JavaFSM

by Karola Krönert and Ulrich Dallmann

ENGLISH translation by Holger Rehmeier
IPCT, PUCRS, BRASIL (more tools)

The written version of our pre-theses in Postscript-Format (German only) is located here.

Introduction and motivation :

Through the increasing complexity and integration in chip-design, more complex course-controls are necessary. Therefore, finite processors gain more importance in modelling in this area. Nevertheless, there is still a lack of suitable graphic devolpment tools, especially in the area of public domain software.

Furthermore, the aim originated in the work-area TECH of the university of Hamburg is to make technical computer science more clear to the student of basic informatics through practical examples. So, the idea originated to develop a program that fulfills both requests. JavaFSM is therefore for students for introduction, as well as for developers for practical use.

As programming language, the new, object-oriented language Java of Sun Microsystems was chosen. Java offers several advantages compared to conventional programming languages :

Java is platform-independent. A Java-program is translated by a compiler in platform-independent byte-code, that is then executed to the term of a Interpreter, the Java Virtual Machine. The Virtual Machine is implemented for all current systems in the meantime.

Java is object-oriented. With the object-bearings, the data and their manipulation stand in the foreground, not the procedures. Objects can transmit their qualities to other objects or can be supplemented with more options.

Java offers extensive class-libraries, that cover beside standard-functions also new areas, as network- and data base-access. Since the libraries come already with the client in the Java-interpreter, these don't have to be loaded from the network.

Beside Standalone-Applikationen, Java offers the possibility to embed programs in HTML-pages. These so-called applets are executed then by the virtual machine integrated in the browser. Therefore Java-appletses are accessible through the World-Wide-Web. Another advantage of applets is that no installation is necessary. Also updates can be done through a simple exchange of the classes on the WWW-Server.

Through the platform-independence from Java and the possibility that applets can be embeded in HTML-pages, Java is especially suitable for the use in education, since a great number of people can be reached.

Reference-platform of Java is the Java Developers Kit (JDK) of Sun company.

The JDK offers the possibility with the command javadoc, to generate a documentation of the classes in HTML-format from the sourcecode and the comments automatically.

Introductory literature to the topic Java and interesting Java-links are located at
tech-www.informatik.uni-hamburg.de/Students/2kroene/java/java/javalinks.html.

Program-construction :

JavaFSM can be started as an applet, as well as an application.

JavaFSM consists of three windows : in the first, the control mechanism-model can be seen with the in- and outputs. Here, an machine-name can be given and the type of network is defined. Furthermore, the in- and outputs can be added, altered and deleted. After defining the machine, it can be simulated here. Therefore you find a bar for clock and reset. In the delta-network, the machine is displayed. The current sate and the transition activated under the momentary conditions is marked red.

The machine is designed in the editor-window. Here, states, transitions, transition- and output-functions are defined. Additionally, it is possible to provide a vending machine with comments. The editor works modus orientated. Using buttons, you can change to the referring mode. There are six different modes.

Conditions can be moved in the move-mode ("Move"). Furthermore, transitions and states can be selected, to change their options (name, function, etc) in the parameter-field,, (see also selection-algorithm). Whether a Moore- or Mealy-machine is edited, the parameter-field of the states changes. While each output at Moore has solid output-values, a function can be defined with Mealy (using inputs). The Lambda-network can be "simulated" this way, even without a correct state-definition.

The state-mode ("State") inserts states per mouse-click. In the transition-mode ("Transition"), two conditions are connected by a transition clicking on each state. The transitions are displayed as arrows between two states. The algorithm to draw the arrows was taken from the program JavaFIG in [He97]. Components can be removed by clicking to the delete-mode ("Delete"). You fix the start-state in the start-state-mode ("Start-state"). In the comment-mode ("Comment"), comments can be inserted. Since the machine may not be changed during the simulation, the editor-window is closed automatically when starting a simulation.

The third window with the impulse-diagram offers the possibility to display the simulation-results. For a better differentiation, the inputs are blue and the outputs red in different color-gradations.

A problem occured with the impulse-diagram with Mealy-machines. Here, the output-values can change also

between the clock, if the input-values change. Here, we have introduced "inter-clock". Each alteration of the output-values generates such an "inter-clock". Additionally, a vector "clock-series" inserted into the class FSM that contains a "true" for each real clock and for each "inter-clock" a "false". Only with Mealy-machines both types are displayed.

FSM :

The class FSM was drafted for presenting and processing of finite state machines. All data of the machine are summarized here. Furthermore, the class provides all basic-functions for processing.

The basis-components of a finite state machine are states, transitions, inputs and outputs. Their options are displayed in the following table :

State	Transition	Input	Output
• Name • Position • Output-function • Start-state?	• Initial-state • Goal-state • Transition-condition	• Name • Initialvalue • Current value • Value-sequence of the simulation	• Name • Current value • Value-sequence of the simulation

The dynamic values of the simulation are saved in each case directly with the corresponding object, i.e.

- each in- and output contains a vector with the value-sequence
- the class FSM contains a vector with the state-sequence
- since with the Mealy-machines the outputs can change not only by clock, FSM contains a vector clock-sequnce, that enables a differentiation of clock and "inter-clock".

"Just-in-time-parsing"

The calculation of transition-tables for the finite state machine necessitates exponential calculation-expenditure and additional memory. With an alteration of the machine, the tables must at least be calculated at the start of a simulation. The used alternative in JavaFSM is the "Just-in-time"-parsing of the required functions during the simulation. Only transitions, that start from the current state, are taken into account.

For the calculation of the transition-conditions and the output-functions in the states, the class Parser is used, that receives the respective function as string.

Bigger calculation-expenditure (exponentially) happen only during the checking of machines ("FSM testen"-button in the editor) and exporting to KISS, because all functions with all possibilities must be calculated here.

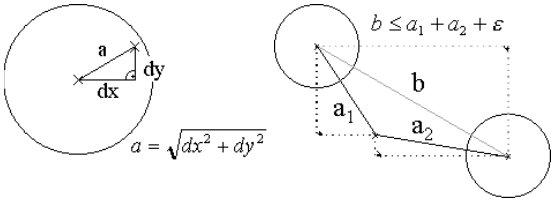
Method to check a defined machine

The editor contains a function, that allows to check the machine for syntax errors.

The "FSM-test" checks first, if conditions and transitions are existing, and if one of the states was defined as start-state. It calculates all possibilities then and determines, whether in each case exactly one transition is activated. In a dialogue-window, errors are displayed.

Selection-algorithm in the editor :

The selection of conditions simply takes place through the calculation of the distance between state-center and click-point (see illustration). In order to select transitions, the distance of the click-point to the line (plumb) should actually be calculated. A simpler possibility consists of the comparison of the distance of both states with the sum of the distances of the states to the click-point. If this is not really longer, the transition has been selected :



Condition-selection

Transitionsselektion

Load and save

JavaFSM offers the possibility to save designed machines and to load them again.

For this you use "File - Load/Save".

In the applet, a name and a password is entered in a dialogue-window and is saved with the machine under this name on the server.

A FSM-Datei has the following construction :

[JavaFSM V1.0]	Header with version-number
[***Machine Type***]	MEALY or MOORE
[Name]	Marks the namen-block
Name	The name of the machine
[Inputs ***quantity***]	Marks the input-block, with the number of inputs
name,initial	Name and start-value of an input (each line contains exactly one input)
[Outputs ***quantity***]	Marks the output-block, with the number of outputs

name	Name of an output (each line contains exactly one output)
[Zustaende ***quantity***]	Marks the states-block, with the number of states
name,xpos,ypos,isStart,output1_fkt...	Name, position, isStart (boolean) and for each output the output-function
[Transitions ***quantity***]	Marks the transitions-block, with the number of transitions
name1,name2,fkt	Names of the two states connected and the transition-function
[Kommentare ***quantity***]	Marks the comment-block, with the number of comments
xpos, ypos, text	Position and comment-text (this can contain "%10" for a new line)
[ENDE]	The End

Example :

```
[JavaFSM V1.0]
[MOORE]
[Name]
Example machine
[Inputs 2]
a,0
b,1
[Outputs 3]
x
y
Z
[Zustaende 2]
state 1,58,129,true,0,1,0
state 2,229,222,false,1,0,1
[Transitions 4]
state 2,state 1,b
state 1 state 1,*
state 1,state 2,a
state 2,state 2,*
[Kommentare 1]
10,10,this is an example machine
[ENDE]
```

Load and save with CGI

A Java Applet cannot access the filesystem of the lokal computer (security). A possibility to the save machines consists of to pass out text in a window and to store this by means of Copy&Paste. This way, we have done with the convertibility of the machine to VHDL / KISS, so that the user can process the data directly.

Sending an e-mail with the data to the user represents another possibility to save data. This however does not solve the problem of saving of machines yet.

The solution, that we have chosen to save and load machines, is a network-connection to the host. Instead of implementing an own server in Java, the application of the existing WWW-Servers offeres itself. A WWW-Server offers the possibility with the CGI (Commin Gateway interface), to start programs, [He96]. These CGI-Programmes are executed over a certain URL. You can generate HTML-Seiten for example dynamically (like search-engines).

The communcation between browser, server and CGI-Programm : the browser sends an inquiry to the server and waits for an answer. The server starts the aquired CGI-Programm, which processes the inquiry, that generates the aquired answer and sends it from the server to the browser.

In the Hypertext-Transfer-Protokol (HTTP) there are two commands for browser-inquiries : with the GET-kommand, data can be send only directly to the URL. Since a URL is limited in its length, this is not enough all the time. All big data can be sent with the POST-kommand.

JavaFSM uses separate CGI-programmes to save and load. When loading, the applet sends the machinename and a password as parameters. The CGI-programm looks for that machine and checks the affiliated password. If no errors appear, the data of the vending machine is sent to the applet. To save a machine, the applet also sends name and password, followed from the data. Provided the name is not forgiven yet, as well as the password is correct, the CGI-Programm writes password and data into a File. The answer to the applet consists of a confirmation or a error message.

The password is mainly used for the protection of accidantally overwriting an existing file through other users. So, the designer of the machine can access the file only.

With applet-parameter, existing machines can be implemented into the menu as examples and be loaded from there.

More literature :

- HTTP-spezification : www.w3.org

Parser

The Parser is needed to calculate logical expressions of the transition-conditions and output-functions (only with Mealy).

Basic-units of an expression are the operators AND (&), OR (|) and NOT (!), the names of the inputs as variables and the values 0 and 1, as well as brackets. The grammar of the parsers is :

program :
    END
    expression END
expression :
    expression | term
    term
term :
    term & primary
    primary
primary :
    number (0,1)
    name (inputs)
    (expression)
    ! primary

The syntax-analysis is done in the manner of the rekursiven descent(top down). Each prozess-rule of the grammar corresponds to a function, the executes other function. Terminale symbols (0, 1, names of the inputs) get recognized by the syntax-analysis-functions expr(), term() and prim(). As soon as both operands of a (sub-)expression are known, it is appraised.

The Parser uses a function get_token () for reading the input. This reads the next basis-unit of the expression and saves them in a global variable. Before executing a parser-funktion the next token of get_token is determined. If get_token reads a name, it is searched in the vector of the inputs with the function look() and the current value is determined.

With the call of the parsers, the expression is extended with a return, which is used for the recognition of the end. If Get_token() finds this sign, the evaluation-process is finished and the truth-value of the expression is reruned. If an error occurs during the evaluation-process, a BadExpressionException is done, which ends the Parser appruptly.

Export of machines to VHDL / KISS

In order to process defined machines more further, JavaFSM offers the possibility, to convert these into the formats VHDL and KISS. The edition of the VHDL- / KISS-codes is exported into a text-field, from where you can copy & paste it.

VHDL

VHDL is a hardware-description-language accepted world-wide meanwhile. It was originally developed for the documentation of circuits : the from outside visible behavior is defined in form of (legible) behavior-descriptions. High-level-language-struktures like IF-THEN-ELSE or CASE are used within. In circuit-design, so-called synthese-tools generate complete networks in form of gate-network-listings from it. These can be optimized on different goals (time, space,...).

Since VHDL is supported by numerous tools, it is also used for data exchange. Front-End tools generate compiler-specific VHDL-Codes, that can be processed by other tools .

To be able to use generated machines with JavaFSM it is possibe to export them to VHDL.

KISS / PLA design format

KISS is mainly used witch programmable logic. It is a technology-independent format, which is based on truth-talbes. Beside miscellaneous features like "don't cares", "subsets" etc it supports also finite state machines. A Design-File consists of a header with basic-information and a Body with the actual logic-tables. The table used by JavaFSM is described here :

#	Commentary-line

.design ***name***	Name of the design
.inputnames ***name_1, [name_2...]***	Here, all inputs are defined (incl. clock- and reset-Signal). In this sequence, they are listed in the value-table.
.outputnames ***name_1, [name_2...]***	All outputs are defined here. In this sequence, they are listed in the value-table.
.clock ***signal sense***	One of the inputs is defined as Clock-Signal here • signal : Name of the input • sense : can assume the values "of rising_edge" or "falling_edge" (Flip-Flop is activated with rising or falling edge of the clock-signal)
async_reset ***signal sense state***	One of the inputs is defined here as Reset-Signal • signal : Name of the input • sense : can assume the values "of rising_edge" or "falling_edge" (s.o.) • state : state, that should be assumed after a reset, (start-state)
010 z1 z2 01	a line of the value-table : • input-value • sate regarded in this line • state, in to the machine changes with this configuration, • output-value with this configuaration
01 - z1 z2 01	a "-" in the input-value marks a "don't care".

In JavaFSM it is abstracted by a concrete condition-coding. If a certain coding is required (for example as direct edition), you can define it in KISS later :

.encoding	state-coding
state_name encoding	• state_name : state • encoding : numerical value : 2#1010 binary 10#10 decimal 16#A hexadezimal

JavaFSM tech  Ulrich Dallmann  Karola Krönert

JavaFSM

by Karola Krönert and Ulrich Dallmann

ENGLISH translation by Holger Rehmeier
IPCT, PUCRS, BRASIL (more tools)

What is a FSM?

A FSM (FiniteStateMachine) is a formal model, with which many problems of the computer science can be described. In this program, finite state machines are used for the modelling of control mechanisms.

That machine describes a system, that exists of a final number of internal configurations - so called states. The state of the system includes the information, that has emerged from the previous inputs and those, that required, in order to decide the reaction of the system on the still following inputs [HU93]. One of these conditions is defined as start-state. The system can now change the state under certain conditions. The states are graphically displayed as circles and transitions as arrows, signed with the transition-conditions (see illustration).

Formally, a machine (A) exists of a finite quantity of conditions (Q), an input-alphabet (I), a transition-function (d), a start-state (q0) and a quantity of final-states (F).

$$A = (Q, I, d, q_0, F)$$

With the modelling of networks, the quantity of the finalstates is cancaled, as it is not about accepting words but about the calculation of output-values and transition-conditions. The input-alphabet is $(0,1)^n$, as n is the number of the inputs.

Example :

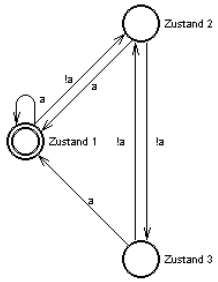
Q={Zustand 1, Zustand 2, Zustand 3}

q0=Zustand 1

I=(0,1) (values for a)

d(q,z)->(q ')

for example : d(Zustand 1, !a)->(Zustand 2)



In principle, there are determining and non-determining machines. The transitions are always defined unequivocally with determining machines, while there can be several possible transitions with non-determining machines, of which the machine must guess "the right one". For the design of networks, only determining machines are used, thats why they form the basic this program.

In JavaFSM there is not, as in the formal model, only one single transition-function declared. Instead, for every transition one transition-condition is defined. Because it is a determining machine, exactly one of the transitions is activated in each case.

In chip design, final state machines are used for the design of clocked control mechanisms, because they simplify the transposition of many electronic functions, especially sequence-controls. In order to avoid designing with unwieldy and error-susceptible logic-tables, new hardware-description-languages often offer a direct syntax for final state machines.

JavaFSM tech  Ulrich Dallmann  Karola Krönert

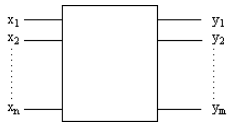
JavaFSM

by Karola Krönert and Ulrich Dallmann

ENGLISH translation by Holger Rehmeier
IPCT, PUCRS, BRASIL (moore tools)

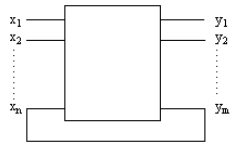
The processing of binary information takes place in digital computers with help of logical circuits.

A network has n inputs and m outputs :



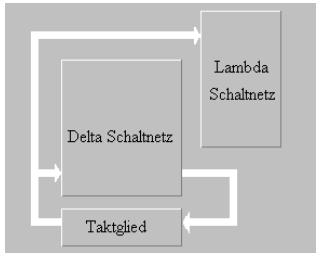
At the inputs, binary values $x_1,x_2,...,x_n$ with x_i out of {0,1}can be input. At the outputs, binary words are output according to the input-signals $y_1,y_2,...,y_m$ with y_i out of {0,1}.

Is the output only refering to its momentarily input, so it is named **control-network.** Is it also refering to former outputs (repatriation of the outputs), so it is called a **control-mechanism**. Information can be stored by means of the repatriation.

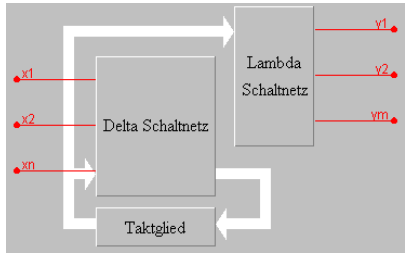


As direct repatriations are rarely easy to handle even with small control mechanisms (Hazards), a clock-device is inserted into the repatriation. For a better overview, the switch-network is divided into two parts : the first part is used for the regulation of the consequence-state (delta-switch-network), the second calculates the edition-values (Lambda-Schaltnetz).

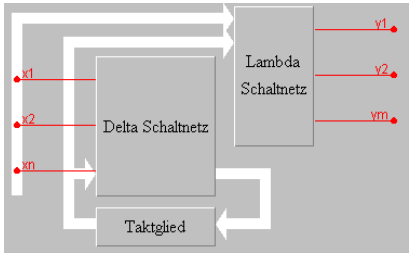
The control-mechanism is designed as follows :



- **Delta-switch-network :** here, out of the state, in which the machine is now, and the input-condition the following state is calculated.
- **Bar-limb :** here, the condition in which the machine is now is stored by Flip-Flops. With a clock-signal, the following condition, that has been computed by the delta-network, becomes the new condition in the delta- and lambda-network. A condition-alteration can only take place with a clock-signal.
- **Lambda-network :** the current outputs are calculated here. With a **Moore**-machine, these refer only to the current condition. However with a **Mealy**-machine,also the inputs are included to the calculations. So, also between two clock-signals alterations can be generated. Mealy-machines are not mightier however, because (apart from a delay) there is always an equivalent Moore-machine.



Moore-Schaltwerk



Mealy-Schaltwerk

An important description-method for control mechanisms are **Finite State Machines.** JavaFSM uses these for the design and offers the possibility, to export defined machines as control mechanisms to VHDL and KISS.

JavaFSM tech  Ulrich Dallmann  Karola Krönert

JavaFSM

by Karola Krönert and Ulrich Dallmann

ENGLISH translation by Holger Rehmeier
IPCT, PUCRS, BRASIL (moore tools)

JavaFSM as application
Selection between Mealy - and Moore networks
Saving and loading of designed machines
Loading of example-machines
Definition of the machine name
Definition of In- and Outputs
Design of the vending machine with help of the editor
Switch between edit - and simulation-mode
Definition of the logical transition-conditions and output-functions
Check the machine on correctness
Simulation through changing of in- and output values and clock
Output of the simulation results on the impulse diagram
Export of the machine in the VHDL / KISS-format

JavaFSM as application

To execute JavaFSM, you require the JDK of Sun. Download the file JavaFSM.zip and unpack it. Besides the program JavaFSM, it contains also some example-machines and documentation. Change to the JavaFSM folder and start the Interpreter doing this :
" Java -classpath PATH DE.uni_hamburg.informatik.tech.JavaFSM.JavaFSM ", the path must point to JavaFSM.

Using Windows 95 / NT you can adapt the included batch file "JavaFSM.bat ". If JavaFSM is not direct on the drive C : \ , you have to enter the path there. Furthermore, the path must be set on the JDK

Selection between Mealy - and Moore networks

When starting the JavaFSM, a dialogue-window appears to the selection of the network type. Using Moore networks, the output values only depend on the calculation of the machine. However with Mealy, the input-values influence the lambda network, too, so the output values are depending on the input-values and can change their value without clock.
Through the menu "Machine - Moore" as well as "Machine - Mealy" the type of network can be altered.

However this is only possible in the edit mode, because the machine may not be changed during the simulation. With the change of the Mealy nchaltwerk to the Moore nchaltwerk, complex edit functions get lost and are replaced with "0".

Saving and loading of designed machines

JavaFSM offers the possibility to save designed machines and to load again.

In the application, you can select the file name in the menu "File - Load / Save". To the uniform labeling, these should carry the extension " .fsm ". On this occasion it is to be heeded that the CLASSPATH must be put down! Applets are prohibited to accesses the client computer due to security and it is not possible therefore to store data on the user drive directly.

With selection of the menu "File - Load / Save" in the Applet a dialogue-window opens . Files are saved under a name and a password, that should prevent an inadvertent overwriting and protects data of other users. You should be aware that all machines are saved on the server in the same folder. Therefore, a machine should get an unambiguous name (username_machinename). After entering the filenames and the password and confirmation the data is transferred to the server. Possibly appearing errors are passed out in the status line. If the machine was properly loaded / saved, the dialogue window closes. If an already existing filename is chosen when saving, the affiliated password must be declared. Otherwise, the error message that the file is already existing / password-error appears. Choose another filename instead.

Firewall / Proxy

A Java Applet may build a network-connection only to the server, from which it was loaded, due to security. Is there a Proxy-Server (Firewall) between this connection, unfortunately no machines can be loaded as well as saved, because the Proxy itself cannot be contacted! Nevertheless whoever would like to work with JavaFSM can download it and start as application.

Loading of example machines

In the menu "File - Examples" are some example machines, that are loaded from the Webserver. If any errors should appear, these are passed out in the status line. This selection is not possible in the application. There, the examples can be loaded from the subdirectory directly Beispiele.

Definition of the machine name

In the menu "Machine - Machine-name" you can choose a name for the machine. It appears on the window titles then.

Definition of in- and outputs

With the buttons as well as the menu "Signal" you can install the in- and outputs, altered as well as delete.
New : Opens a dialogue window, in that the name of the input / output and with inputs the initial value can be defined. A name is allowed to contain letters, digits and the signs "/" and "_", but the first sign may not be be a

digit.
Change : After the selection of the change-button or the menus click on the input / output to be changed. The dialogue-window appears, in which the alterations can be done.
Delete : After the selection of the delete button or the menu click on the input / output to be deleted.

Design of the vending machine with help of the editor

With the Menu "Machine - Editor", you get to the machine editor. The left side contains the buttons and below the input-field. The grafic display is on the right.

The editor works modus orientaded. The buttons have following functions :
States : clicking the button "State" leads into the mode insertion of states. Conditions can now be inserted by clicking on the grafic display.
Transitions : clicking of the button "Transition" leads into the mode to insertion of the transitions. Inserting a transition one connective state after the other is selected. Clicking the first state, colors itself red. The following state is to be clicked afterwards. The transition is inserted. The state last selected remains selected (red), so that it is already the starting state for the next transition. Clicking on empty space, the state is unselected and so a new starting state can be selcted.
Deleting : clicking of the button "Delete" leads to the deleting modus. Everything, on that is clicked in this modus, gets deleted. Deleting states, also the affiliated transitions get deleted.
Start-state : In this mode, the the starting state can be fixed through clicking. A starting state must be fixed in each machine.
Move : States and comments can be moved in the move mode. Moving states, the transitions will move themselves with them. Furthermore, transitions and states can be selected, to change the transition conditions, the edit functions, and to define the names of the states in the options field. For a comment, the text can be defined in that field as well.

Comment : Here, comments can be inserted on the grafic display. These are framed and may include several lines. By clicking on the "Assume"-button, the text is pasted into the comment.

Definition of the logical transition conditions and output functions

When clicking on one transition (in the move mode), there appears a text field beyond the buttons, that contains the logical transition function. Following syntax is valid :

& : logical AND

| : logical OR

! : logical NOT

Transitions pointing from a condition, the transition function can also contain "*". That "*" stands for all other conditions, i.e. this transition is activated, if no other transition is activated.

Check the machine on correctness

Clicking on "FSM test" (in the editor) the current machine is checked on correctness. Errors are passed out in a window. With more complex machines, this can last some minutes (beside the size of the machine the number of inputs is relevant for the calculation time). It checks first, if conditions and transitions are existing at all, and if one of the states was defined as start. All possibilities are calculated, after that for each possibility it is testet if a following state is defined. In a dialogue window, statuses and errors are displayed.

Switch between edit and simulation mode

The program distinguishes between edit and simulation mode. In the editmode, inputs / outputs can be defined and are inserted into the machine. In the simulation mode, input values can changed, the clock generated and the impulse diagram displayed. Choose the menue "Simulation" to switch between simulation "Simulate" and edit "Edit". Changing the mode from the menu simulation to edit all previous data gets lost.

Simulation through changing of in- and outout values and clock

After changing to the simulation mode, the values of the inputs can be changed by clicking. Through clicking on the clock button "Clock" (or the menu "Simulation - Clock") the clock is generated. The machine calculates the following state and changes steps to it. At the outputs new the values of the refering state are given. Through clicking on "Reset" (or menu "Simulation - Reset") the machine is reset (inputs and outputs as defined and start at start state). The impulse-diagram is also reset.

Output of the simulation result on the impulse diagram

Using the menu "Impulsediagram - Show" (only in the simulation mode) the impulse-diagram is opened.

Export of the machine in the VHDL / KISS-format

Through the menu "File - Convert - VHDL / KISS" the machine is converted to the VHDL / KISS ormat. Because an Applet cannot store (security), the text is displayed in a text-field, from where it can copied and pasted into a normal editor.

JavaFSM tech  Ulrich Dallmann  Karola Krönert

JavaFSM

by Karola Krönert and Ulrich Dallmann

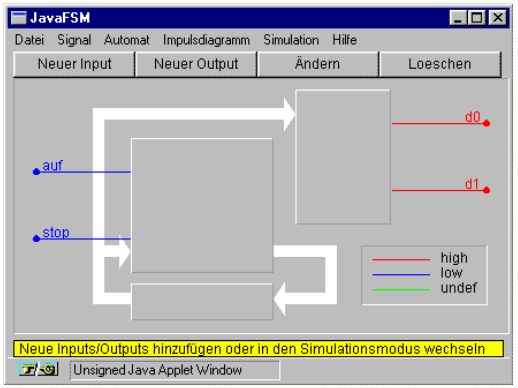
ENGLISH translation by Holger Rehmeier
IPCT, PUCRS, BRASIL (more tools)

Example machine

To explain JavaFSM a 2-bit-upward /downward-counters is constructed. It can be found in "File - Examples - Counter" in the example folder.

As we don't require asynchronous outputs, a Moore-Automat is enough. Either we already select this with the start of JavaFSM, or we put set in the menu "Machine".

Now, we require two inputs with the names "auf" (up) and "stop". The first one gives the counting-direction and the second stops the counter. Furthermore, we require two outputs with names "d0" and "d1".



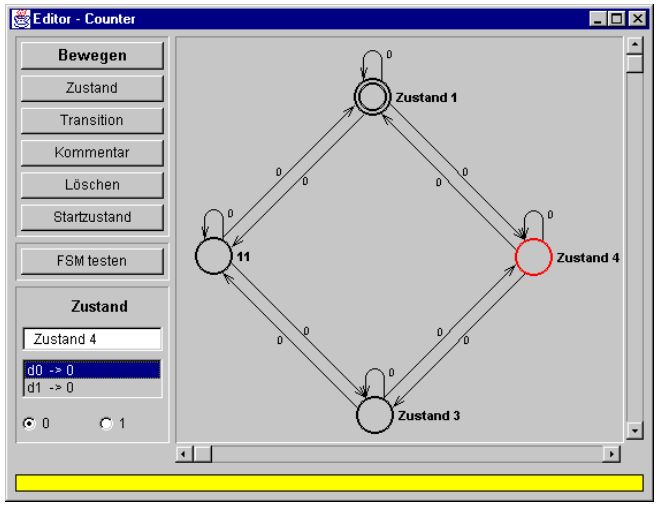
In the menu "Machine - Machine-name" we can name the machine. Then it appears also in the title of the windows.

Using the menu "Machine - Editor" we get into the editor and edit the machine there.

First, we change to the insertmode for states with the "State"-button. With each mouse-click onto the grafic display, a new state is inserted there. In the "Move"-mode, the conditions can be arranged at will afterwards. Generate 4 states doing this.

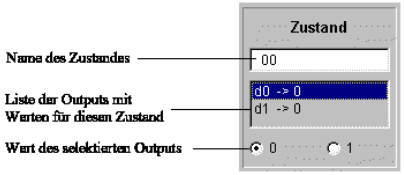
Now we form two opposite circles of transitions, first changing to the transition-mode clicking on the "Transition"-Button and click from state to state then. The first selected state (red marks) is the start of the transition. The next clicked state it is leading to. If no state was is marked red, then the state is selected merely.

In order to be able to stop, each condition needs a transition on itself (loop). We click twice on each state.



In order to give the states meaningful names, we click on these in the "Move"-modus. As result, the options of the state are displayed in the window left below. The name can be altered in the input-field. This alteration is

entered with *Enter*. In our example, we name the conditions " 00 ", 01 ", 10 " and " 11 ".

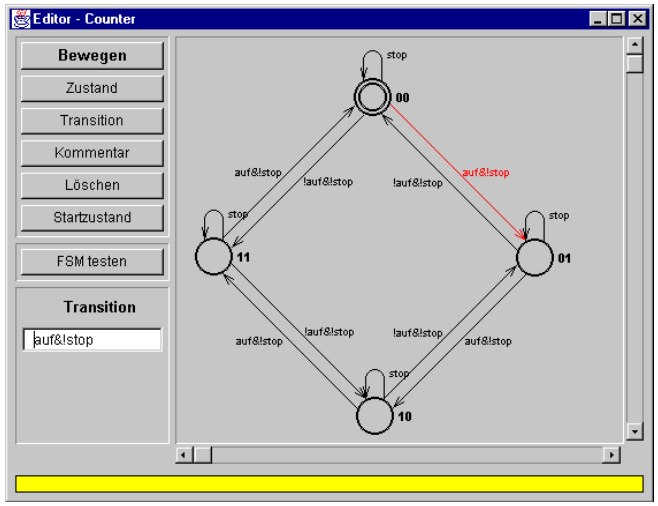


Furthermore, the output-values of the state can be defined in the option-field (for d0 and d1). The corresponding outputs are selected in the list and the value is selected :

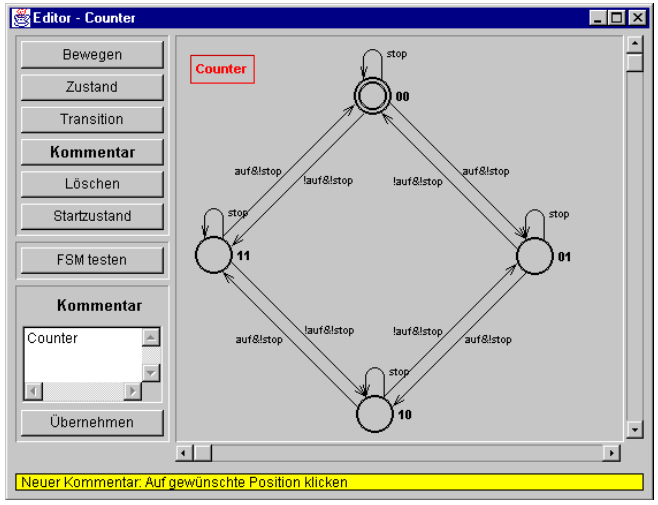
Since it is a Moore-machine in this example, only "0" or "1" can be declared here. With Mealy-Automaten, it is also possible to declare more complex functions at this position.

Afterwards, the transition-conditions of the transitions are defined. Through clicking on the transitnions (in the "Move"-mode) there appears a corresponding input-field in the options-field.
As written above, alterations are only entered after pressing *Enter*.
For transitions, that proceed clockwise, we enter "auf&!stop", for transitions, that proceeds contrary to the clock sense, we enter "!auf&!stop". For loops, the transition-condition is "stop".

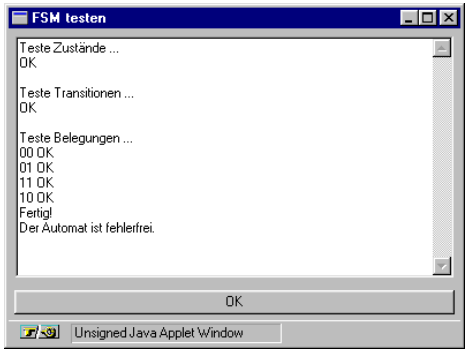
Now, we select a state as start, pressing the "Start-state"-buttonfirst and after that the state "0 ".



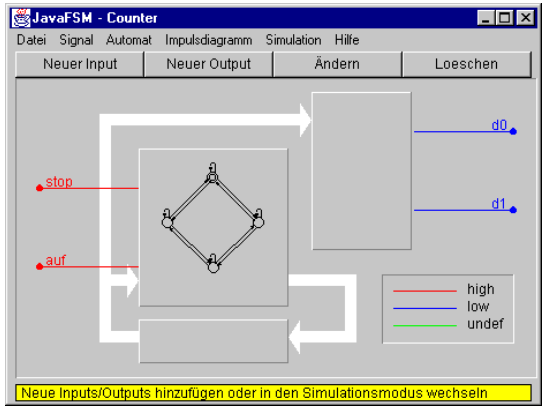
Additionally, it is possible to provide the machine with comments. You change to the "Comment"-mode and insert the comment. In the options-field, the comment-text can entered then and is pasted on by clicking on the "Assume"-button.



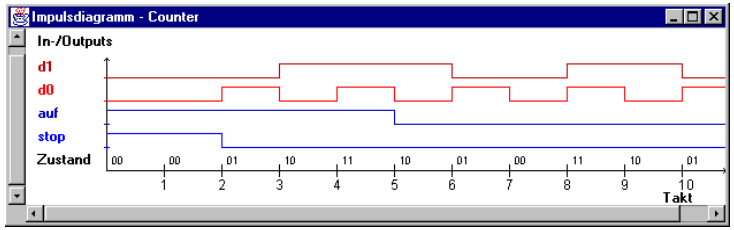
To finish, we can even test the machine with "FSMtesten" if it is correct. Following display should occur :
("Der Automat ist fehlerfrei" = the machine has no errors)



We now change to the simulation-mode, in that we choose the menu-point "Simulation - Simulate" in the main-window. In the delta-network, the machine is visible. The current state and the momentarily active transition is marked red. Through clicking, we can alter the values of the inputs. With a click on the "Clock"-Button, the next state is calculated and the machine changes to it. The outputs assume the values defined in this state.



Through "Impulsediagram - Show" the impulse-diagram can be displayed.



To save the machine you choose "File - Save".

A file-window opens in the application, in which the filename can be entered.

You should be aware, that all files are saved in the same directory. Therefore, an unambiguous name should be chosen (username_machinename). Additionally, a password is given. This protects from overwriting through other users. An existing file can be overwritten only with the propper password. If an already used file-name with a wrong password is chosen an error occurs (file existing / wrong password).

A saved file can be loaded later through the menu "File - Load", file-name and password must be entered.

In JavaFSM it is possibe to export designed machines in VHDL and KISS. In the menu "File - Exportt" you choose the propper format. Unfortunately, Appletses have no possibility to access local files. The text is displayed in a seperat window where it can be processed by copy and paste.

JavaFSM tech  Ulrich Dallmann  Karola Krönert

JavaFSM

by Karola Krönert and Ulrich Dallmann

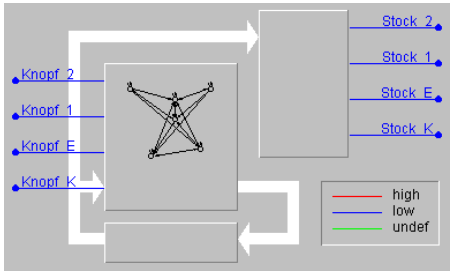
ENGLISH translation by Holger Rehmeier
IPCT, PUCRS, BRASIL (more tools)

Example "Fahrstuhl" (elevator) :

The "Fahrstuhl"-example models an elevator, that serves 4 floors.

As input-values, four buttons stand for the respective floors. It doesn't matter in this case, whether somebody in the elevator presses the button 2nd level or somebody in the 2nd level calls the elevator.

As output-values, we have four signals, that declare, in which floor the elevator is now.



The machine contains 6 conditions :

Condition	Description	K	EC	1	2
Keller	the machine is in the cellar	1	0	0	0
Erdgeschoß vu	the machine is coming to the ground floor from below	0	1	0	0
Erdgeschoß vo	the machine is coming to the ground floor from above	0	1	0	0
1. Stock vu	coming from below the machine is in the 1.floor	0	0	1	0
1. Stock vo	coming from upwards the machine is in the 1.floor	0	0	1	0

2. Stock	the machine is in the 2. floor	0	0	0	1

Is the elevator in the ground floor or 1. floor, so it shall, provided several buttons are pressed, proceed in the same direction first. Therefore two conditions were inserted for these floors for each case, that define from which direction the elevator is coming.

Transition-conditions :

If only one button is pressed, the elevator should go into this floor of course. If several buttons are pressed, it should retain its direction and should approach the next floor that has been called.

from	after	if
Keller	Erdgeschoß vu	Knopf_EG
Keller	1. Stock	!Knopf_EG&Knopf_1
Keller	2. Stock	!Knopf_EG & !Knopf_1&Knopf_2
Erdgeschoß vu	Keller	!Knopf_2 & !Knopf_2&Knopf_K
Erdgeschoß vu	1. Stock	Knopf_1
Erdgeschoß vu	2. Stock	!Knopf_1&Knopf_2

and so on