel himself was involved in the development of this tool.

The tool requires a top down approach to generate the Module-charts, then the Activity-charts and finally the Statecharts. How to implement the Statecharts is left fully to the designer of the system. There is no approach given, but with a little experience modelling can be performed intuitively.

The main features of *Statemate* are:

It consists of a very vivid simulation feature.

Charts are tested for violation in syntax, semantics and also the relation between charts is checked. *Statemate* detects inconsistencies in the model, such as an Or-state without a default entrance and it detects redundancy and incompleteness in the model, such as unresolved elements.

The Code Generator generates a source code that reflects the same behaviour as the original model. This code can then be used to:
   - Develop prototypes
   - Simulate and/or validate large-scale systems

There is also a Panel Graphic Editor (PGE) which allows you to draw a mock-up of the system´s interface, and to bind the components of the mock-up panel to the *Statemate* model. These panels may also be included as part of the generated code.

The Documentor can be used to design and produce the documentation for the designed system. *Statemate* can include textual and graphical information from a variety of sources, including the project database and external files.

1. 2 Approach and Result

We tried to divide the whole specification into several parts which could be developed independently. First of all it was necessary as a team to determine the architecture of the system. We had to delimit the functionalities of the tamagotchi such as playing, eating and so on. Later, each functionality was mapped to an Activity-Chart. The refinement of these Activity-Charts lead into several State-Charts. It was necessary to determine the top-level information-flow between

the Activity-Charts before splitting up the system into parts which could be developed by each team member.

Due to the functional decomposition of the system, the possibility was confirmed to develop each part independently. The tool provided version-management so we could work parallel on each activity of the tamagotchi. We defined one control activity which managed the life-cycle of the tamagotchi. This control unit worked together with all the other activities, reducing the information-flow between the other parts of the system and thus we could develop the activities like playing, eating, menu-control, buzzer, weight- and luck-management independently. These activities have also been simulated separately during the modeling phase.

We covered all details of the specification. It was not necessary to simplify the requirements of the tamagotchi because the Harel machine provided all the features we needed.

Statemate Magnum supports a consistency-check so the consistency of our specification is guaranteed. Furthermore we tested our system with the simulator against the textual requirements. No errors remained except the reset of the tamagotchi with the paper-strip. This part of the requirements has not been worked out but it could be easily completed.
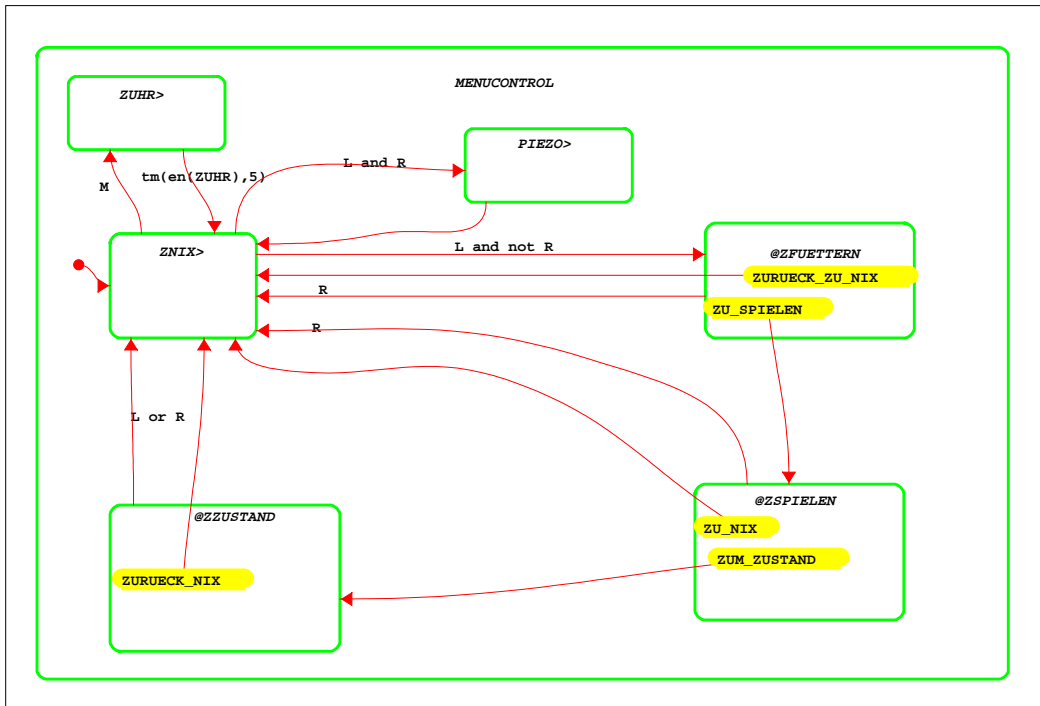
The specification comprises 30 charts. A printed version with all data-dictionary entries takes up about 40 pages.

1. 3 Tamagotchi Specification of the functionality "Playing"

In our specification the functionality "Playing" is distributed across several statecharts. Within the chart 'Menucontrol' there is the most important chart for the playing functionality: the chart named 'Zspielen'. In this chart the game itself is performed while the chart 'Displaysteuerung_control' decides the appearance of the playing tamagotchi. This control-state for the display determines the development phase of the tamagotchi and also decides what to display.

Within these states there are several substates that care about the different kinds of behavior the tamagotchi can perform (e.g. playing, laughing, crying).

Below each state-diagram you find the corresponding datadictionary which mainly describes entry and exit activities of several states.

MENUCONTROL

ZUHR>

PIEZO>

L and R

M   tm(en(ZUHR),5)

ZNIX>

L and not R

@ZFUETTERN

ZURUECK_ZU_NIX

R

ZU_SPIELEN

R

L or R

@ZZUSTAND

@ZSPIELEN

ZU_NIX

ZURUECK_NIX

ZUM_ZUSTAND

Part of BA-F23

ZUM_ZUSTAND

ZU_NIX

L   tm(en(ZNIX),5)

ZNIX>

M

ZSPIELEN

SPIEL

ZNEUES_SPIEL>

tm(en(ZGEWONNEN),5)[ANZ<5]

ZLINKS_GEWAEHLT>

[ZUFALL=0]

ZGEWONNEN>

L

tm(en(SPIELSTAND),5) or L or M

[ZUFALL=1]

ZSTART>

[ZUFALL=1]

SPIELSTAND>

M

ZRECHTS_GEWAEHLT>

[ZUFALL=0]

ZVERLOREN>

/GESPIELT

tm(en(ZVERLOREN),5)[ANZ<5]

tm(en(ZVERLOREN),5)[ANZ=5]

ZSPIEL_GEWONNEN>

[LOCALWIN<3]/GESPIELT

RUNDE_ENDE

[LOCALWIN>2]

tm(en(ZGEWONNEN),5)[ANZ=5]

Part of BA-F23, BA-F24, BA-F25

State ZNIX
  Static reactions:
   entering/MENUAUSGABE:='spielen';
  Elements used:
    TAMAGOTCHI:MENUAUSGABE (Variable Data-item) String length=30
Part of BA-F23

State ZNEUES_SPIEL
  Static reactions:
   entering/ANZ:=0;LOCALWIN:=0;WERTEANZEIGE:=SPIELEN;fs!(GEWONNEN);
  Elements used:
    ZSPIELEN:ANZ (Variable Data-item) Integer length=8
    ZSPIELEN:LOCALWIN (Variable Data-item) Integer length=8
    MAIN_ACTIVITY:WERTEANZEIGE (Variable Data-item) DISPLAYTYP
     Elements used:
      TYPES:DISPLAYTYP (User-defined Type) Enum-type
      Defined as:
       {NIX,ALTERGEWICHT,SATT,GLUECKANZ,UHR,LACHEN,WEINEN,SPIELEN,TSUSHI,TS
       NACK,ESSEN}
    TYPES:SPIELEN (Enumerated value): of Enumerated Type DISPLAYTYP
    MAIN_ACTIVITY:GEWONNEN (Condition)
Part of BA-F24

State ZLINKS_GEWAEHLT
  Static reactions:
   entering/ZUFALL:=RAND_IUNIFORM(0,1);
  Elements used:
    ZSPIELEN:ZUFALL (Variable Data-item) Integer min=0 max=1
    RAND_IUNIFORM(Predefined Function)
Part of BA-F24

State ZGEWONNEN
  Static reactions:
   entering/ANZ:=ANZ+1;LOCALWIN:=LOCALWIN+1;
   WERTEANZEIGE:=LACHEN;
  Elements used:
    ZSPIELEN:ANZ (Variable Data-item) Integer length=8
    ZSPIELEN:LOCALWIN (Variable Data-item) Integer length=8
    MAIN_ACTIVITY:WERTEANZEIGE (Variable Data-item) DISPLAYTYP
     Elements used:
      TYPES:DISPLAYTYP (User-defined Type) Enum-type
      Defined as:
       {NIX,ALTERGEWICHT,SATT,GLUECKANZ,UHR,LACHEN,WEINEN,SPIELEN,TSUSHI,TS
       NACK,ESSEN}
    TYPES:LACHEN (Enumerated value): of Enumerated Type DISPLAYTYP
Part of BA-F24

State ZSTART
  Static reactions:
   entering/WERTEANZEIGE:=SPIELEN;
  Elements used:
    MAIN_ACTIVITY:WERTEANZEIGE (Variable Data-item) DISPLAYTYP
     Elements used:
      TYPES:DISPLAYTYP (User-defined Type) Enum-type
      Defined as:
       {NIX,ALTERGEWICHT,SATT,GLUECKANZ,UHR,LACHEN,WEINEN,SPIELEN,TSUSHI,TS
       NACK,ESSEN}
    TYPES:SPIELEN (Enumerated value): of Enumerated Type DISPLAYTYP
Part of BA-F23

State ZRECHTS_GEWAEHLT
  Static reactions:
   entering/ZUFALL:=RAND_IUNIFORM(0,1);
  Elements used:
    ZSPIELEN:ZUFALL (Variable Data-item) Integer min=0 max=1
    RAND_IUNIFORM(Predefined Function)
Part of BA-F24

State ZVERLOREN
  Static reactions:
   entering/ANZ:=ANZ+1;WERTEANZEIGE:=WEINEN;
  Elements used:
    ZSPIELEN:ANZ (Variable Data-item) Integer length=8
    MAIN_ACTIVITY:WERTEANZEIGE (Variable Data-item) DISPLAYTYP
     Elements used:
      TYPES:DISPLAYTYP (User-defined Type) Enum-type
      Defined as:
       {NIX,ALTERGEWICHT,SATT,GLUECKANZ,UHR,LACHEN,WEINEN,SPIELEN,TSUSHI,TS
       NACK,ESSEN}
    TYPES:WEINEN (Enumerated value): of Enumerated Type DISPLAYTYP
Part of BA-F24

State SPIELSTAND
  Static reactions:
   entering/TAMAGOTCHIAUSGABE:=STRING_CONCAT('#gewonnen: ',INT_TO_STRING
   (LOCALWIN));
  Elements used:
    TAMAGOTCHI:TAMAGOTCHIAUSGABE (Variable Data-item) String length=30
    STRING_CONCAT(Predefined Function)
    INT_TO_STRING(Predefined Function)
    ZSPIELEN:LOCALWIN (Variable Data-item) Integer length=8
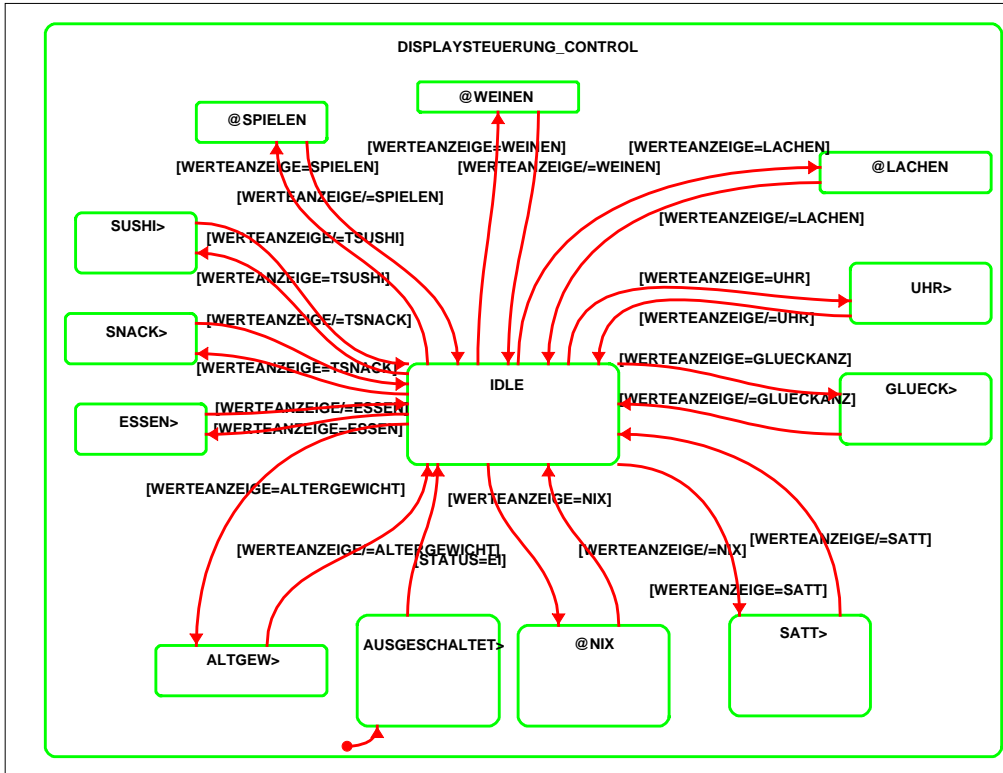## Part of BA-F25

State ZSPIEL_GEWONNEN
  Static reactions:
   entering/tr!(GEWONNEN);
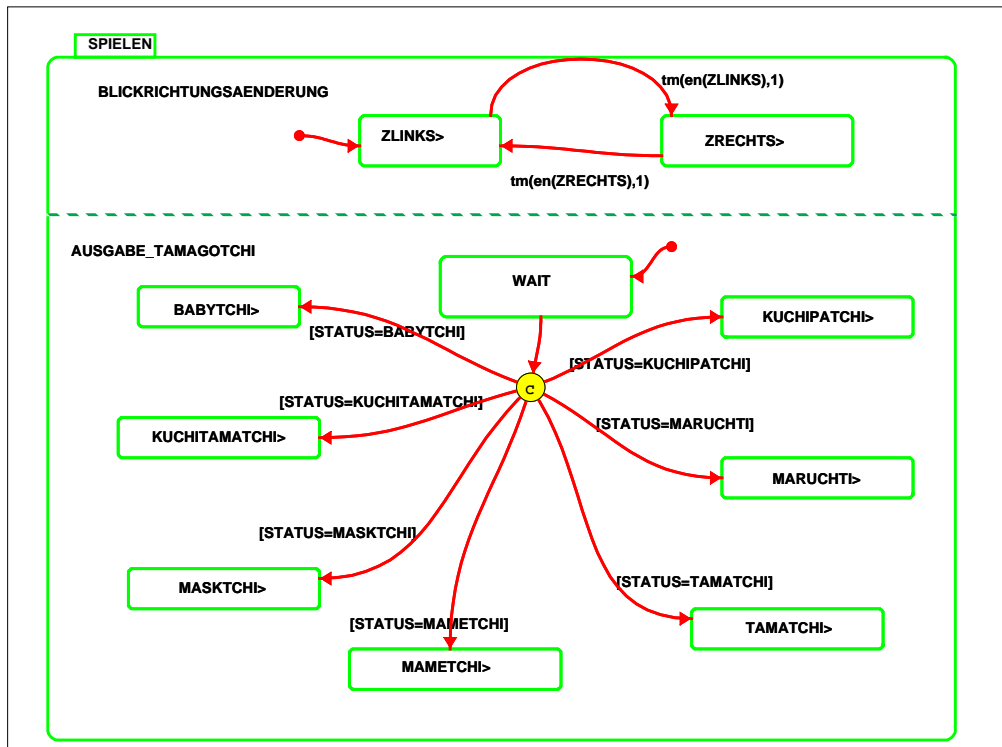 Elements used:
    MAIN_ACTIVITY:GEWONNEN (Condition)
Part of BA-F25

DISPLAYSTEUERUNG_CONTROL

@WEINEN

@SPIELEN

[WERTEANZEIGE=WEINEN]

[WERTEANZEIGE=SPIELEN]

[WERTEANZEIGE/=WEINEN]

[WERTEANZEIGE=LACHEN]

[WERTEANZEIGE/=SPIELEN]

@LACHEN

SUSHI>

[WERTEANZEIGE/=TSUSHI]

[WERTEANZEIGE/=LACHEN]

[WERTEANZEIGE=TSUSHI]

[WERTEANZEIGE=UHR]

UHR>

SNACK>

[WERTEANZEIGE/=TSNACK]

[WERTEANZEIGE/=UHR]

[WERTEANZEIGE=TSNACK]

IDLE

[WERTEANZEIGE=GLUECKANZ]

[WERTEANZEIGE/=ESSEN]

[WERTEANZEIGE/=GLUECKANZ]

GLUECK>

ESSEN>

[WERTEANZEIGE=ESSEN]

[WERTEANZEIGE=ALTERGEWICHT]

[WERTEANZEIGE=NIX]

[WERTEANZEIGE/=ALTERGEWICHT]

[WERTEANZEIGE/=NIX]

[WERTEANZEIGE/=SATT]

[STATUS=EI]

[WERTEANZEIGE=SATT]

ALTGEW>

AUSGESCHALTET>

@NIX

SATT>

Part of BA-F24

SPIELEN

BLICKRICHTUNGSAENDERUNG

tm(en(ZLINKS),1)

ZLINKS>

ZRECHTS>

tm(en(ZRECHTS),1)

AUSGABE_TAMAGOTCHI

WAIT

BABYTCHI>

KUCHIPATCHI>

[STATUS=BABYTCHI]

[STATUS=KUCHIPATCHI]

C

[STATUS=KUCHITAMATCHI]

[STATUS=MARUCHTI]

KUCHITAMATCHI>

MARUCHTI>

[STATUS=MASKTCHI]

MASKTCHI>

[STATUS=TAMATCHI]

[STATUS=MAMETCHI]

TAMATCHI>

MAMETCHI>

Part of BA-F23, BA-F24

State ZLINKS
  Static reactions:
   entering/LINKS
  Elements used:
    SPIELEN:LINKS (Event)
Part of BA-F24

State ZRECHTS
  Static reactions:
   entering/RECHTS
  Elements used:
    SPIELEN:RECHTS (Event)
Part of BA-F24

State BABYTCHI
  Static reactions:
   LINKS/TAMAGOTCHIAUSGABE:='Babytchi schaut links';;
   RECHTS/TAMAGOTCHIAUSGABE:='Babytchi schaut rechts'
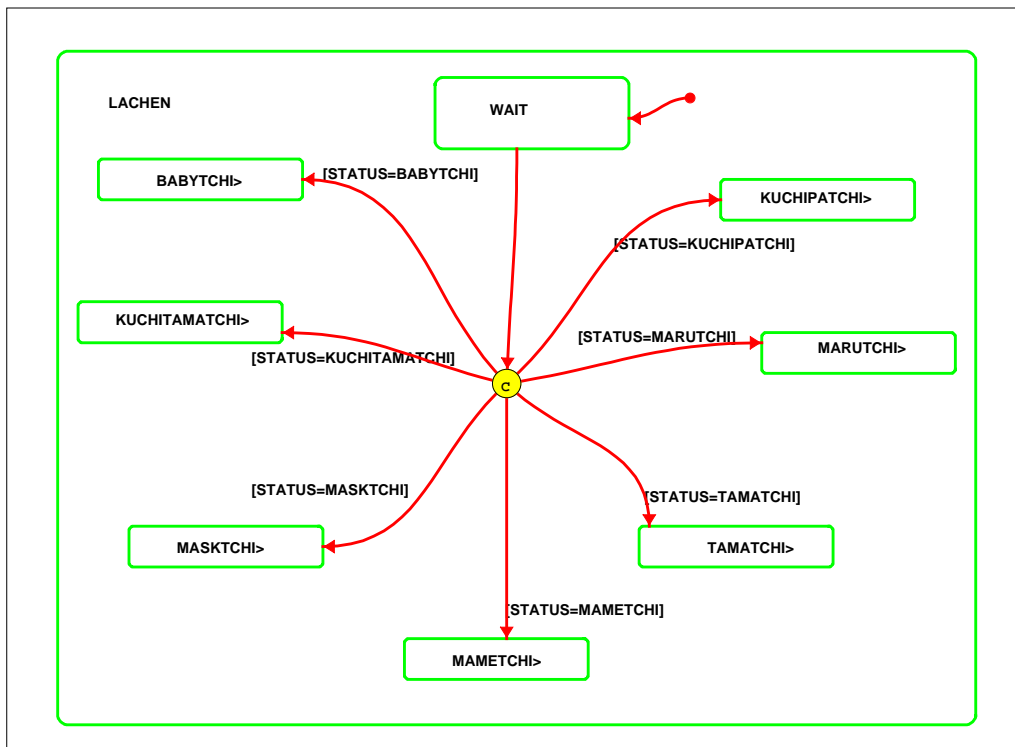 Elements used:
    SPIELEN:LINKS (Event)
    TAMAGOTCHI:TAMAGOTCHIAUSGABE (Variable Data-item) String length=30
    SPIELEN:RECHTS (Event)
Part of BA-F24

States KUCHITAMATCHI, MASKTCHI, MAMETCHI, ... are similar to the state BABYTCHI

Chart: LACHEN   Version:2   Date: 9-FEB-1999 10:56:07

Part of BA-F24

State BABYTCHI
  Static reactions:
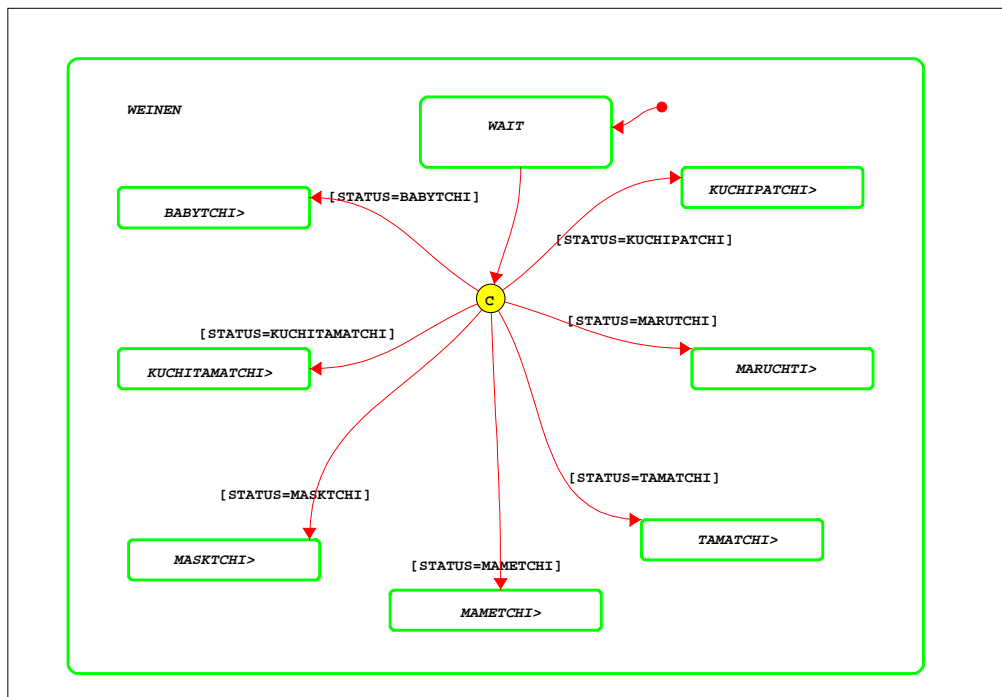 entering/TAMAGOTCHIAUSGABE:='babytchi lacht'
  Elements used:
    TAMAGOTCHI:TAMAGOTCHIAUSGABE (Variable Data-item) String length=30
Part of BA-F24

States KUCHITAMATCHI, MASKTCHI, MAMETCHI, ... are similar to the state BABYTCHI

```
Chart: WEINEN    Version:1    Date:  9-FEB-1999  10:58:01
```

WEINEN

WAIT

KUCHIPATCHI>

BABYTCHI> [STATUS=BABYTCHI]

[STATUS=KUCHIPATCHI]

C

[STATUS=MARUTCHI]

[STATUS=KUCHITAMATCHI]

KUCHITAMATCHI>

MARUCHTI>

[STATUS=TAMATCHI]

[STATUS=MASKTCHI]

TAMATCHI>

MASKTCHI>

[STATUS=MAMETCHI]

MAMETCHI>

Part of BA-F24

State BABYTCHI
  Static reactions:
  entering/TAMAGOTCHIAUSGABE:='babytchi weint'
  Elements used:
    TAMAGOTCHI:TAMAGOTCHIAUSGABE (Variable Data-item) String length=30
Part of BA-F24

States KUCHITAMATCHI, MASKTCHI, MAMETCHI, ... are similar to the state BABYTCHI

1. 4 Experience

To become acquainted with the tool we had to work on the safety injection example. We read Harels paper and started drawing the Statechart of the reactor. This took us about two hours. Later we noticed that the approach of drawing only statecharts was not appropriate for the tamagotchi; we had to deal with several State- and Activity-Charts and the information flow between them. So we had to rework the safety injection example and read the online documentation which took us about 30 hours as a team. The only helpful documentation was an online manual. It was described easy to read and very detailed on how to work with the tool and what the semantics of the different diagramm-types is. The online-documentation comprises several thousand pages. The most important part is: "the statemate approach", which has about 250 pages and contains everything that is important to deal with Module-Charts, Activity-Charts and State-Charts. Working with Statecharts is quite easy as they provide an intuitive model for specifications.

The modelling of the tamagotchi took us about 100 hours as a team. We specified all textual requirements of the tamagotchi. At the beginning we simulated parts of the system and later the system as a whole. Neither the method nor the tool forced us to restrict of the textual requirements of the tamagotchi.The Harel machines are a good modelling concept for the tamagotchi, because the cyber-chicken consists of states, events, variables and timeouts. These concepts could all be modelled with *Statemate Magnum*. There is no difference between the method which is described in Harels paper and the implementation of the method in Statemate. This is not surprising as David Harel is one of the founders of i-logics, the company which developed Statemate.

At the beginning we had some problems defining hierarchies of states. If one starts modelling the statecharts one is used to model the system with flat state-machies, as used from other modelling concepts. This leads to many states and transitions. After this brainstorming session we get a deeper understanding of the system as a whole and one might redesign the system with hierarchies of states. If one gets used to doing so, one can model the hierarchies directly. For a beginner it will take some time to get used to thinking in hierarchical and concurrent states. The usage of the mouse buttons in *Statemate* is also a little bit tricky for beginners, because the buttons are mapped differently according to other systems.

In the following part we will do a short evaluation of *Statemate* and *Octopus*, the method we had to review. The comprehensibility of *Statemate* is better as Statecharts are defined formally. Statecharts are easy to understand and give a very detailed view of the problem. *Octopus* on the other hand is too informal. Many things are described only in plain text. There are also too many diagram types. You have to read about twelve diagrams to cover every aspect of the system. The possible advantage of *Octopus* is that you can decide how detailed you want to describe the system. With statemate you have to describe everything very detailed so you can simulate it.

The traceability of the requirements is provided by *Statemate* with a traceability matrix with which you can describe which requirements are resolved by which states or transitions. In octopus you have to do that manually. If one wants to check the specification for consistency, Statemate provides tool support. There is a consistency-checker which looks for type mismatches, transitions leading into nirvana and checks the usage of variables in different charts. In octopus there are many different views: e.g. use cases, use case sheets, system context diagrams, system diagramms class diagrams operation sheets, sequence diagrams, event lists, event sheets, action tables, and state charts. All of these have to be manually checked for consistency which is almost impossible for large scale systems. We found 18 inconsistencies in the review of the octopus specification. This was only possible because the specification was so small. If it were to have been larger, it would have been impossible to remember all names and check them for consistency.

We found several errors in our own specification with the help of the consistency checker and the simulation of the statecharts. We did not only find the errors/inconsistencies in the textual requirements but also recognized some errors in our design of the tamagotchi. The octopus group which did the review after our simulation did not find any further errors.

1. 5 Conclusions

The tool supports teamwork and Projectmanagement. There is a databank from which each Project member may check out the charts he wants to edit. The other project members have read only access to the checked-out parts of the specification. The teamwork should only be performed for the lower parts of the system. For the toplevel charts it is necessary that all team members are present while designing the subactivities and determining the information flow between the charts. For larger systems it would also be necessary to use the function to document all events and variables used by the system, so that people who work with certain sub-charts get an idea what kind of events there are, and what they do. In a smaller team where the architecture of the system is carried out by all team members this explicit documentation is not necessary. After you have done this; you can parallelize the work very well.

As the tool supports all the features of the method we will restrict our conclusions on the statemate tool. The strengths of the tool are the consistency checking features and the simulator.

The extensions of the finite state machines were very useful for practical purposes. There is a little disadvantage in the hierarchy of states: If one often uses this feature to recurse, one might lose the overview of what will happen, e.g. with nested hierarchies with depth of 5. On the other hand if you make only flat statecharts, there is an explosion of states. You have to decide, how to simplify the system. But in general the hierarchic states help to reduce the complexity of the systems specification and the possibility of concurrent states can help to reduce it, too.

As a weakness one could denote that it takes long to become aquainted with the tool. There is also a possible enhancement, which should be included in the tool: The ability to scale the fonts when zooming in and out with the graphic editor for the charts. The states and transitions are resized, only the size of the fonts remain. This causes some trouble when you have to resize your chart.

However, after having successfully understood the range of functions that can be achieved with *Statemate*, we would definitely use it again.

Bibliography

[Harel 87] Statecharts: A Visual Formalism For Complex Systems
David Harel 1987