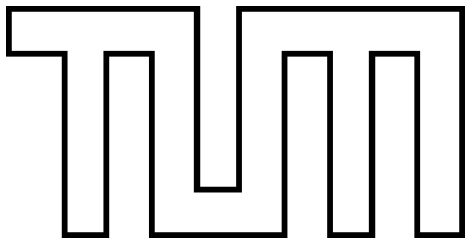
Technische Universität München
Fakultät für Informatik



Lehrstuhl für Programmiermethodik und Verteilte Systeme
Univ.-Prof. Dr. M. Broy

Hauptseminar

Werkzeuggestützte Modellierung des Tamagotchi mit Hilfe des PEP-Tools

Claudia Beetz
Christian Harboeck
Michael Karas

Betreuer: Bernhard Schätz

Index

1 Method and PEP-Tool

1.1 Introduction

The PEP tool (**P**rogramming **E**nvironment based on **P**etri Nets) is a programming environment for modeling, compilation, simulation and verification of parallel systems based on Petri nets. The basic idea is that the modeling component allows the user to design parallel systems by using parallel finite automata, parallel programs, process algebra terms, high-level or low-level Petri nets. Of this models the PEP tool automatically generates Petri nets for simulation and verifications purposes.

1.1.1 History of Development of the PEP Tool

The PEP system is being developed since 1993 as a part of a common research project of the Universities Hildesheim and Oldenburg sponsored by the DFG (German Research Foundation). The project was initiated by Hans Fleischhack and Eike Best who's theoretical work along with the results of other EU projects like DEMON (Design Methods based on Nets) and CALIBAN (Casual Calculi based on Nets) was the basis for this project. Javier Esparza, from the Technical University of Munic, developed the theory for the central model checker algorithm which tests the Petri nets generated by the PEP system for security of the net. Bernd Grahlmann who led the Hildesheim group and hold overall the responsibility of the project, designed and took care of the implemtation of the PEP tool. This was carried out in cooperation with the Oldenburg group, mainly interacting with Matthias Damm, Marc Langnickel, Andree Seidel and the STONE group of students.

The following persons made contributions to the implementation:

Burkhard Bieber	Implemented the main part of the graphical user interface
Stefan Römer	Took care of all kinds of algorithms and optimized them
Burkhard Graves	Implemented some of the compilers, and the model checking algorithm for safe Petri nets
Tobias Himstedt	Implemented two Petri net editors and simulators
Lars Jenner	Developed and implemented the low-level procedure semantics
Martin Ackermann	Implemented the compiler from a $B(PN)^2$ program to a M-net
Michael Kater	Implemented different text editors and contributed to the Linux port
Stephan Melzer	Developed and implemented the C code generator
Thomas Thielke	Implemented a dedicated model checking algorithm for T-systems
Stephan Merkel	Improved the functionality of some of the compilers
Carola Pohl	Implemented the on-line help system

Further contributions were made by Farzad Eghtessadi, Johannes Jäger, Eberhard Meisel, Robert Riemann, Hartmut Trüe and Fatma Uzel. The PEP project is being continued in cooperation with the Humboldt-University of Berlin.

1.1.2 System overview of the PEP-Tool

As mentioned before the PEP tool is a programming environment based on Petri nets. The PEP system contains editors and simulators for modeling and simulating of parallel systems. Furthermore there are compilers which generate Petri nets of this systems. The PEP tool not only combines two of the worldwide acknowledged theories Petri nets and process algebra, it also offers the possibility to apply them to the integrated programming language B(PN)^2 (**B**asic **P**etri **N**et **P**rogramming **N**otation). The main part of the PEP system is a comprehensive verification component which permits checking important properties of the parallel systems being modeled before. Furthermore the whole functionality of the PEP tool is embedded in an user-friendly graphical user-interface. But the documentation of the PEP system is not very helpful using the tool.

For modeling parallel systems with Petri nets using the M-net editor of the PEP tool the user has not to take care of any sequence of the proceeding. The system offers the possibility to connect several single nets as sequential or parallel or combine them to one large net. Small partitions of Petri nets still can be simulated during modeling with the graphical programming environment of the M-net editor. But this induces the user to neglect conceptual considerations.

The following different classes of components are embedded in the completely graphical user interface of the project window (shown in picture 1) of the PEP tool. In this window all available editors and compilers can be started.

1. **Editors:**
   PFA (**P**arallel **F**inite **A**utomata)
   B(PN)^2 programs
   PBC terms (**P**etri **B**ox **C**alculus)
   Formulae
   High-Level nets
   Low-Level nets
2. **Simulators:**
   PFA
   B(PN)^2 programs
   HL nets
   LL nets
3. **Compilers**:
   PFA $\Rightarrow$ B(PN)^2
   B(PN)^2 $\Rightarrow$ PBC, Exp-B(PN)^2, C Source
   Exp- B(PN)^2 $\Rightarrow$ HL net
   PBC $\Rightarrow$ LL net, HL net
   HL net $\Rightarrow$ LL net
   LL net $\Rightarrow$ Prefix, FC2, INA, SMV, SPIN
4. **A verification component**
5. **A reference component,** e. g., transforms program formulae into net formulae or triggers program simulation by net simulation

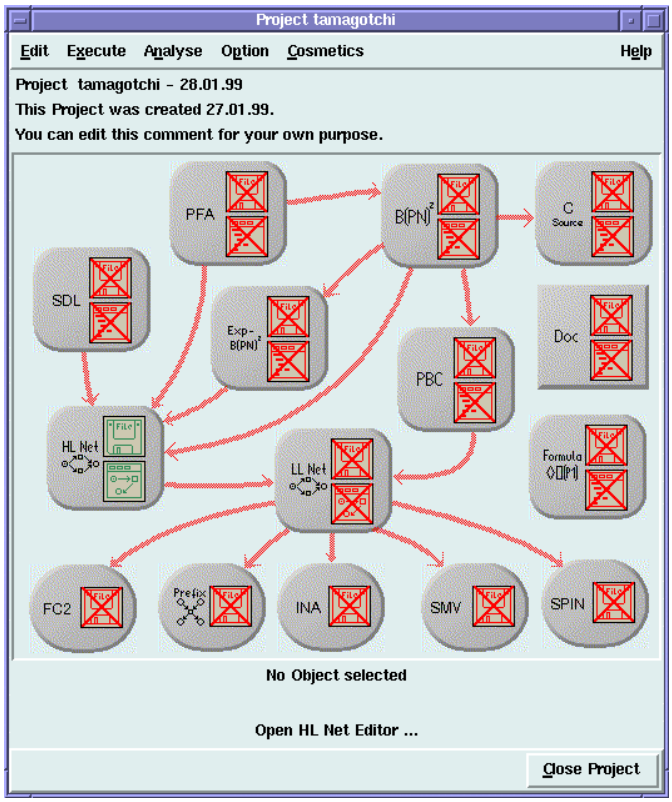


Figure 1

1.1.2.1 Objects of the PEP-Tool for modeling parallel systems

The PEP system gives the user a choice between five different kinds of objects for modeling parallel systems, which are briefly described in the following part of this section:

1. **P**arallel **F**inite **A**utomata (PFA) with B(PN)^2 actions as edge annotations can be edited and compiled into B(PN)^2 programs.
2. **B**asic **P**etri **N**et **P**rogramming **N**otation B(PN)^2 for implementing parallel algorithms, which is an imperative/predicative programming language.
3. **P**etri **B**ox **C**alculus (PBC), which includes terms of a process algebra, can be derived automatically from a B(PN)^2 program or be designed independently.
4. M-Nets, which allows to design and edit HL programming notation. The HL nets can be created from B(PN)^2 programs or can be designed independently.
5. P/T-Nets, which allows to design and edit LL programming notation. Petri boxes are a special case of LL programming notation, which may arise out of a translation from B(PN)^2 programs.

The use of the M-net editor will be described in detail in the next section because the tamagotchi was designed and also simulated with the high-level net editor. The reason of

choosing this graphical programming interface of the PEP tool is, that with this editor it is easy to design Petri nets and the simulation can be started every time during the modeling.

A more exactly description of the other objects of the tool is not possible because in none of the documentations these components are described more precisely.

1.1.2.2 High Level Nets and the M-Net Editor

The M-Net editor, shown in figure 2, has a graphical user interface for modeling and also simulating high-level nets. These nets consist, as usual, of places (shown as cycles), transitions (squares) and arcs, which all have a unique name. The names of places and transitions can be changed by the user, as long as these given names are still unique. But for arcs, this name is internal and can not be changed. All of these elements of high-level nets can be positioned easily on the graphical surface of the M-net editor. For this, the three buttons with the symbols of a cycle, a square and an arc, located underneath the menu bar of the window, can be used. Furthermore the Petri net elements can also be created by pressing the p-key (**p**lace), the t-key (**t**ransition) or the **a**-key (**a**rc).

In the picture below, a timer which counts minutes, hours and, at least, days is realized as a high-level Petri net in the M-net editor.

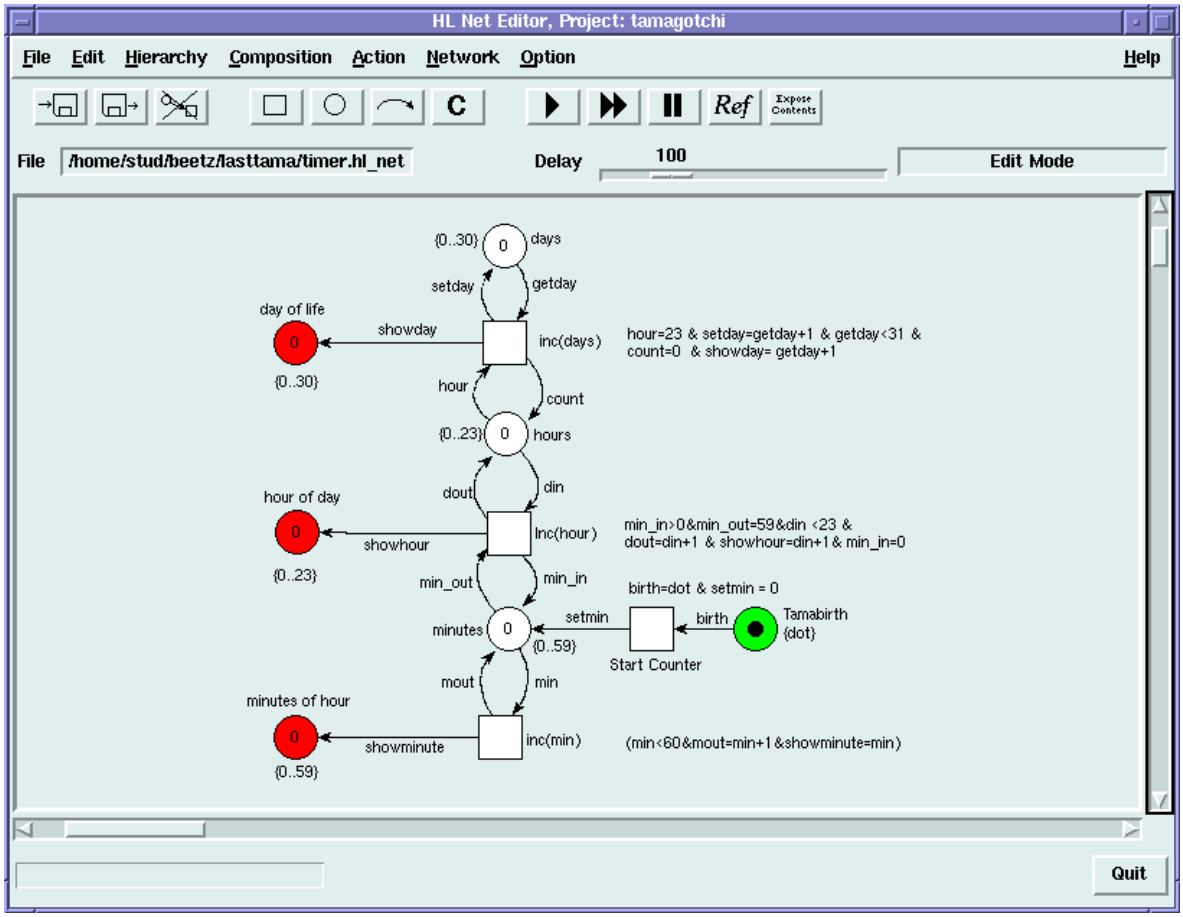


Figure 2

For the simulation there are three buttons (>>, >, ▮▮) underneath the menu bar which can be used. The user has the possibility to decide whether to run the simulation in a single step modus or to run the simulation completely from the beginning to the end. With the 'pause'-button the simulation can be stopped every time, e. g. to change the current markings of several places. During the simulation, operating transitions appear in a yellow color, active arcs appear green and the current markings of the places are shown in the symbols of the places.

Dialog windows of places, transitions and arcs permit the user to make some inscriptions, for example to test conditions or to allocate values. Modeling the high-level Petri nets with the M-net editor this is the only possibility to test conditions or to change values of variables. These dialog windows can be opened by mouse click on the corresponding element of Petri net symbol. The possibilities of inscriptions in the dialog windows and the results to the functionality and also the simulation of the Petri nets will briefly be described in the following section. Inscriptions which are not important for modeling Petri nets with the M-net editor are not described more exactly.

1.1.2.2.1 Place Dialog Window

Places may carry a name, a meaning, a type, a marking and a status. The name is created by the tool and can be changed by the user. If a net is compiled out of a $B(PN)^2$ program the meaning gives a reference to the program it arises from. A place corresponding to a variable in the $B(PN)^2$ program e. g. is labeled with the name of the variable.

But more important for the creation of Petri nets in the M-net editor is the type of a place. Types can be given as, e. g., {dot}, {tt, ff} or {1..10}, with tt representing true and ff false. Furthermore cartesian products are allowed, e. g. {1..3} X {dot, 1..2}.

A marking must be a multi-set over type. E. g., 3*2, 3 specifies a marking 2 2 2 3. 2-tuples can be specified, e. g. (dot, dot), (1, 2), (3, 1). The 'initial marking' shows the marking at the start of the simulation, and 'current marking' shows the marking at this time when the simulation has been stopped. This gives the possibility to change the current markings of the places during the simulation.

The status of a place can be one of three kinds: entry, internal or exit. Entry and exit places have the default type {dot}, internal places may have other types. By modeling Petri nets with the M-net editor, this will give the possibility to connect two existing nets as sequence. Two separate nets can be combined by gluing the exit places of the first net to the entry places of the second.

1.1.2.2.2 Transition Dialog Window

The transitions may also carry a name and a meaning. If the net is compiled out of a program the meaning is a reference to a $B(PN)^2$ program. But for designing Petri nets with the M-net editor the meaning is not important because it is only a comment an can be changed by the user.

Furthermore transitions carry value terms which are a conjunction of terms referring to the variables used in arc annotations. They include condition terms which are Boolean expressions and must be true for the transition to operate, and terms which can change the values of the variables in the arc annotations. For example, referring to the timer in figure 2, the transition 'inc(min)' with the value term 'min<60 & mout=min+1 & showminute=min' will operate as long as the value of 'min' (current marking of the place 'minutes') will be smaller than 60. During each operation of the transition the value of 'mout' will be increased and the value of 'showminute' will be the same as the value of 'min' (current marking of the place 'minutes').

1.1.2.2.3 Arc Dialog Window

The Arc dialog window may carry only one inscription: multisets of variables which can be used in value terms of neighbored transitions. Only this variables may be used in the value terms of the neighbored transitions.

1.2 Approach

1.2.1 Modeling of the tamagotchi

Modeling the tamagotchi with the PEP tool using the M-net editor there is no special sequence in proceeding which has to be observed. Designing Petri nets with the M-net editor the user also has not to take care about any levels of hierarchy of the system because the high-level net editor of PEP uses no hierarchical structure for creating Petri nets.

First of all, the functional requirements to the tamagotchi have been inspected. By doing this it was important to consider how this requirements can be divided into independent acting functional units, to enable a concrete distribution between the members of the team. For this the approach of the PEP tool always had to be in the foreground.

During this considerations, first of all, the functionality of the menu control was inspected to find a possibility for the realization of this requirement with the support of Petri nets. Furthermore a possibility had to be found for modeling that part of the menu control with Petri nets which represents the user interface of the tamagotchi – the 'L'-, 'M'- and 'R'-button. Later, this buttons have been realized as 'places' in the Petri net. During the simulation the 'place' of the button which is being pressed can be marked with a 'dot' so that the neighbored 'transitions' are able to occur.

Furthermore the options oft the menu were inspected, that means it was recorded which menu only displays information, e. g. the weight or the age, which menu has sub menus and with which menu an action can by started (e. g. playing with the tamagotchi, feeding the tamagotchi).

In addition to this it had to be fitted which abilities the tamagotchi should have, that means what additional components have to be modeled as separate Petri nets. This components are: playing, feeding and sleeping. But also the in- and decrease of the weight and the happiness had to be taken in consideration. Furthermore a timer was necessary which is used from some

other components of the tamagotchi depending on the time (e. g. sleeping time). At the end a buzzer and a display should realize an acoustical respectively an optical output.

To get an overview over the phases of evolution of the tamagotchi, irrespective of the approach, a diagram has been designed representing the phases of development containing the special differences between the separate stages (shown in *picture 3*). Some of this special differences are, for instance the sleeping time, the minimum weight, the loss of weight per each night or the growing of appetite during the day. Furthermore in this picture (*picture 3*) the conditions are shown which are decisive for the following stage of evolution. An example for such a condition is the 'playing condition' which represents the number of games per day.

Later in the model of the tamagotchi, an INTEGER variable represents the corresponding stage of evolution as numerical code. In the Petri net this variable is realized as 'place' which is able to contain markings as this numerical values of this code. So this markings can be read out an used as condition for neighbored 'transitions' to occur. As this variable, named "stage of evolution", all other variables in the high-level Petri net were realized as places. Only in this way it is possible to read out values and to use them as conditions. But in this way realized variables only can be of type INTEGER or BOOL respectively 'dot', and that is the reason why the stages of evolution are represented as a numerical code. Also all conditions which have to be proved at the time of changing the stage of evolution are realized as markings of 'places' to make it possible to decide which stage of development of the tamagotchi is the nest. For example, whether the value of the happiness never felt short of three or whether the number of games was not lower than two, are conditions which belong to such variables described in the upper part.

1.2.2 Distribution of Requirements and Definition of Interfaces

Because the approach of the PEP tool is based on Petri nets it is easy to distribute the functional requirements to the tamagotchi. On principle very small Petri nets can be designed consisting of only two 'places' and one 'transition'. The status of one 'place' can be 'entry', 'internal' or 'exit'. Because of this there is the possibility to combine some single nets by connecting the 'entry-' to the 'exit-places'. The result status of such 'places' will be 'internal'. By doing this using the M-net editor it is possible to design all single nets in one file or to import the nets, which should be connected, from another file. Furthermore the M-net editor of the PEP system gives the possibility to compact the whole net or only parts of Petri nets into 'boxes' by loosing the graphical representation of the connecting 'arcs'. Perhaps this would be favorable for the clearness of the representation of the net but for the understanding of the whole net and for also for the simulation it is not very advantageous. This so-called 'boxes' can be denoted with a name but this name is also not displayed on the screen. This given name only appears in the property window of the 'box'. Through that in the graphical representation it is not possible for the user to recognize which box is containing which net without opening the 'box' dialog or the 'box' itself.

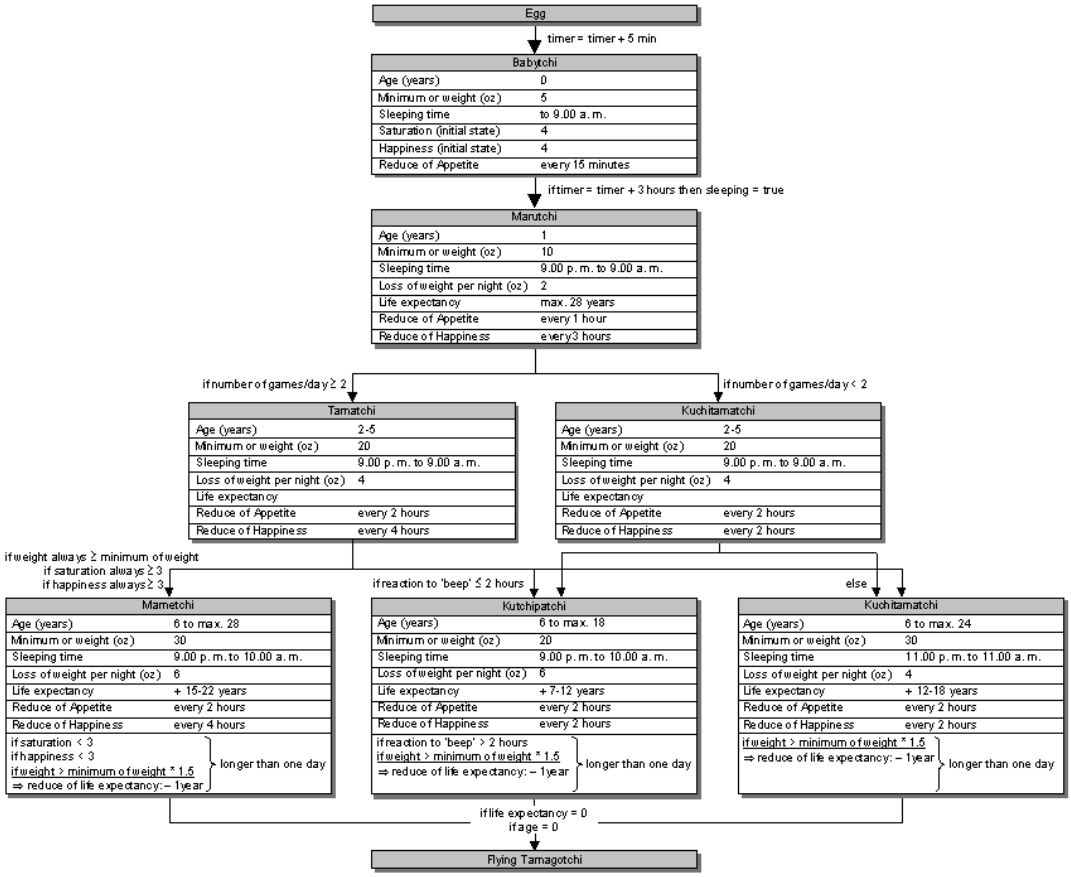


Figure 3

To avoid later appearing difficulties by connecting the single components of the Petri nets, during the distribution of the functional units into single the interfaces between the nets have clearly to be defined. To get an efficient distribution of the functional requirements to the tamagotchi the single nets, as listed as follows, were chosen. The most important functions and interfaces between the net components are briefly described in the following part of this chapter:

- 'Menu-handler' including 'L'-, 'M'- and 'R'-button
- 'Display'
- 'Timer'
- 'Stages of evolution'
- 'Game'
- 'Feeding'
- 'Sleeper'
- 'Variable-handler'

A schematic overview of the modeled net components (shown as boxes in *figure 4*) and their interfaces between the single nets.

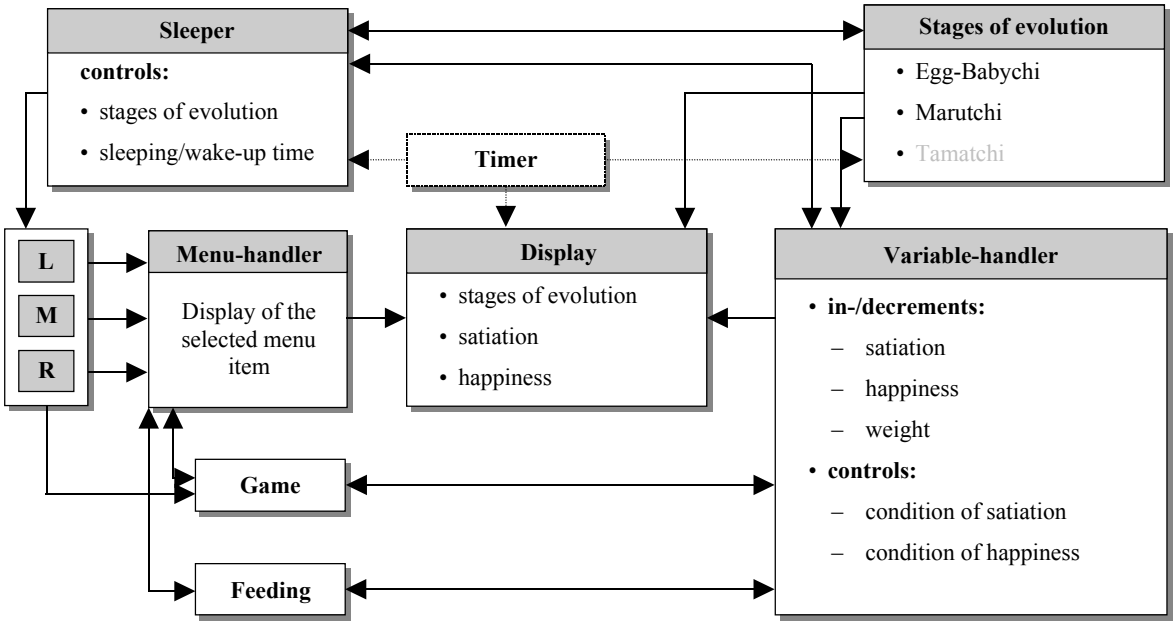


Figure 4

1.2.2.1 The Menu-handler

Fist of all the modeling was started with the component of the 'Menu-handler' which also contains the 'L'-, 'M'- and 'R'-button. With this buttons, as mentioned before, which are realized in the Petri net as 'places', the menu items can be selected and actions like the game can be started. The selected menu item is displayed in a separate 'place' in the 'Menu-handler'. For displaying this chosen menu item in a 'place' the items of the menu are represented as a numerical code. Similar as the code of the stages of evolution. Selecting the menu items satiation, happiness or weight the values of this variables also will being displayed in a separate 'place'.

There are interfaces between the 'Menu-handler' an the 'Game' and between the 'Menu-handler' and the 'Feeding' of the tamagotchi. The interfaces are realized as 'places' having the type 'dot'. Activating the 'M'-button the corresponding 'place' is marked with a 'dot' and the 'Game' net respectively the 'Feeding' net is activated. In the opposite direction the 'place' as interface between the 'Game' net and the 'Menu-handler' gets a 'dot' by pressing the 'R'-button and the 'Menu-handler' will be activated again. After feeding the tamagotchi the menu item 'Food and Snack' is automatically activated again.

1.2.2.2 The 'Variable-handler'

The 'Variable-handler' has been modeled as a separately acting net allocating values to the variables, controlling this values and also make this values available for the other nets for checking the conditions which have to be fulfilled for starting different actions. For this, the variables, realized as 'places', represent the interfaces between the 'Variable-handler' and the other net components so that the required values could be read out or could be changed by the other nets.

For example the 'satiation', the 'happiness' and the 'weight' belong to these variables which are in- or decremented by playing with the tamagotchi or feeding the tamagotchi. In addition to this, after every game and every feeding the values of the variables 'satiation' and 'happiness' will be checked to their maximum value. This means, amounts this value to four, the corresponding variable will not be increased.

The condition of the satiation and the happiness can be read out through the interface between this net and the net of the 'Sleeper'.

1.2.2.3 The 'Display' and the 'Timer'

The 'Display' is not realized as separate acting Petri net. The values of the 'satiation', the 'happiness' and the 'weight' were displayed during the simulation directly as markings in the corresponding 'places'. This 'places' are represented, as described in the upper part of this text, as variable-interfaces between the nets. The stage of evolution of the tamagotchi can also be read out, as a numerical code, of the variable-interface between the nets 'Stages of evolution' and 'Sleeper'.

The 'Timer' is represented as a net which consists of three 'places which contain the values of days, hours and minutes. These 'places' are also used as variable-interfaces between the nets 'Timer' and 'Stages of evolution' and between the nets 'Timer' and 'Sleeper'.

1.2.2.4 'Stages of evolution'

With the net 'Stages of evolution' only the transition from the stage 'Egg' into the stage 'Babytchi' and the change from stage 'Babytchi' to stage 'Marutchi' can be simulated. To start the simulation of changing the stage of evolution the values of the variable-interfaces of the 'Timer' will be read out. Fulfil this values of the variables 'days', 'hours' and 'minutes' the condition for changing the stage of evolution the simulation of the Petri net 'Stages of evolution is being started. Through the interface between this net and the net of the 'Variable-handler' the values of the variables 'satiation', 'happiness' and 'weight' will be changed correspondingly.

1.2.2.5 The 'Sleeper'

The function of the Petri net 'Sleeper' is on the one hand to block the 'L'-, 'M'- and 'R'-button of the tamagotchi so that they are out of order. By doing this the 'place' (type {0..1}) as interface between this net and the net 'Menu-handler' is set to '1' with the result of the deactivation of the buttons. On the other hand the 'Sleeper' controls the stages of evolution and the corresponding sleeping and wake-up times. Through a variable-interface to the net component 'Stages of evolution' the value of the current stage of evolution is being read out and the sleeping and the wake-up time will be determined. Furthermore the value of this variable gives the decision for the value of the decrease of weight during the night. Again this value can be read out as an INTEGER from the net 'Variable-handler' which decrements the weight after the night by this value.

In addition to this the 'Sleeper' also decides whether the tamagotchi changes its stage of evolution during the night and determines which stage of development will be the next one. For this the values of the variable-interfaces between this net and the net 'Variable-handler' will be checked. The condition-variables of the happiness, of the satiation and of the number of games per day belong to this kind of variables which will be read out to decide which stage of evolution will be the next.

In the case of corresponding values of the 'Timer' and the wake-up time of the tamagotchi the value of the 'place' between the net of the 'Sleeper' and the net of the 'Menu-handler' will be set to '0' and the buttons will be activated again.

1.2.3 Extensiveness and Grade of Details of the Specification

The specification contains the net components described in the previous section. Designing such a comprehensive model like a tamagotchi the approach of the PEP tool using Petri nets is not very suitable. Because of this and also having some problems with the PEP tool itself not all functional requirements to the tamagotchi were completely designed. Which parts of the functional requirements have been modeled and which simplifications were made will be described in the following part of this chapter relative to the list of requirements "functional B-requirements". For this the structure of the list of requirements has been adopted. Starting the simulation while modeling was possible at every time.

1.2.3.1 The User Interface of the Tamagotchi

The functions of the user interface have completely been realized. The functions of the 'L'-, 'M'- and 'R'-buttons fulfil the requirements being described in the list of 'functional B-requirements'. Activating the 'L-'button the main menu and the sub menu items could be selected. With the 'M'-button the so selected function can be activated or executed. Pressing the 'R'-button the current function can be interrupted or the sub menu will be left and the previous menu item of the main menu will be selected again.

Also the main menu has been modeled as asked in the list of 'functional B-requirements'. In the initial state of the tamagotchi no menu item is selected what is realized in the Petri net as a 'place' containing the marking '0'. This 'place' displays also the other menu items as a numerical code. Activation the 'L'-button the code of the first menu item of the main menu will be displayed in this 'place'. Pressing the 'L'-button after the last menu item has been selected, the first menu item will be activated again. With the 'R'-button the user is able to get back to the initial state.

Modeling the 'Timer with the M-net editor of the PEP system there appeared some difficulties to run the simulation in a realistic time (the duration was nearly the real time). Because of this some functional requirements which use the 'Timer' have not or only partially been modeled. That is the reason why no timeout has been realized which should cause a reset to the initial state of the tamagotchi, if no button is pressed during the time of five seconds while a menu item is selected.

1.2.3.2 Start, Restart and the Display of the Time

Because of the problems to realize the 'Timer' there is no possibility to display the time during the initial state or by pressing the 'M'-button. The paper-strip is realized as 'place' in the Petri net which will be marked with a 'dot' at the beginning (pulled paper-strip) so that the neighbored 'transition' can occur and the 'Timer' begins to count. The 'Timer' should represent the age of the tamagotchi. Because the 'Timer' which has been modeled with the M-net editor of the PEP tool can not be simulated, the following simplification was made: to make it possible to read out the age of the tamagotchi, three 'places' were modeled representing the number of the days, hours and the number of the minutes of the its age. These 'places' have to be marked with the values of the age of the tamagotchi during the simulation.

Another simplification is that the paper-strip could not be pushed back. In this case all current markings of the 'places' would have to be set back to their initial markings, and that would be impossible to be realized in such a comprehensive Petri net.

1.2.3.3 The 'Buzzer'

In the reason of having not enough time no 'Buzzer' has been modeled, that means that no critical states can be signalized with a 'beep'.

1.2.3.4 Satiation, Happiness, Weight and Age of the Tamagotchi

The sub menu items of the menu 'Status' can be activated by pressing the 'M'-button while the menu 'Status' is selected. To activate the menu items of the sub menu representing the variables 'satiation', 'happiness', 'weight' and 'age' of the tamagotchi the user has to press the 'L'-button. By doing this the values of these variables are displayed in the corresponding

'places' of the Petri net 'Variable-handler'. Activating the 'R'-button the menu item 'Status' of the main menu will be selected again. The values of the variables 'happiness' and 'satiation' can not be greater than four.

1.2.3.5 Nutrition

The functional requirement according to the nutrition, that means the function of feeding the tamagotchi is completely realized. Selecting the menu item 'Food and Snack' the tamagotchi can be fed with 'sushi' or 'sweets' by activating the 'L'-button once or twice and then pressing the 'M'-button to start the feeding with the selected kind of food. After providing 'sushi' the satiation of the tamagotchi is reduced by one unit and the weight increases to the next higher value. After providing 'sweets' the happiness increases to the next higher value and the weight will rise by two units of the weight. For these both situations no display function has being modeled which shows an eating tamagotchi in the current stage of evolution.

1.2.3.6 The Sleeping Time

The differences between the sleeping times and between the losses in weight during the night, depending on the stage of evolution, have also been realized. While sleeping the 'Marutchi' is losing two units, the teenager four and the adult is losing six units of its weight. During the night the tamagotchi becomes one year older and the 'L'-, 'M'- and 'R'-buttons are blocked so that all functions of the tamagotchi are turned off.

1.2.3.7 The Game

Also the model of the game for playing with the tamagotchi is completely realized according to the functional requirements. Selecting the menu item 'Game' of the main menu the Petri net of the 'Game' will be activated. In this Petri net during the simulation a direction (left or right), in which the tamagotchi will look, will be chosen accidentally. Using the 'L'-or the 'M'-button the user is able to choose the direction he thinks that the tamagotchi will look. The user has won a round if his choice was correct. He has won a game if he has won three of five rounds. Playing one game the tamagotchi losses one unit of its weight. Winning one game the happiness of the tamagotchi rises to the next higher value. Pressing the 'R'-button the game will be interrupted and the next menu item in the main menu will be selected.

For simplification no display of a playing tamagotchi corresponding to the stage of evolution has been realized. Also the regular 'beeps' of the tamagotchi during the game have been announced. The current scores of one game are displayed the whole time during the game as markings of two 'places' representing the number of won rounds.

1.2.3.8 The Stages of Evolution and the Characters of the Tamagotchi

In the reason of not enough time not all of the stages of evolution respectively the characters of the tamagotchi have been modeled. Only the characters 'Egg', 'Babytchi' and 'Marutchi' have been realized as a Petri net component with the M-net editor of the PEP system. Furthermore the transition of stage 'Marutchi' to stage 'Tamatchi' or 'Kuchitamatchi' can be simulated, to show how the condition (here: the number of played games per day), which is decisive for selecting the next stage of evolution, will be read out. The flying Tamagotchi will only be started if the Tamagotchi grows elder than 30 days.

The check of the minimum of the weight has not been. The current stage of evolution is displayed as a numerical code as marking of the corresponding 'place'.

1.2.3.8.1 'Egg'

Pulling out the paper-strip the timer begins to count and the tamagotchi is in the stage of evolution called 'Egg'. Because the simulation of the timer is too slow the markings of the places which represent the time have to be set manual so that other net components which depend on the time can be started nevertheless.

1.2.3.8.2 'Babytchi'

After five minutes remaining in stage 'Egg' the tamagotchi changes its stage to 'Babytchi'. The initial value of the weight will be five ounces and the values of the happiness and satiation will be set to four (maximum value). The satiation will be reduced all 15 minutes by one unit of the satiation.

1.2.3.8.3 'Marutchi'

After three hours remaining in stage 'Babytchi' the tamagotchi begins to sleep and will wake up again at 9.00 a. m. in the next morning being a 'Marutchi'. This stage of evolution will be displayed as marking in the corresponding 'place'. The sleeping time has been fixed from 9.00 p. m. to 9.00 a. m. because no special time was given with the list of the 'functional B-requirements'. The initial weight of the 'Marutchi' will be set to ten ounces and every hour the satiation will be reduced by one ounce. Furthermore every three hours the happiness will be reduced by one unit.

1.2.3.8.4 'Teenager'

After two days the 'Marutchi' changes into a 'Tamatchi' or a 'Kuchitamatchi' depending on the number of games per day. Is this number of games two or more the next stage of evolution is

a 'Tamatchi', in the other case the 'Marutchi' becomes a 'Kuchitamatchi'. The sleeping time will be fixed to 9.00 p. m. to 9.00 a. m.. The reduce of the satiation and the reduce of the happiness have not been realized in this stage of evolution.

1.2.3.8.5 Adult and the Flying Tamagotchi

This stages of evolution have not been realized, as mentioned before, as a Petri net model with the M-net editor of the PEP system and the flying Tamagotchi will be started if the tamagotchi grows elder than 30 days.

1.2.4 Is the Specification Consistent and Complete in itself?

The tool used by us supports "on the fly" simulation. With this function it is possible to check High Level Petri nets during processing again and again for its functionality. One can say therefore that each module is consistent and complete in itself. However we did not implement all modules completely. Despite the module Sleeper can handle the development stages Tamatchi and Kutchitamatchi, the module handling the stages itself were not programmed out completely. The adult stages Mametchi, Kutchipatchi and Masktchi were not implemented, because of a lack of time. After creation we checked most modules in a second step against the textual request in relation to the functionality of our model. The diagrams were reviewed thereby not "to foot" but with the simulator.

The modules Menu, Play, Sleeper and "Variable handler" were executed in the stages egg, Babytchi and Marutchi for a simulation run.

During the test runs of different modules we ran into miscellaneous problems. Often these could be attributed to problems of the Tools. So for example some transitions could not switch no more after joining. Mr. Grahlmann, one of the developer of the Tools, acknowledged on our request, that it concerns a well-known problem of the Tools here.

1.2.5 Where all detected errors recovered?

After joining all modules we recovered only errors, which were not in sub modules, so called "Boxes". The reason for it was, that subnets can only be processed, if these are "expanded" on the higher network. The term expanded means here, that the subnet is visible and therefore editable in the higher level network. Boxes are not stored in hl_net format, but in an own format (box ending). These can not be loaded directly into the simulator. Box files are solved from the high-level network they origin from. Thus editing the high-level Petri net doesn't matter the box file and updating an existing box is not achieved by the tool.

1.2.5.1 Remembered Errors

We made a mistake by implementing the game. Playing a game is not leading to a decrease of weight, if the tamagotchi wins. Additionally the display of the time-of-day was not merged into the model. This should be activated by pressing the middle mouse button, if neither the menu nor the game is busy. The development stage can only be changed while the tamagotchi is sleeping.

1.2.5.2 Size of the Specification

Our specification consists out of 8 high-level Petri networks . These are switched together on a "highest level" to an enormous Petri network. The program consists of approximately 150 places and at least the same amount of transitions.

1.3 Specification of the Functionality "Playing"

Almost all requests were implemented into the program behavior. A play can be started by selecting the menu option "Play". As already mentioned, there is no display showing the recent development stage of the tamagotchi. We simplified the point with the tone output, by simply allocate a dot to the place "beep". All other points are implemented completely, i.e. a game consists out of five rounds. Unless the right button isn't pressed, each game will be followed by a new one and each game can either be lost or won.

The Tamagotchi decides in each play accidentally whether it looks to the left or to the right. Subsequently, the user is requested to meet its selection. The player can indicate his choice by pressing the left or the middle button. If the selected key corresponds to the direction the Tamagotchi decided too, it will be displayed by marking a place. If the player decided for the correct direction, another place will be marked.

In our model the display of the score is visible as long as the play lasts. A play is won, if at least three out of five rounds were won. If the Tamagotchi wins a game its happiness will increase by a unit.

1.3.1 Conversion in the Model

Now functionality “playing” is to be explained according to the user demands of the specification. For this the individual request in the order of their occurring were numbered and the appropriate sections in the specification by the numbers were marked.

1. By selection of the menu option play in the main menu can be played with the Tamagotchi.
2. The display shows a playing tamagotchi in the respective development stage.

3. Even beep
4. it is played until the user aborts.
5. After each play a new one follows if it won't be aborted.
6. Plays can be won or lost.
7. Five rounds result in a play
8. The tamagotchi looks per round alternating once left or right.
9. The user must decide for a direction.
10. Line of sight is selected coincidentally by the Tamagotchi.
11. It displays whether direction is identically to user selection or not.
12. At the end a score will be displayed
13. Play is won, if at least three were typed correctly by five rounds.
14. The Tamagotchi loses a unit of weight per round.
15. Per won play the happiness of the Tamagotchi rises.

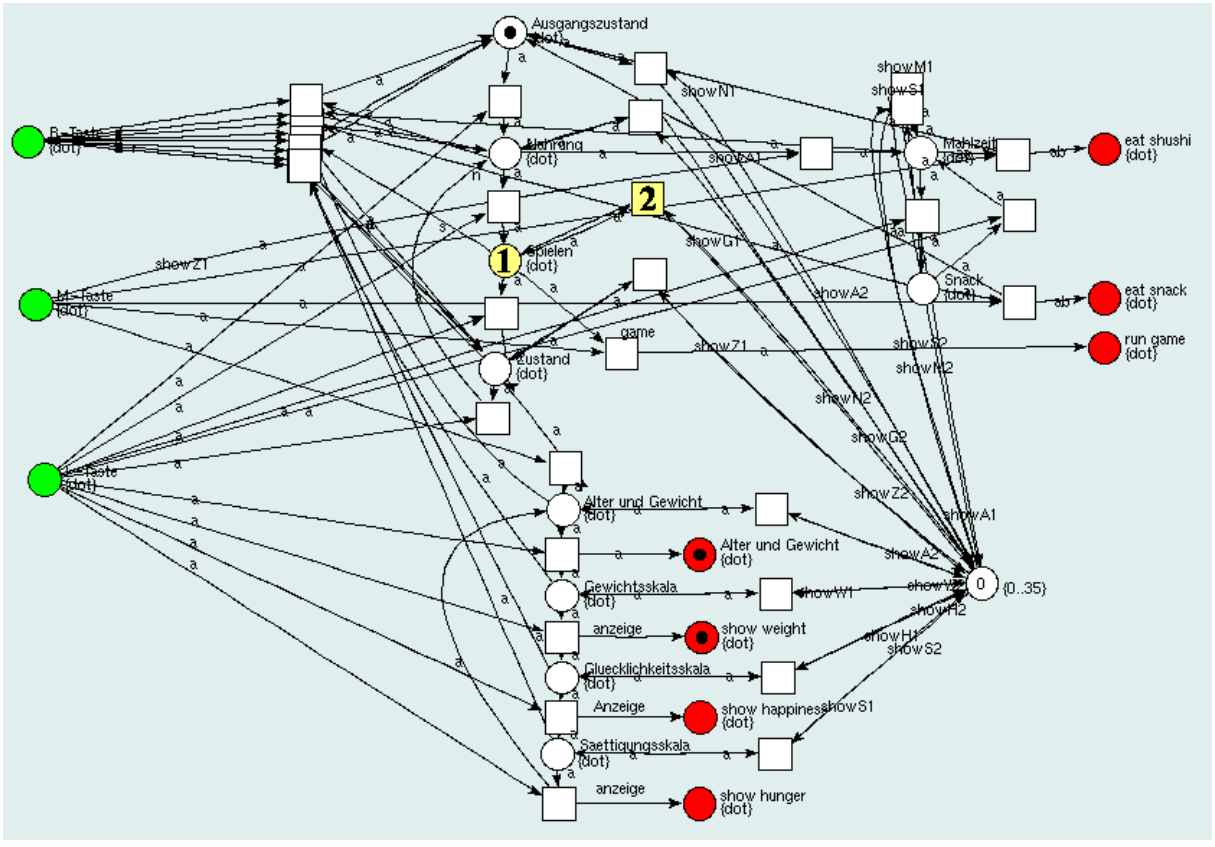


Figure 4

To 1. The user can navigate through the menu by pressing the left button. If one presses twice the left button, the menu option "play" is shown on the display - here coded as number 2. In order to simulate pressing the button, one must stop the simulator. Double clicking the place representing the button (here the place L-KEY). Now insert a

value in the field "current value". The expression DOT marks the place, what means, that the button is pressed. Values can be multiplied, by placing a multiplier in front of them. To indicate that the "L-button" was pressed twice times, the value term is: "2*DOT". Subsequently, the simulator must be shifted again in the Animate mode. Now the DOTs should be taken off one after the other from place "L-button". Thereupon the place "spielen" is marked in module menu (see Figure 4 point 1). The dot switches over the transition marked as 2 to the display of the menu. Now you can start a play by pressing the middle key. The methodology here is similar. Double click the place M-Key and insert the value "DOT" in the field "current value". Note that it does not have to lead you to the desired success, if one makes the entry both for the left and for the middle key in one step, because the DOTs are then taken off in coincidental order from the places. Marking the place M-Key makes the transition "game" adjustable. This now competes with the transition marked as 2. Adjusting the transition "game" will pass the DOT from place "spielen" on to the interface "run game", which is defined as “exit”. Passing the DOT outside from module menu also changes the owner of the display to the module keeping the focus. Now transition "run game" (Figure 6 number 2) will be passed. This transition has the task of doubling the DOT. Whereby the first DOT will be passed to "start game" and the second is passed to the place "piepen".

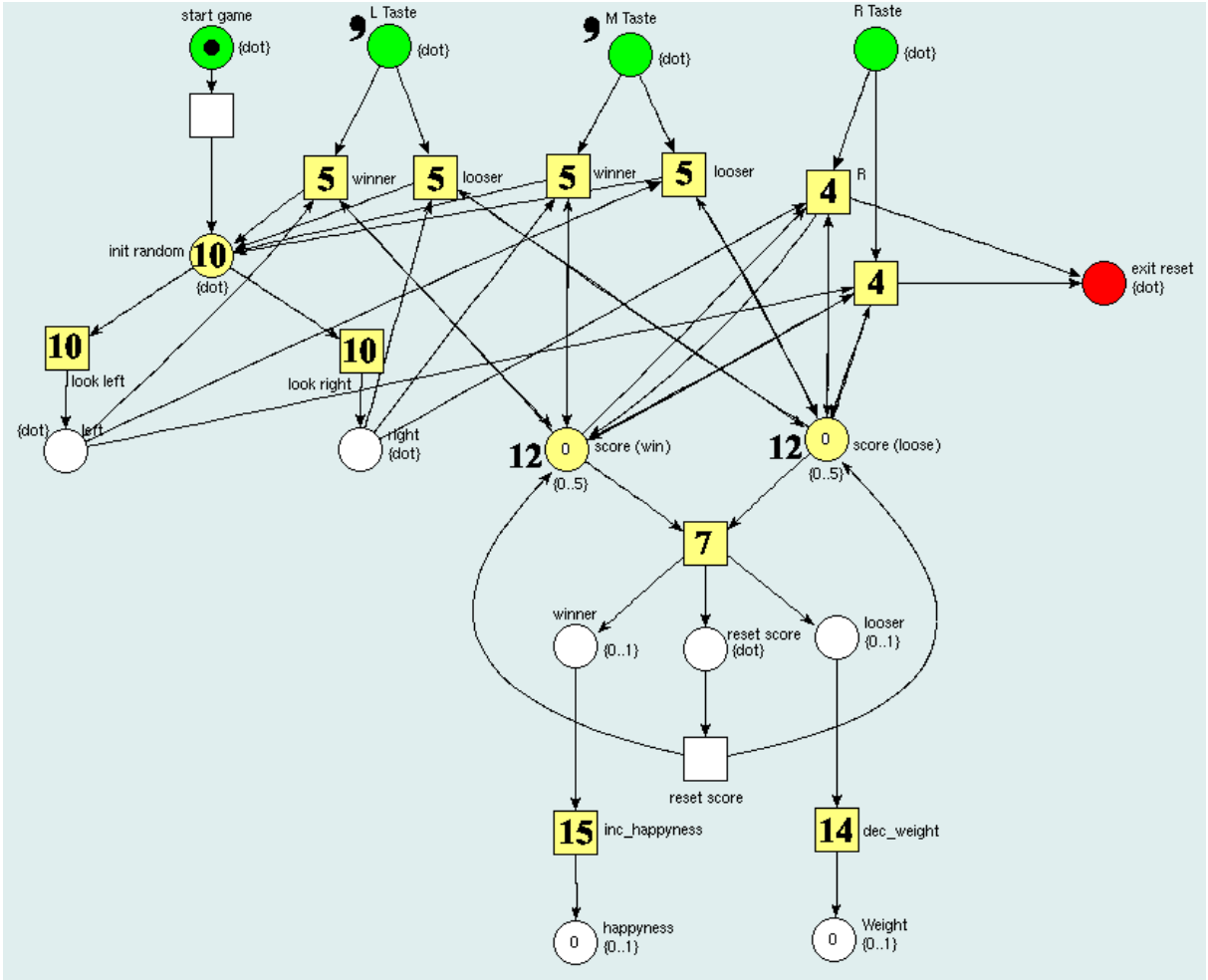


Figure 5

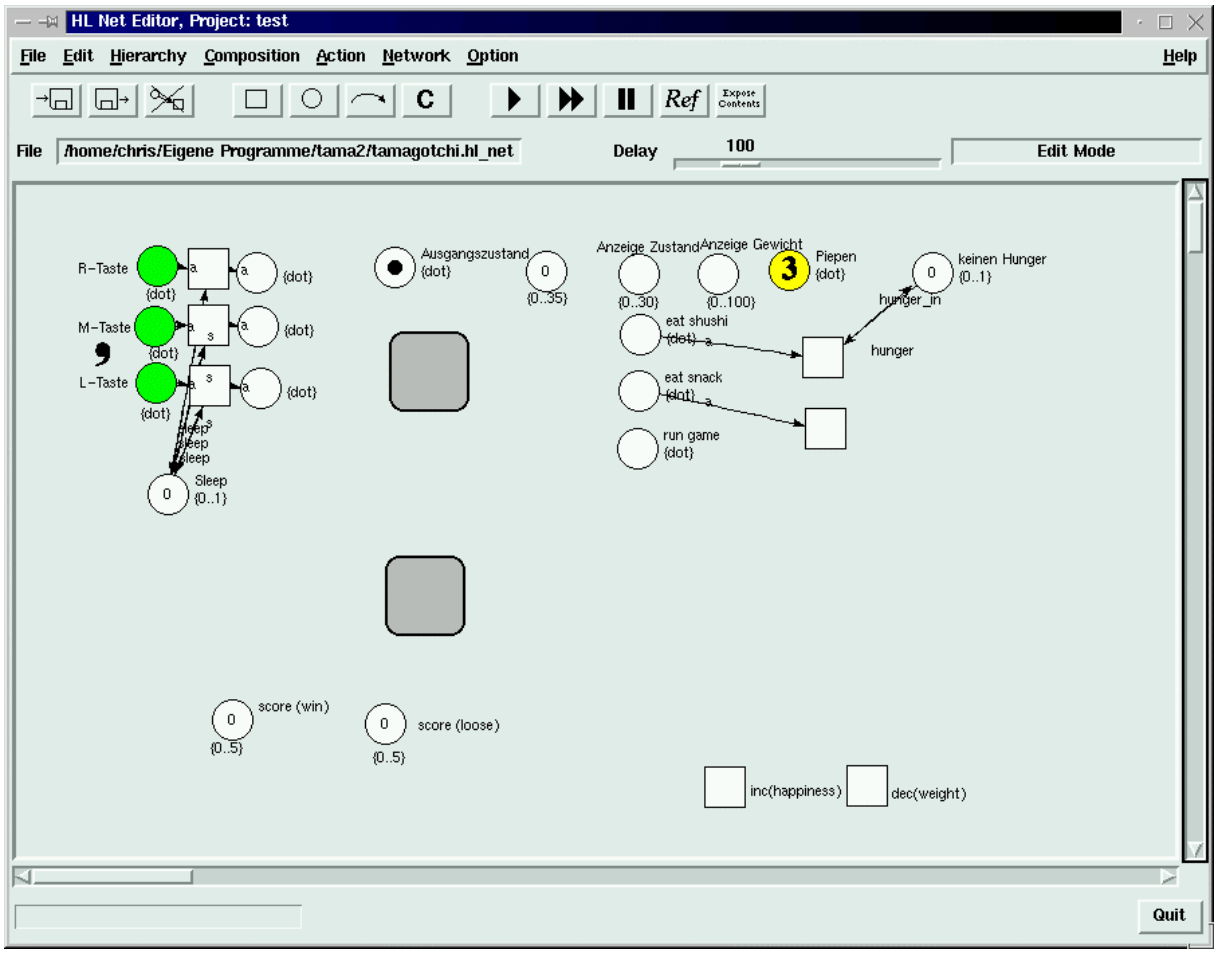


Figure 6

To 2. As previously mentioned, in our specification the display was not prepared as module. Rather it consists of several places, which show the individual values. It is unfavorable that there is no explicit representation, which shows, which place actually emulate the display. Therefore it is assumed, that as soon as the place “start game” receives a DOT a playing Tamagotchi in the respective development stage is shown on the display.

To 3. When switching the Transition "run game" (see Figure 6 number 2) additionally the place "piepen" (figure 3 number 4) is occupied. Occupying this place suggests - as previously mentioned - the output of a even beep while playing. It may be possible, that the Tamagotchi is implemented the same way. If it contains a building block, which indicates an interface for putting the beeper in an even beep mode. The tone will be outputted as long as the signal lies close.

To 4. In each round the play can be aborted by pressing the right key. If a play is not played over five rounds, no weight reduction and no happiness increment is possible. Pressing the R-Key will carry back module play to its initial state. If the markings would not taken off with leaving the play, it could occur, that although one is in the module

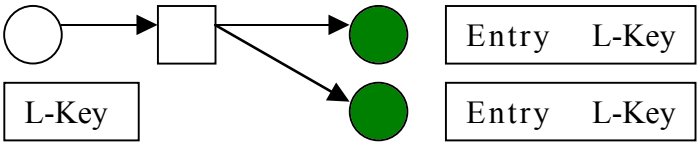


Figure 7

menu, would result in pressing a key on the module play. It should be mentioned, that integrating the modules made it necessary, to put the entry interfaces outward. Additionally the input interfaces of the keys for the modules “menu” and “game” were folded up (see Figure 6: L -, M -, R-Key). This was necessary only with these modules, since here the input interfaces have the same source. Note that it is not possible to connect two entry interfaces from one transition (see Figure 7). The transition would always be adjustable, so that the DOT would be passed on by coincidence to one of the two places (they reside inside a module). By folding up the input interfaces, the marking can be taken off from a following switchable Transition only.

It would also be possible to solve the problem over an additional place, which displays, which module is explained on the display. In this case it would not have been necessary to take the interfaces from the modules. We decided for our solution, because one could save four places and it should also be clear, which module is displayed. Leaving the module causes the signal for the beep to be taken off and the focus returns to the menu. This is then again in the initial state. On the display a Tamagotchi in the respective development stage is to be seen.

To 5. In order to be able to play the game round by round, the random number generator must be knocked again after each played round. Therefore the place “init random” (Figure 5) must be occupied with a DOT again. For this we drew a type feedback from the transitions of the two input keys (figure 2 point 5) on the place “init random”. If the player presses one of the two direction keys, the random number generator “is called” again automatically and the Tamagotchi decides in which direction it will look the next round.

To 6. If the player decides for the same direction as the Tamagotchi, the counter "score(win)” will be incremented, otherwise "score (loose)” will be incremented. It is not possible that a round will be lost and won both at the same time, because "init random" only contains one DOT per round. If a transition contains no explicit "value term", duplicating a DOT is not possible. Since one DOT can only switch one transition, it is not conceivable to win and loose at the same time.

To 7. Place 7 (Figure 5) checks continually, if five rounds played. If so, a winner or a loser is determined. For this the values of the places “score win” and “score loose” are constantly added and checked whether the total result is more largely directly 5. It is necessary to check, whether it is to be more largely, since our Petri net is not synchronized and therefore it would be possible, that the sixth round would follow too fast after the fifth round, and then the check value 5 could never be achieved. If the condi-

tion is fulfilled, the results are compared and a winner or a loser is determined. The places “happiness” and “weight” will be set to a value of 0 or 1 depending on who won the game. Happiness and weight are exit interfaces, thus they pass there data outward at the "variable handler". The Transition "reset score" ensures, that the counters are reset to zero.

To 8. This point wasn’t implemented, because there is no explicit display. Implementing a new place with the meanings for looking to the right and looking to the left only would make it more difficult to read the main diagram. Thus we decided to not implement this point as required.

To 9. The player has to guess which line of sight the Tamagotchi will decided itself. Pressing the left (view to the left) or middle (view to the right) button will show the tamagotchi, for which side the player decided for.

To 10. Our random number generator consists out of the place "init random" and two transitions “look right” and “look left. The functionality of a random number generator already places the Petri network respectively the Peptool. You can switch only one transition from one place. If two transitions are switched after one place like in our case, then the DOT is distributed by the Petri network simulator coincidentally on one of the two transitions.

To 11. Since the score of a game is visible during the whole play over, an explicit representation whether the played round was won or lost was omitted. However it would have been possible to create its own display for this. We decided for reasons of clarity since each further place would made the backbone network only more difficult readable. Therefore the player has to check out the score of the places score(win) and score(loose) in order to find out, whether he win or loose the round.

To 12. There is also no explicit display for indicating, whether the player won or loose the last game. Implicitly this can be seen by watching the places “winner” and “looser”.

To 13. In this point the textual request is somewhat inaccurate. A game will be won, if at least three of five rounds were won. Although the request would allow, to interpret, that a game is won if exactly 3 of five rounds are to be won, we didn't implement it this way, because this does not reflect the behavior of a real tamagotchi.

To 14. As previously mentioned the request was not transferred correctly. In our model the Tamagotchi loses only one unit of weight, if a game will be lost. The decrement of weight will be done by moving the value 1 to the place looser. This place is connected with the variable “weight” respectively with the decrement interface of the variable. The interface accepts numerical values, which will be passed over a transition to the variable itself. The transition reads in the recent value of variable weight and subtracts the value passed from it. The result will be written back afterwards.

To 15. If the Tamagotchi wins, its happiness rises a unit. The happiness cannot exceed the maximum value of 4 units. Nevertheless if one run a game, the Tamagotchi can only lose weight!

Up to filling out the questionnaire we had not thought over, what would occur, if the Tamagotchi will go sleeping while a game will be played. Since Petri networks run to a considerable degree parallel, the passing from awake status to sleep status would lock the input of the L- M- and R-Key for module Sleeper. Thereupon the display would have to be prevented from freezing. Since we do not have a separate module handling the display, this is not possible. Thus the display "would freeze". The same would also apply to the output of the buzzer. Since the buzzer is waiting for a stop signal, after it is started, the tone would be played during the whole night on.

If the Tamagotchi wakes up the next morning, it would be in the place in the play, in which it fell asleep the day before. The play could then be continued at the place the play was stopped.

It was tried to operate all statuses from the textual request. In this step also a model view of the Tamagotchi was sketched(figure 4). We implemented modules as separate high-level Petri nets. Within a high-level Petri net another one can be embedded as so called "Box" via the menu option "load module". We used these feature for example in Figure 6, where the modules from Figure 4 and Figure 5 are embedded into the highest level network.

Statuses become places in the Petri network. We tried to cast all data types, which were not processible to the two data types of the simulator. For example we decoded the display of the menu by numbers. Furthermore we tried to work out all changes in status. These are treated in the Petri network as transitions. However we had the problem beforehand yet not to have operated on the High level Petri network level. We had worked out the "safty injection" example as BP(N)_ program. However it is not possible to simulate in this modus and the translation into a High level network was satisfying everything. Therefore we gain experience by programming out the handling of the module menu as already mentioned in 1.2.1. Concretely all menu options and interfaces became places. One could very beautifully solve dependencies between the individual menu options and the keys, which receives an incoming arc from the preceding menu option and the appropriate key. The menu option itself gets an outgoing arc on the following menu option (see also Figure 6).

1.4 Experiences

1.4.1 Getting started

First of all we started with the modeling of the "Safety Injection" System [Parnas,1993]. Altogether, as a group of three, it took us about 72 hours to get it done completely. We spent eight hours on reading manuals, about seven hours on creating a Petri net-model.

After creating a model in the HL_NET-Editor we decided to create it with the B(PN)_-modeling language. Which is very simple, easy to learn and the notation is similar to pascal. The reason for our decision was that with a bigger problem, such as the Tamagotchi, there will be a need for a better overview of the model and of its parts. Using the PEP-Tool it is

quite simple to translate such a given piece of B(PN)_-code in a high-level Petri net. After the translation of our "Safety Injection" System we spent another hour on net cosmetics because once you translate your code it is not that simple to understand the translated high-level Petri net afterwards, because of the complexity of the net.

When modeling the system we experienced that the manual of the PEP-Tool was of little use, because they only explain how to use the editors and how to handle pull-down menus, but not how to create a proper Petri net step by step. Besides that the models shown in the manual are all high-level Petri nets which were translated from B(PN)_-code. And for example some of the arcs are hidden in those translations, because they are only visible in the low-level Petri net. This made it very hard to understand the given examples and the way they work. So for us at the very beginning of modeling high-level Petri nets in the HL_NET-Editor it was quite hard to create a functioning model due to the fact, that we did not understand why those models shown in the manual and the given example-files were working and our created nets were not, even if they looked just the same in the HL_NET-Editor.

Another problem of understanding the manual was triggered by the fact that all the screenshots were based on version 1.4 and we had the actual version 1.8. So most of the time the screenshots were of no use. Even the translation of some B(PN)_-code ended with the following error: "Number of transactions exceeded" and the problem was that this program was only about 100 lines of code. So as a matter of fact we decided to start modeling the Tamagotchi in the way we started with the "Safety Injection" System: With the HL_NET-Editor. This Editor is very simple and quite easy to use. You build a Petri net as a set of transitions, places and arcs which can be named afterwards. It is very easy to get more experience, because you can start the simulation once you have one or several transitions marked as beginning and ending transitions. The single step simulation mode is ideal for stopping the simulation at any time and for manipulating some transitions or values. So it is quite simple to learn more about the functioning of the net simply by trial and error.

1.4.2 Modeling the Tamagotchi

The modeling alone took us about 105 hours together as a group of three. We started modeling the whole lifecycle which took us about nine hours. After that we decided to start modeling with the HL_NET-Editor for known reasons. First of all we modeled the menu-handler that took us about thirty hours, afterwards we implemented the game and food component which took us approximately eighteen hours. We tried hard to get a timer component working, but after twelve hours we finally had to give up. Subsequently we implemented the sleeper component in about twelve hours, after which we created the lifecycles which we had modeled before. This took us about eighteen hours to get it done. Finally we did the variable handler component in six hours.

We tried to divide the designing of the components, but as it was the first time for us to work with the PEP-Tool, none of us did his part alone, but more or less together in a group. Only at the end of the modeling phase we had enough experience to do components all on our own. That is why we cannot just multiply these numbers by three. But working on different components is easy to manage once you have gained the needed experience and the group is sure about the interfaces they want to use. Otherwise when you want to connect the high-level Petri nets you will have severe problems. At the beginning it was hard for us to "think" in

Petri nets and we had some initial trouble because we realized that e would not be able to use variables and that we would have to cope with places instead. The only Problem we had to manage was the fact that when a place is polled, all the information is gone. So we had to implement multiple polling, which made the Petri nets look more complex. One disadvantage of the PEP-Tool is that no timers are implemented. Thus we had to implement the timer ourselves. A timer, however, consists of many multiple polling, leading to the problem that the simulation was very slow because the low-level Petri net generated by the tool for simulation reasons was very huge. In some cases the simulation did not work even when the created net was "correct". So we had to omit the timer component and take the single step mode of the simulation to manipulate our global time.

Another problem was that the PEP-Tool does not provide a real hierarchy which leads to the problem that the Petri nets are very fast growing and soon very complex. We tried to reduce complexity by using the so called boxes, provided by the tool. But it is not possible to label a box and when you take some parts of the net into a box, the arcs which were combined with the rest of the outside net are not visible anymore Consequently it is harder for the viewer to understand the functioning of the net, but the simulation ,nevertheless, acts fine.

1.4.3 Review and simulation

We did our review together with the group who had the ObjectGEODE-Tool. The PEP-Tool offers several features for a review / simulation. On the one hand there is the possibility of an immediate simulation at any time leading to a far better understanding of the net. Furthermore the PEP-Tool offers an interruption of the simulation process at any time as well as the opportunity to change certain values and continue simulation afterwards. On the other hand, the complexity and simply the size of the Petri net make it hard for someone not really familiar with the matter to understand the model. The flat hierarchy of the net is a negative fact as well. Another point is that the boxes used to make the net smaller and easier to understand can not be labeled and that the incoming and outgoing arcs are not visible in the drawing, leading to a negative traceability of the model.

But of course there are some nice feature of the HL_NET-Editor and the simulator as well, such as the good visualization of active components in the Petri net and the polling. An active scrolling during the simulation would be desirable but unfortunately this has not been realized yet. The known "box problem" is again of negative effect, because if you want to make the contents of a box visible in a so called frame, you will have to face the problem that some Petri nets will be disarranged in the frame and an up and left scrolling will not be possible. To put it in a nutshell: It is very hard for someone not familiar with the problem or Petri net to get a clue of what the net does and why. Therefore the outcome of the review and simulation phase was that it is very hard to find and to inspect the user requirements given in the specification.

1.5 Conclusion

Finally we have to admit that the PEP-Tool made group work pretty easy to handle, as long as everybody was quite sure about the interfaces. It is very simple to build a Petri net in the HL_NET-Editor since only three different components are needed: Transitions, places and arcs. The immediate simulation supports "Rapid prototyping" techniques. The first experiences with the tool are unfortunately based on trial and error, because of the bad manual and examples given in it. It would be great to have the chance of modeling on a higher abstraction level and to have a flowcharter.

The simulation of bigger Petri nets or multiple polling is very slow and sometimes not possible. For us the HL_NET-Editor was not quite stable enough: Saving and reloading some nets leaded to total chaos as some arcs between places were missing, others were somewhere else. It will be very helpful to have a syntax checking while you enter polling-code in the transitions. An undo button would be very helpful as well. Another point is that the copy function acts only visually, because it copies the net but all the polling in the transitions are gone. So to put it all together: We would only use the PEP-Tool once again if it is more stable and reliable.

1.6 Appendix