Tamagotchi specification with OCTOPUS

Elena Torres, Gerhard Landeck

Seminar Software Engineering
Fachbereich Informatik.
Universität Kaiserslautern
WS 98/99

Table of contents

1 Introduction

This chapter contains an introduction to the components used in this system modelling, the notation, the specification modelling method and the used tool.

1.1 Notation

The OCTOPUS methodology uses many different diagrams, tables and sheets. We will describe the notation of each element at the same time as we explain its function.

1.2 The method: OCTOPUS

A basic principle used in analysis, as in any complex task, is divide and conquer. This means to understand the total problem, partition it into subproblems and then try to understand each subproblem and its relationships. The difficulty here is under which aspects we have to partition. OCTOPUS combines some partitioning aspects, it is a method that makes it possible to develop object-oriented software for embedded real-time systems[AWA '95], like telecommunications and mobile phones. It was developed in the Software Technology Laboratory at Nokia Research Center by Maher Awad, Juha Kuusela, Jurgen Ziegler.[AWA '95].

OCTOPUS is based on James Rumbaugh's 'Object Modeling Technique' (OMT)[RUM '91] and Derek Coleman's 'Fusion' [COL '93]. The object model notation of the OMT allows compact expression of all the necessary details, furthermore the separation of structural, functional and dynamic aspects makes the models easier to build and to understand. Basic separation between the analysis phase, concentrating on describing the external behaviour (behaviour to the customer), and the design phase, concentrating on the internal behaviour of the application, is taken from the Fusion method[AWA '95].

The development of a large system with OCTOPUS is divided into five phases, most of them are built up by three different models. The first model is the structural model, which describes the static structure of the system. The second is called functional model, which shows the behaviour of the system without considering time aspects. The last one is the dynamic model, which considers the time course of the system.

1.2.1 System requirements specification

On this level the system shall be described with its relations to its environment and from the users point of view. First, one must find the use cases which build the functional as well as the dynamic model. Each use case is recorded with the help of a use case sheet, the relations among the use cases are shown in a use case diagram using the notation of OMT class diagrams.

Use cases describe the system from a users view. The system interacts with external agents, that can be humans or computers. These agents can play different roles, each agent associated with a role is called an actor. One can get the use cases by determining the different external actors.

The **system context diagram** is based on the use cases. It shows the structure of the problem and the relations between system and environment . It uses the notation of OMT class diagrams.

A **scenario** is a sequence of events. Use cases may be complemented by scenarios between the system and its environment to show the dynamic behaviour. Scenarios are a possible start point for test cases.

For the authors of OCTOPUS, **verification** at this level means to examinate the use cases exhaustive with the customer .

1.2.2 System architecture phase

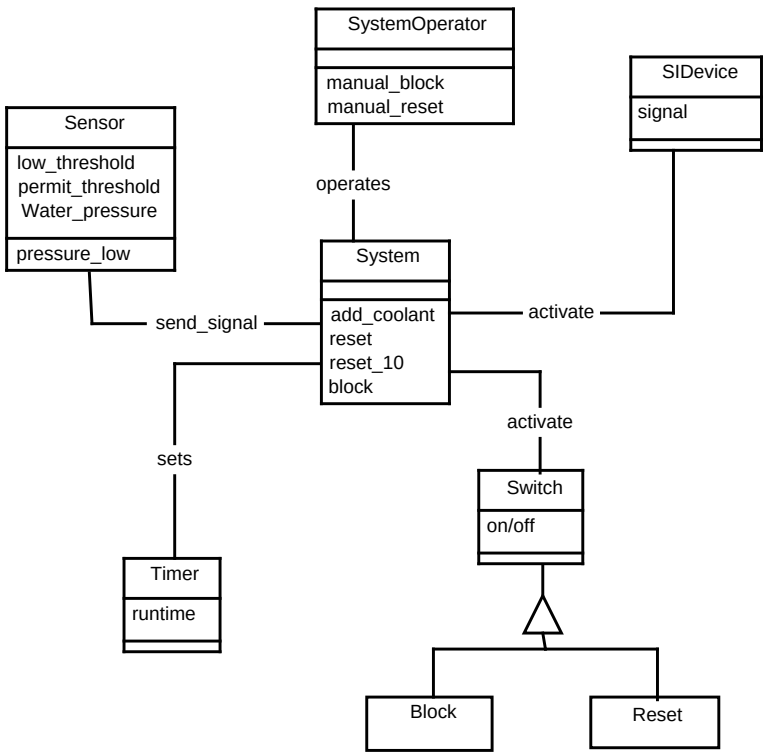
During this phase only the structural model is of importance. The system is divided into subsystems, and the interfaces between them are shown in OMT class diagrams as associations. The hardware wrapper is considered as a special subsystem.

1.2.3 Subsystem analysis phase

In this phase all subsystems will be analysed incrementally and parallel, so that the interfaces between them are clear specified and the system architecture can be verified.

The *structural model* in this phase is strongly oriented on Rumbaugh's OMT method. It is based on **class diagrams** *(Fig.1)*, a **class description table** and **object diagrams** when needed. The class diagram contains classes and its relations. A class represents objects with the same properties (attributes) and the same functional behaviour (methods and states). Classes can have associations among each other, like aggregation, certain cardinality or specialization, each of them possessing own symbols noted at the lines between the class symbols.

Fig.1: Structural model, **class diagram** for Safety Injection Device.



The *functional model* describes the functional interface of the subsystem with the help of **operation sheets.** (*Fig. 2*) These sheets are part of the Fusion method, and define operations declaratively in an informal way.

Fig.2: Functional model, **operation sheet** for Safety Injection Device

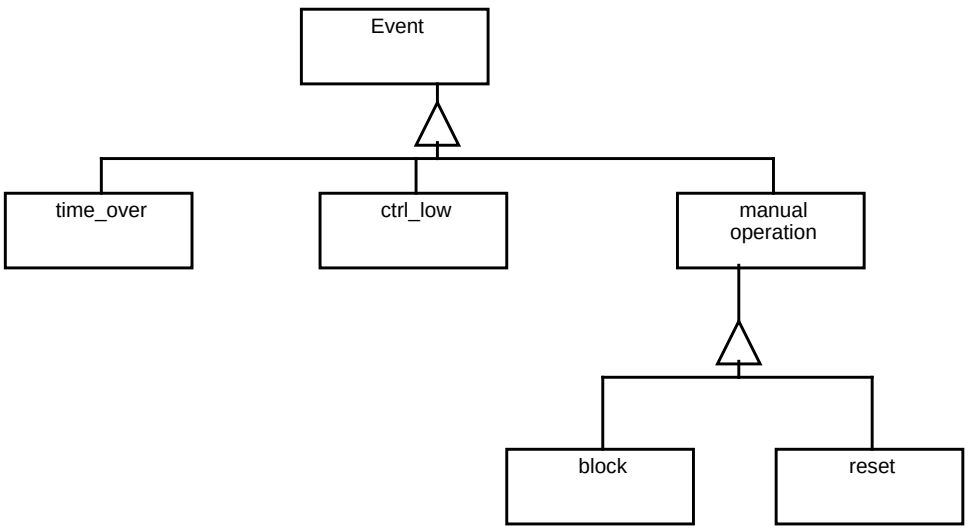
Operation 3	block Safety Injection Device
Description	system operator blocks SID
Association	Class system operator, Class Computer
Preconditions	no safety injection signal present, water pressure < permit
Inputs	water pressure, permit, safety injection signal
Modifies	block switch, timer
Outputs	---
Postconditions	SID blocks, Operation 4 starts

The *dynamic model* shows the operations of the subsystem, but under real time and reactive aspects. It treats conditions for performing the operations, their duration and order, and also which operations are possible in the different states and how operations affect each other.

The dynamic modelling phase is achieved into the following steps:

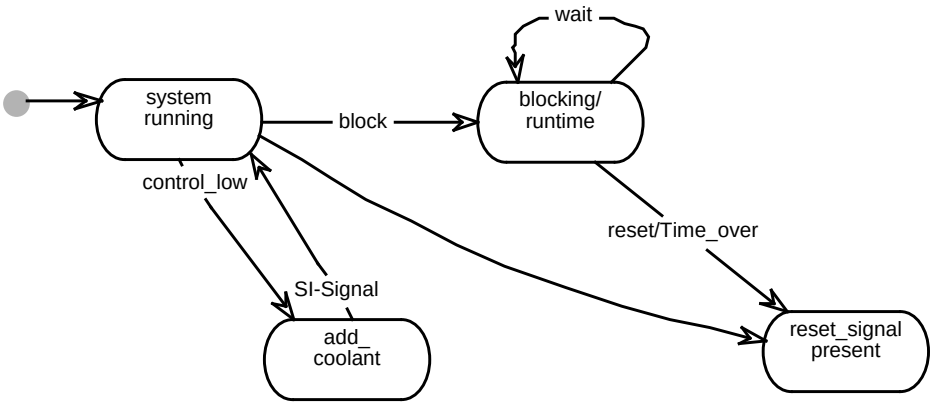
The first step is the *analysis of events*, it consists of building an **event list**, grouping of events in a **event diagram** (*Fig. 3*)and **event sheets**. An event is a request for an operation, it is short and comes from another subsystem or from the system environment. Event diagrams use the class diagram notation. An individual event is described by an event sheet.

Fig.3: Dynamic model, **event diagram** for Safety Injection Device



The next step is the *analysis of states* with concurrent **statecharts** (*Fig. 4*) and an **actions table**. State analysis allows modelling state dependent behaviour. It uses the notation of Harel[HAR '87] . In opposite to OMT, which requires state analysing for each class, OCTOPUS is more flexible, because it allows to analyse states for whole subsystems as well as for parts of subsystems. The information a single statechart provides is limited in order to maintain overview. Actions which occur in a certain state are described in a separate table, the actions table.

Fig.4: Dynamic model, **statechart** for Safety Injection Device



To *further analysis of states and events* belongs a **significance table** (*Fig. 5*), it explains the importance of events according to different states.

Fig.5: Dynamic model, **significance table** for Safety Injection Device

STATE	-------EVENTS-------- control_low	block	reset	timeover
system running	c	c	0	0
add coolant	0	0	0	0
reset signal	0	0	0	0
blocking/runtime	0	0	c	c

Finally, for a better understanding of the dynamic model , **scenarios** will be used. They show the order of events with the help of message sequence charts.

At this point it is possible to work parallel on the design and implementation of each subsystem. OCTOPUS enabels the hardware wrapper to be considered as a subsystem, that permits broader perspective to take important decisions about hardware - software interaction.

1.2.4 Design and implementation phase

Based on the analysis model object interaction threads will be developed and combined or extended to event threads. Details will be recorded in 'class outlines'. Then communication mechanisms between the objects will be established in every event thread. Concurrency will be designed by grouping objects and the outlines of the processes will be developed. These steps will be verified by removing inconsistencies, balancing the design decisions and determining how to synchronize the access to shared objects. In the implementation phase these results will be changed into code. The last two phases, design and implementation, do not belong to client specification modelling.

1.3 The tool

We used the tool StP/OMT (Software through Pictures / Object Modeling Technique), release 2.4.2, which was created by Aonix in from 1984 to 1997. In this chapter we describe some features of this tool.

The name StP/OMT already points out, that the tool has been developed to support the OMT method. Similar to OCTOPUS, OMT uses three views to a system, in fact an object model, a functional model and a dynamic model. The differences between OMT and OCTOPUS models we already pointed out.

So it is not surprising, that it is not possible to build up all parts of the OCTOPUS model with StP/OMT. For instance there is a feature of StP/OMT which allows to make use cases, but the syntax of OCTOPUS use cases is completely different. Therefore we couldn’t use this feature.

All diagrams which use the syntax of OMT class diagrams can be drawn with the StP/OMT class diagram editor, like use case diagrams, class diagrams and event diagrams. But semantic checks make no sense, because they have other semantics.

The state charts of the dynamic model in the analysis phase can also be made with the tool, because in this case there is no difference between OCTOPUS and OMT. Furthermore one can draw up scenarios as message sequence charts with the help of an event trace editor.

The tool has features for cross-checking OMT models, generating C++ code and building an OMT functional model, which is completely different to the OCTOPUS functional model. There is no possibility to simulate the system.

2 Our approach

We did not have a tool which worked on the methodology of OCTOPUS directly. Instead, we used the StP tool that originally supports OMT. For this reason we could get advantage from some diagram editors, but some verification features could not be used. OCTOPUS works strongly with informal description techniques, as sheets in natural language, StP/OMT does not contain any support to manage sheets.

All steps which led to a diagram in OMT class diagram notation could be made with StP, also the statecharts and the MSC diagrams (message sequence charts). We used the appropriate editor of the tool to build state diagrams. All kind of sheets and tables we made with a normal text editor.. The correlation between the different models and phases we had to ensure by means of name consistencies(manual consistency check).

The specification contains:

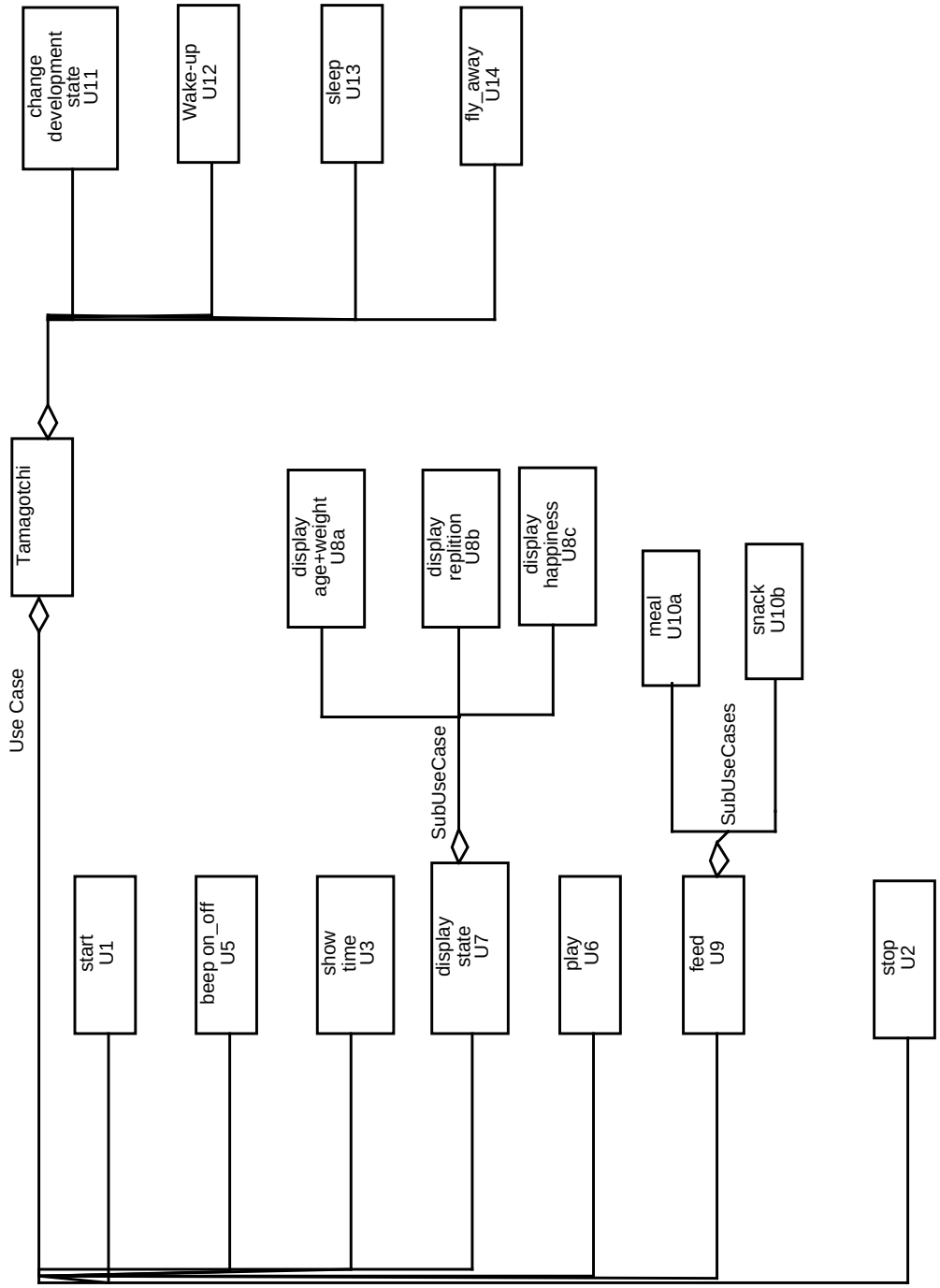
1 System requirements specification	
1.1 Use case diagram	1 diagram
1.2 Use case sheets	1 sheet
1.3 System context diagram	1 diagram
2 System architecture phase	
2.1 Subsystem diagram	1 diagram
3 Subsystem analysis phase	
3.1 Class diagram: object model	1 diagram
3.2 Class description table	1 table
3.3 Operation sheets: functional model	1 sheet
3.4 Events lists: dynamic model	1 list
3.5 Event diagram	1 diagram
3.6 Event sheets	1 sheet
3.7 Actions table	1 table
3.8 Statecharts	4 charts

Total: 9 graphical representations and 4 tables or lists, 20 pages.

3 Specification of the functionality "play"

In this chapter we show that part of our specification, which contains the functionality "play". We start at the use case diagram and the corresponding use case sheet. Then we present the class diagram of the analysis phase, followed by the statechart diagrams, which belong to "play", and the event sheets according to the charts. At last we show the actions table.

3.1 Use case diagram "Tamagotchi"

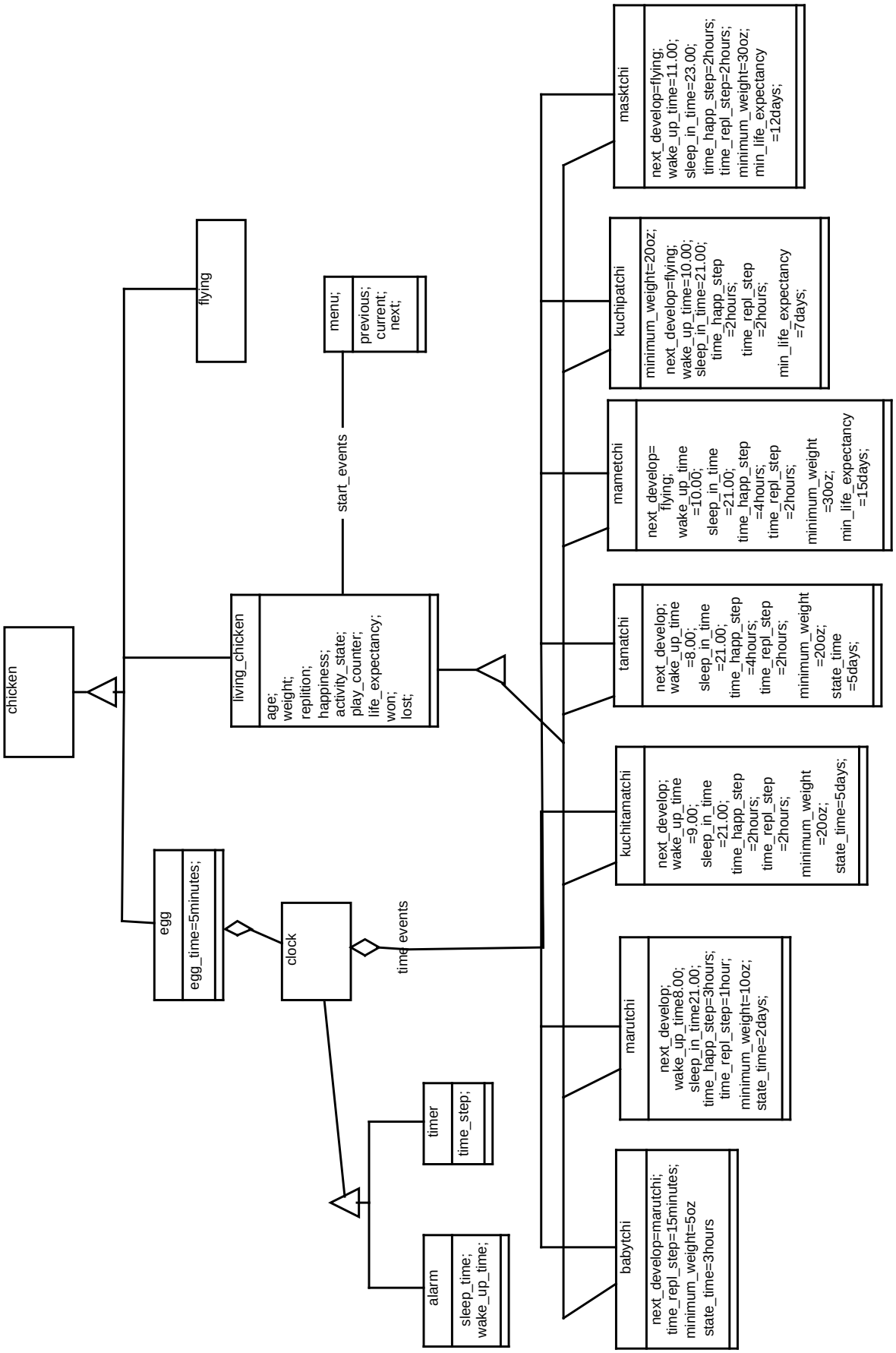


3.2 Use case sheets

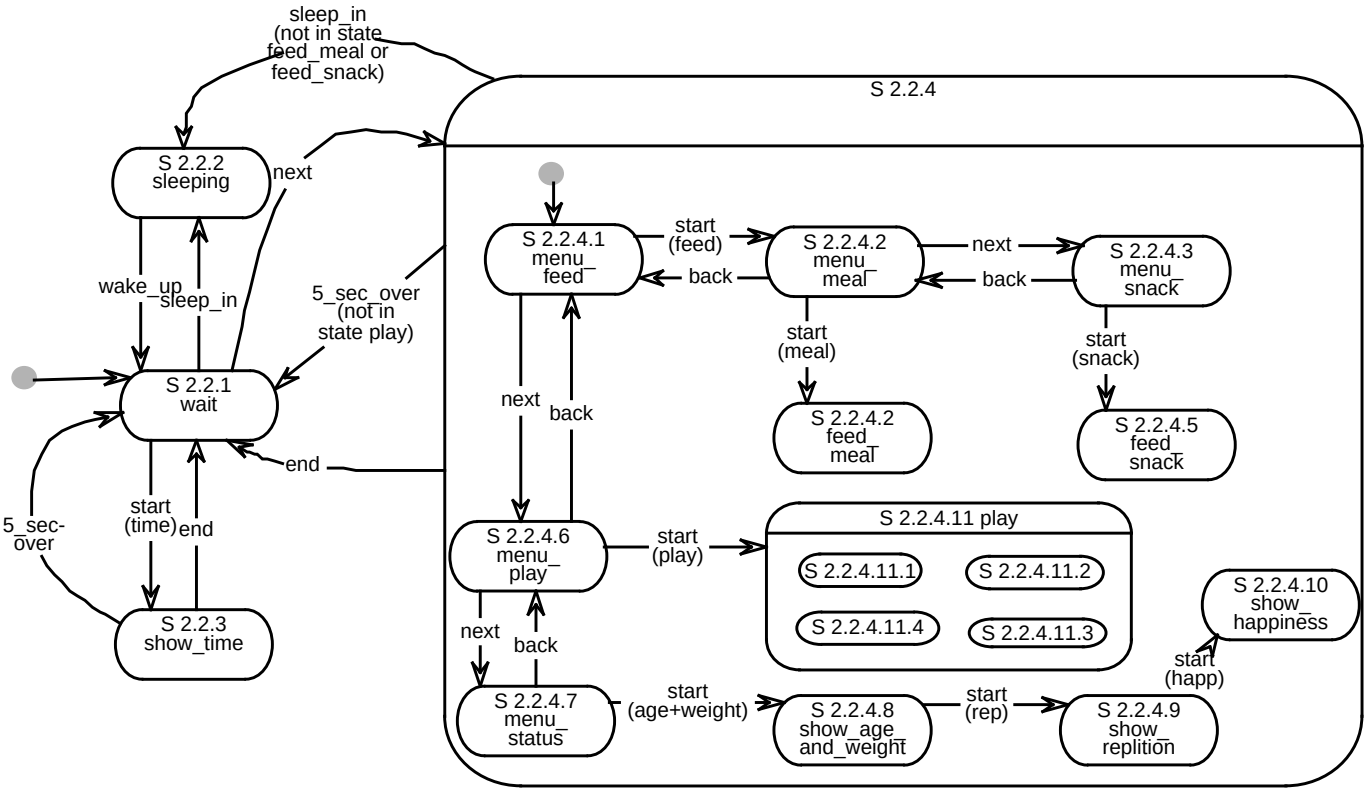
Use Case (U6) **play**

Actor	Benutzer
Preconditions	Das Tamagotchi ist wach und befindet sich nicht im Zustand Ei oder fliegendes T.
Description	Das Display zeigt ein spielendes Küken des aktuellen Entwicklungsstadiums, der Piezosummer piept periodisch. 5 mal wählt der Benutzer mit der linken oder mittleren Taste eine Richtung aus, das Küken ebenso per Zufallsgenerator. Stimmt die Richtung überein, zeigt das Display ein lachendes, sonst ein weinendes Küken. Nach 5 Runden wird der Spielstand angezeigt.
Sub Use Cases	---
Exceptions	end (rechte Taste), stop (Streifen rein), sleep_in (Einschlafzeitpunkt erreicht);
Activities	---
Postconditions	Das Display zeigt ein Küken des aktuellen Entwicklungsstadiums.

3.3 Object model of subsystem tamagotchi



3.4 Statechart activity



3.5 Extract of event sheets

Event E3 next

```
Response      Display zeigt nächsten Menüpunkt
Associations  siehe menu
Source        siehe menu
Contents      ---
Response Time siehe user event
Rate          siehe user event
```

Event E4 start

Response	Beginn einer Aktion
Associations	siehe menu
Source	siehe menu
Contents	Name der Aktion
Response Time	siehe user event
Rate	siehe user event

Event E6 end

Response	Display zeigt Küken
Associations	siehe menu
Source	siehe menu
Contents	---
Response Time	siehe user event
Rate	siehe user event

Event E7 direction

Response	chicken moving eyes, laughinf or crying chicken
Associations	current instance of living_chicken
Source	siehe user event
Contents	right or left
Response Time	siehe user event
Rate	siehe user event

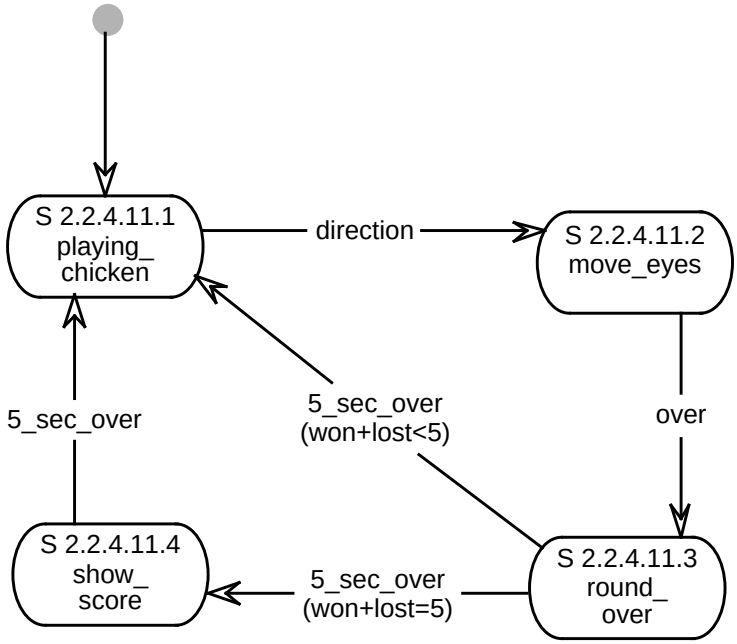
Event E14 5_sec_over

Response	Display zeigt vorhergehenden Menüpunkt oder Küken;
Associations	Timer
Source	Display hat Anzeige geändert;
Contents	---
Response Time	siehe time event
Rate	unregelmäßig.

Event E15 over

Response	T. wechselt in Zustand round_over;
Associations	Aktuelle Unterklasse von living_chicken, display
Source	Küken hat die Augen bewegt
Contents	---
Response Time	siehe time event
Rate	immer nach dem Ereignis direction.

3.6 Statechart play



3.7 Extract of actions table

Elementary state	**Event**	**Actions/activities**
S 2.2.1 wait	<enter> <wake_up>	Display zeigt waches Küken entsprechender Entwickl. age := age+1; play_counter := 0; Gewicht wird um entwicklungsabhängigen Wert verringert;
S 2.2.4.1 menu_feed	<enter>	Display zeigt "Nahrung und Süßigkeiten"
S 2.2.4.6 menu_play	<enter>	Display zeigt "Spielen"
S 2.2.4.11.1 playing_ chicken	<enter>	Display zeigt spielendes Küken des aktuellen Entwicklungsstadiums;
S 2.2.4.11.2 move_ eyes	<enter>	T. wählt Richtung, bewegt Augen auf Display, vergleicht, ändert won oder lost;
S 2.2.4.11.3 round_ over	<enter>	Display zeigt lachendes oder weinendes Küken, T. zählt won und lost zusammen;
S 2.2.4.11.4 show_ score	<enter> <exit>	Display zeigt Spielstand an; play_counter := play_counter + 1;

4 Experiences

In this chapter we talk about our own experiences with OCTOPUS and the StP tool. This contains how much time we needed for the work and which difficulties we had to surmount. Another point is the comparison with the group that used the statechart method.

4.1 Preparation for modelling

We had to work eight hours per person and week during the training period, which lasted for four weeks. There was to read the only existing book about OCTOPUS, to understand the complex method and to get used to work with the tool.

One bad point of the book is the order of building the different models during the subsystem analysis phase. There is first described the static model, then the functional and last the dynamic one. It is better to build the dynamic model before the functional model or, if possible, build them parallel. If one begins with the dynamic model, it is easier to identify operations, events and also the attributes which will be needed in the static model.

Another problem for beginners is to distinguish events from operations. Because we developed the functional model before the dynamic one we identified some operations which had more the character of events. Dividing the system into subsystems and classes is a difficulty which appears to people who have no experience in object modelling. This means in our own case, that we built too large classes. So later we had to split it into several classes. A special problem of OCTOPUS is how deep to go with the use cases. There is no limit given in the book telling you when to stop with building sub use cases or activities.

4.2 Modelling

Modelling the tamagotchi took also eight hours per week and person during six weeks. It contents all software requirements, but we didn't analyse the hardware wrapper. The notation of OCTOPUS allows to describe all requirements because you model on a very high level and you have static, functional and dynamic structures. This makes it possible to describe the time and reactive aspects.

With the tool it is possible to draw all diagrams which use the notation of class diagrams.Statecharts and scenarios (as message seqence charts) can also be drawn. It is not very useful to check syntax or semantic of use case and event diagrams although the tool is able to do so. The reason is that these diagrams use the notation of class diagrams, but have a completely different semantic. In the analysis phase informal text is used. The informal sheets and tables could not be done with the tool, because there is no editor to write informal text. This was very bad because we could not develop the whole model with one tool.

In our opinion it is not easy to get an overview about the model for beginners of the method. The numerous tables, diagrams and sheets make it difficult. It is also not easy to follow a line of user actions like "what happens if you press the middle button". It seems best to begin with the state charts and follow the line through the actions table, event sheets and operation sheets. The depiction between requirements and diagrams is not 1:1, but 1:n.

In the analysis phase frequently informal text is used. Therefore it is not difficult to understand the specification if you know the way through the diagrams.

4.3 Review

The method we reviewed was the statecharts method. The members of that group showed us their simulation on the machine. They gave us their written specification. We examined it in a rough way. That means we looked only at the highest levels and some certain requirements. Because there is only one kind of diagrams it was easy to understand. We had no problems to follow a line of user actions. We couldn't find any mistake in the specification.

The other group reviewed our work by reading our written specification. They didn't find any mistakes.

5 Conclusions

Working parallel was not possible for us because we had only one subsystem. With a larger system which would be divided into more subsystems we think it would be easy to work in parallel.

In OCTOPUS there is no problem with modelling any part you want. Real time and reactive aspects can be shown very good. Negative is that you don't know how deep you shall go in detail. The number of diagrams, tables and sheets make an overview difficult. For building the sheets and tables you must use an external editor. It would be better to make them with the same tool as the diagrams and state charts. It would also be good if there were possibilities to check the consistencies between the different models and if it could be simulated.

Probably it would have been enough to make a rough design and programme it then. This would have been much faster to get finished. However, if one has experience with OCTOPUS he would have been much faster than we were.

To learn the method it was useful to make a model of the tamagotchi. The safety injection device was much too small for learning the method. The tool didn't help us very much because we used it in the way of a graphics editor. Some things we wanted to do probably would have been easier done with a good graphics editor.

We think the most disadvantages we mentioned here will be eliminated with growing experience. The exception is the support of the tool.

6 Reference

[AWA ’95] Maher Awad, Juha Kuusela, Jurgen Ziegler, *Object-Oriented Technology for Real-Time Systems*, Prentice Hall, 1995.

[COL ’93] Derek Coleman, Patric Arnold, Stephanie Bodoff, Chris Dollin, Helena Gilchrist, Fiona Hayes and Paul Jeremaes, *Object-Oriented Development - The Fusion Method*, Englewood Cliffs, NJ: Prentice Hall, 1993.

[RUM ’91] James Rumbaugh, Michael Blaha, William Premerlani, Frederick Eddy and William Lorensen, *Object-Oriented Modeling and Design*, Englewood Cliffs, NJ: Prentice Hall, 1991.