Plant Simulation
Finite State Machine library
Reference manual

TECNOMATIX

SIEMENS

Proprietary and Restricted Rights Notice

TECNOMATIX
Plant Simulation

SIEMENS

Plant Simulation

Finite State Machine Library

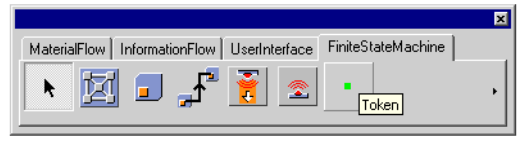
Version 9.0

December 2008

1

Finite State Machines

The objects of the library *FSM* (Finite State Machines) allow the modeling of complex material flow systems, how it is realized by industrial robots.



The application objects of the library *FSM*

Basic notions

This chapter gives a short introduction to Finite State Machines (*FSM*). The concepts for Finite State Machines, described in the technical literature, were extended by additional functionalities, which facilitate the modeling of dynamic flow of material systems.

States and Signals

An *FSM* receives signals from other *FSM*s or from material flow objects, which are watched by a assigned sensor (input of the *FSM*). The *FSM* processes these signale and affects the flow of material by signals to an object, which which releases appropriate actions (output of the *FSM*).

A *FSM* can take different states. At one time the *FSM* has a clearly identifiable state, which is represented by the movable element *Token* on the an object *State*. At the beginning of the simulation the *FSM* is in the state *Start*. A final state is not intended in the library *FSM*. The *FSM* can change from a state into another, if one or more conditions are fulfilled (transfer function of the *FSM*). A condition is determined by the occurrence of signals. Every time a *FSM* receives a signal, it examines whether thereby its state can change.

According to the permanence of signals we differentiate the *signal types*:

1. *permanent*: If the *FSM* changes into a new state, some or all permanent signals can be deleted, which were needed as condition for this state change.
2. *persistent*: A persistent signal remains existing always. It can be *active* (i.e. valid) or *inactive* (i.e. invalid).
3. *impulse*: An impulse signal is immediately deleted after the examination, whether this signal is a component of a condition. These signals are deleted also if they are not a component of the condition for the next condition change.

Sending signals

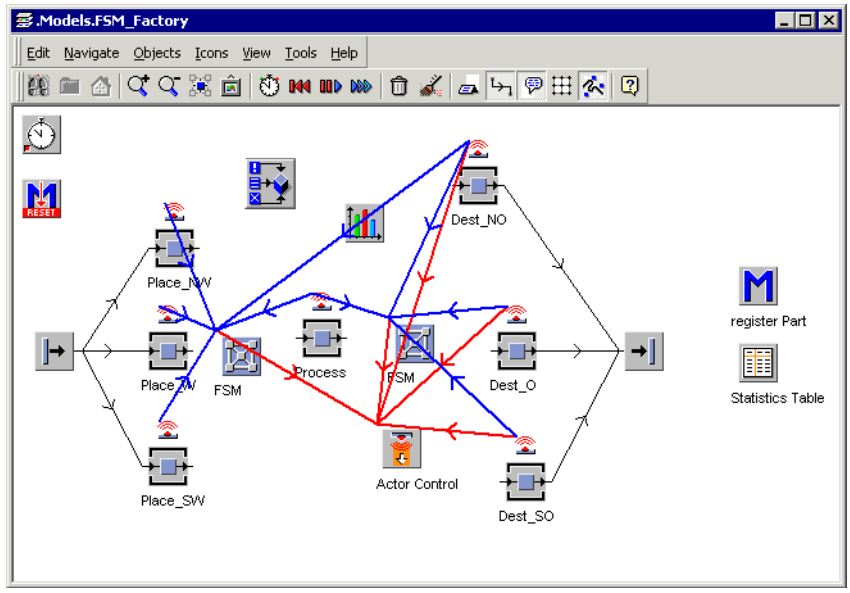
If a *FSM* changes the state, then the *Token* changes the object *State* and signals can send to other *FSM* and/or *ActorCtrl*. If a certain material flow event occurs on an basic object (like a *singleProc*), then sensor control sends signals to the objects *FSM* and/or *ActorCtrl*. The sensor control reacts to the following events:

- *empty*: The object became empty.
- *exit*: An movable element left the object.
- *entrance*: An movable element entered into the object.
- *ready*: The processing of the movable element is finished.

Receiving signals

A *FSM* registers all received signals. After the examination, whether a state change is possible, signals are deleted according to their permanence.

If an action is to be done by a signal, then the signal is sent to an the object *ActorCtrl*. The object *ActorCtrl* performs an action according to the signal, which can lead to a material flow event, an animation (icon change) or to a method call.



Animation of the signals with special marking of the signals to the object *ActorCtrl*

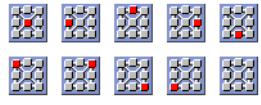
The elements of the FSM

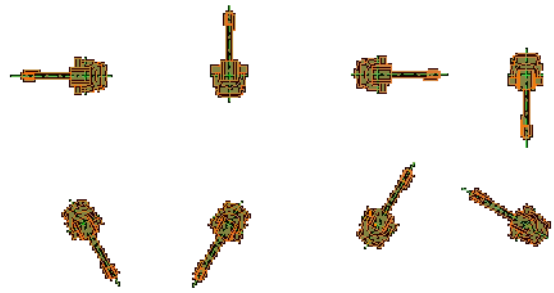
After the description of the concepts and fundamental ideas of the *FSM*, now the application objects are described. Since for signals no application objects is intended and communication between the objects of the *FSM* is realized by signals, the objects of the *FSM* have menu and context menu entries Animate signals, Show signals and Delete animation. The exchange of signals is visualized by showing comments and connections. By the menu and context menu entry Help you can open a short explanation for the application object.

FSM (Finite State Machine)

Symbols: 

The object *FSM* represents an automat. Into this object you can insert arbitrary many objects *State*, which the *FSM* can take. For the animation of the different states the *FSM* has the following icons. For special applications further icons can be added.

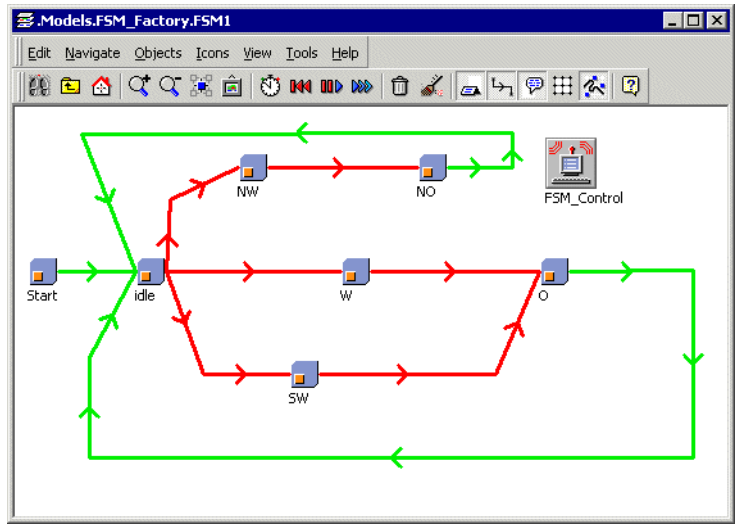




Prepared icons of the *FSM* for the animation of the states

At the beginning of the simulation the *FSM* is in the state *Start*. The appropriate state object is already inserted in the class.

A *FSM* can be failed. During this time the *FSM* shows the Icon with the name *failed:* 



States and transitions of the *FSM*

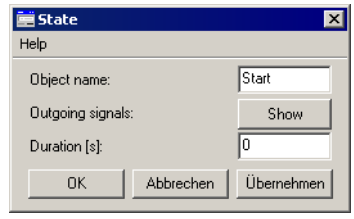
With the occurrence of a state actions can be linked, which are done by signals to the object *ActorCtrl*.

State

Symbols: 

This object represents a state of the *FSM*. As soon as the token enters into the state object, signals to other *FSM* can sent. Likewise signals can be sent to an object *ActorCtrl*, in order to do certain actions, e. g. a move of a movable element in the model.

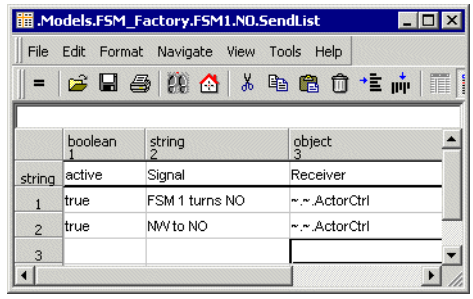
For the purpose of visualization a new icon for the *FSM* can be set.



The dialog of the *State*

Object name: This dialog element contains the object name. This name must be a valid designator in the frame of the *FSM*. Read in addition in the Plant Simulation manual 'User-Defined Names'.

Outgoing Signals: If the *FSM* accepts a new state, then signals can be sent to other *FSM* or *ActorCtrls*.



Signals, which are sent when occurring a state.

In this table the columns have the following meaning:

1. active: In this column it is adjusted whether the signal is set (*true*) or reset (*false*). This column has only for persistente signals meaning.
2. Signal: Into this column the signal name is entered.
3. Receiver: Into this column the object is entered, to which the signal is to be sent. Enter here only application objects *FSM* or *ActorCtrl*.

Dauer [s]: If the *FSM* accepts a new state, then firstly the signals defined under Outgoing Signals are sent. Thereafter the *FSM* examines whether it can change to a new state. The duration between sending the signals and this examination is 0, by default. Then the state object shows the first icon.

For time-dynamic simulation it is however sometimes important that a certain duration lies between these events. This duration is modelled by the processing time of the Tokens on the state object. A state object with time consumption shows the second icon.

signal_connector

Symbol: 

So that the *FSM* can change from a state into another state, you must connect these states with this connector object. The transition from a state to another state is defined by a condition, which is described by a combination of logic operations like AND and OR of signals. These conditions are assigned to this connector object between state objects. After connecting two state objects you can open a table by a doubleclick, in which you can enter the conditions. To describe an AND operation of signals you must use &. To describe a OR operation of signals and/or of AND operations, you must enter it into different lines. For the condition

(*Signal_A* AND *Signal_B*) OR (*Signal_1* AND *Signal_2*)

you must fill the table as follows:

	string 1	integer 2
string	signal	priority
1	Signal_A&Signal_B	2
2	Signal_1&Signal_2	3
3		

Condition table of a transition between two states

In this table the columns have the folowing meaning:

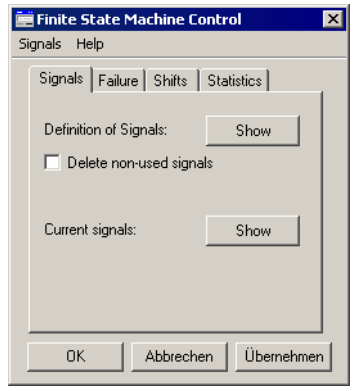
1. signal: In this column a AND operation of signals is entered by.
2. priority: By the priority you specify the order of the evaluation of the AND operation.

The connection object is red, if a condition is defined, without a condition connection object is green.

FSM_control

Symbol: 

This object controls the *FSM* and offers different settings. The object is inserted in the *FSM* by default.



The tab Signals of the *FSM*-Control

Definition of Signals: In this table you enter the permanence of the signals. The value *permanent* the predefined default value.

	string 1	string 2
string	signal	type
1	Process ready	permanent
2	O empty	persistent
3	SO empty	impulse
4	NO empty	permanent
5		
6		permanent
7		persistent
		impulse

The definition of the permanence of signals

In this table the columns have the following meaning:

1. signal: The entries of this column automatically adjusted during the first simulation run.
2. type: Select the permanence of the signals here.

Delete non-used signals: The transition from a state to another state depends on a condition, which is described by OR and/or AND operations of signals. After a change of the state permanent signals are deleted, which can be contained in an OR operation of signals. With this check box you can adjust whether all permanent signals of AND operations or only

the signals of the fulfilled AND operation are deleted. If all signals are deleted, then also the not used signals are deleted, which are contained in possibly not fulfilled or not examined AND operations.

Let us see the following example of a condition of permanent signals:

(*Signal_A* AND *Signal_B*) OR (*Signal_1* AND *Signal_2*)

If this check box is selected, then all permanent signals are deleted after a transition of states, which are contained in the condition of the transition. If the signals *Signal_A*, *Signal_B* und *Signal_1* are given in our example, then all three signals are deleted aften the transition.

If this check box is not selected, then only signals are deleted, which are contained in the AND operation of the condition. If the signals *Signal_A*, *Signal_B* und *Signal_1* are given in our example, then only signals *Signal_A* and *Signal_B* are deleted aften the transition. The not used signal *Signal_1* is not deleted.

Current signals: In this table the signals existing at the current simulation time are administered.

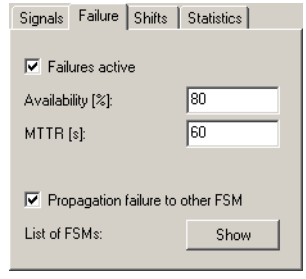
	string 1	object 2	boolean 3	string 4
string	signal	sender	active	type
1	Process empty	~.~.ProcSensor	true	permanent
2	W occupied	~.~.Sensor_W	true	permanent
3	SW occupied	~.~.SensorSW	true	permanent
4	W ready	~.~.Sensor_W	true	permanent
5				

At the current time existing signals.

In this table the columns have the following meaning:

1. signal: In this column the name of the signals are contained.
2. sender: The object had sent this signal to the *FSM*.
3. active: If a persistentes signal is active (i.e. valid), then here *true* is registered.
4. type: The permanence of the signal is here entered.

For the simulation of failures you use the settings on the tab Failure.



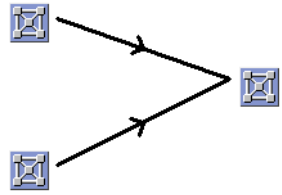
Availability of the *FSM*

Failure active: With this check box you adjust whether the *FSM* is to be failed. During a failure the *FSM* remains in the current state. There is no separate failure state . A failed *FSM* receives further all signals, but the *FSM* does not change its state.

Availability [%], MTTR: Here you enter the desired availability and the mean time to repair (MTTR).

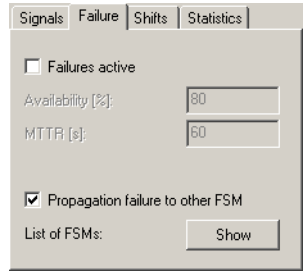
Propagation of failures to other FSM: As soon as the *FSM* is failed, this failure can be passed on to other *FSM*. The availability of the *FSM*, which are contained in the list of *FSM,* entered on the tab Failure will be reduced thereby in the simulation.

List of FSMs: In this list you define the *FSM*s (also by Drag & Drop), to which the failure are propagate. These *FSM*s are combined into so-called protected circuits. This list can also be opened over the context menu entry Propagate failures.



Propagation of failures

The *FSM* is failed, if it has an own failure (that is by the internal generator of failure) or a failure of the *FSM* in this list happened. The propagation of failure to be visualized by the context menu entry Animate protected circuits.



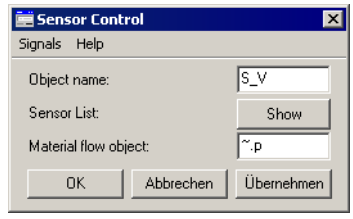
FSM without own generation of failures

On the tab Shifts you can register you an calendar object for a shift system. Alternatively you can drag a calendar object onto an object *FSM*. The dialog of the automat control with the registered calendar object opens.

SensorCtrl

Symbol: 

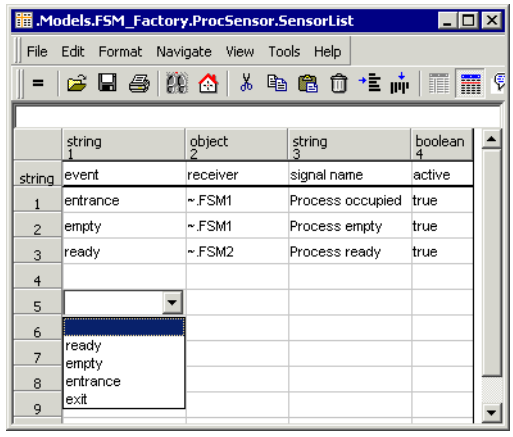
This object is assigned to basic object and can send signals to *FSM*s and/or *ActorCtrl*, as soon as the state of this basic object changes. Sensor control is not firmly assigned to a *FSM*, but it can send signals to several *FSM*.



Sensor on a singleProc *Place_NW*

Object name: This dialog element contains the object name. This name must be a valid designator in the frame of the *FSM*. Read in addition in Plant Simulation manual 'User-Defined Names'.

Sensor List: In this table you specify, by which material flow events signals are sent.



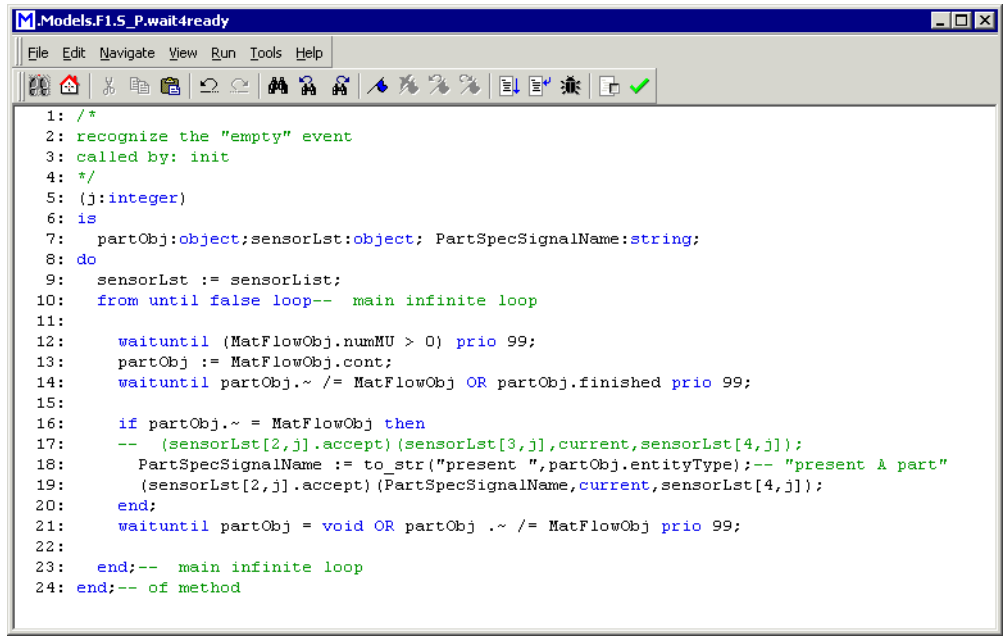
Sensor list of the object *SensorCtrl*

In this table the columns have the following meaning:

1. event: In this columns the event of the movable element on the assigned material flow object is defined. The entries have the following meaning:
   - *empty*: The object became empty.
   - *exit*: A movable element has left the object.
   - *entrance*: A movable element has entered the object.
   - *ready*: The processing of a movable element has finished.
2. receiver: In this column the *FSM* or the object *ActorCtrl* is entered, to which the signal is sent.
3. signal name: Into this column the signal name is entered, which is sent.
4. active: In this column it is adjusted whether the signal is set (value *true*) or reset (value *false*). This setting is used only for persistent signals.

Material flow object: In this dialog field the material flow object is entered, which is watched by the object *SensorCtrl*. The entry can take place also via Drag & Drop of the object on the object *SensorCtrl*. The object *SensorCtrl* positions itself after this action.

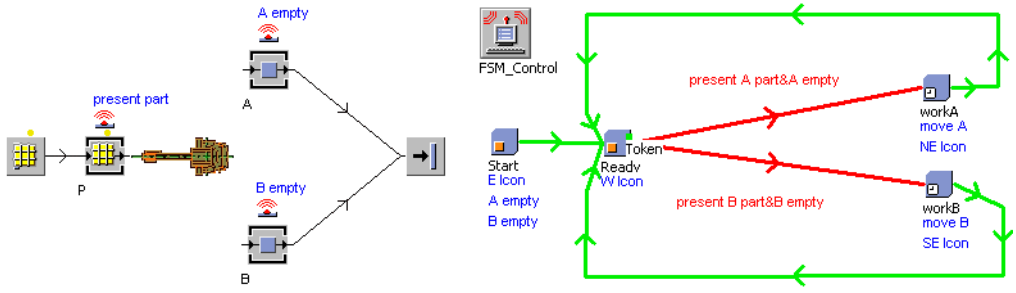
In this section it is described, how the object can sent signals depending on attributes of the movable element. Only experienced simulation specialists should make this extension. The object *SensorCtrl* can send only signals at other *FSM*s and/or *ActorCtrl*. These two objects have a method *accept*, which expects three parameters. The first parameter is the name of the sent signal, which is normally taken from the third column of the sensor list.




```
 1: /*
 2: recognize the "empty" event
 3: called by: init
 4: */
 5: (j:integer)
 6: is
 7:   partObj:object;sensorLst:object; PartSpecSignalName:string;
 8: do
 9:   sensorLst := sensorList;
10:   from until false loop--  main infinite loop
11:
12:     waituntil (MatFlowObj.numMU > 0) prio 99;
13:     partObj := MatFlowObj.cont;
14:     waituntil partObj.~ /= MatFlowObj OR partObj.finished prio 99;
15:
16:     if partObj.~ = MatFlowObj then
17:     --  (sensorLst[2,j].accept)(sensorLst[3,j],current,sensorLst[4,j]);
18:       PartSpecSignalName := to_str("present ",partObj.entityType);-- "present A part"
19:       (sensorLst[2,j].accept)(PartSpecSignalName,current,sensorLst[4,j]);
20:     end;
21:     waituntil partObj = void OR partObj .~ /= MatFlowObj prio 99;
22:
23:   end;--  main infinite loop
24: end;-- of method
```

Extension of the method *wait4Ready* of the sensor control object *S_P*

In the shown method the inheritance is switched off. A new string variable *PartSpecSignalName* is declared, which is the signal name depending on the attribute *entityType* of the movable element. The call of the method *accept* in line 19 is modified. In line 17 you see to the original call of the method *accept*. The sensor at the station *P* in the shown model sends instead of the signal *present part* the signals *present A part* or to *present B part*. Depending on the signal the *FSM* moves the part on the station *A* or *B*.

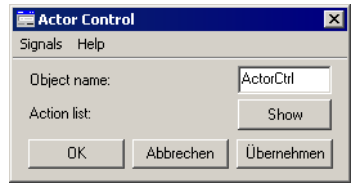


Example of a *FSM*, performing attribute-depending actions.

ActorCtrl

Symbol: 

This object receives signals from *FSM* and converts it in actions. If the *FSM* should accomplish an action, the *FSM* sends a signal to an *ActorCtrl*. In the *ActorCtrl* there must be for each received signal an action.



The action control object *ActorCtrl*

Object name: This dialog element contains the object name. This name must be a valid designator in the frame of the *FSM*. Read in addition in Plant Simulation manual 'User-Defined Names'.

Action list: In this table is defined, what action is done depending on the signal.

	string 1	string 2	string 3	string 4	string 5	string 6
string	signal	action type	parameter 1	parameter 2	parameter 3	parameter 4
8	FSM 1 turns NO	SetIconName	~.FSM1	NO		
9	NW to NO	Move	~.Place_NW	~.Dest_NO		
10	register NO	Method	~.regPart	.Models.FSM_Factor		
11	register SO	Method	~.regPart	.Models.FSM_Factor		
12	register O	Method	~.regPart	.Models.FSM_Factor		
13	W ready	SetIconName	~.FSM1	W		
14	W to Process	Move	~.Place_W	~.Process		
15	FSM 2 turns O	SetIconName	~.FSM2	O		

The action list of the object *ActorCtrl*

In this table the columns have the following meaning:

1. signal: Into this column the signal name is entered, which activates the action.
2. action type: In this column action type is set. The prepared action type are described below.
3. parameter 1– 4: Dependent on the action type you indicate the associated parameters in these columns. These parameters are abbreviate as *P1*, ..., *P4*.

The following action types are prepared:

- *Move:* A movable element is moved from an place-oriented material flow object (parameter *P1*) to a second object (parameter *P2*).
- *LineMove*: A movable element is moved from an material flow object (parameter *P1*) to the end position of a length-oriented material flow object (parameter *P2*).
- *LineBack*: The driving direction of a vehicle is turned around. As parameter *P1* you indicate the location of the vehicle.
- *Remove*: A movable element (parameter *P1*) is deleted.
- *Method*: A method is called, the path is the parameter *P1*. This method must have three *string* parameters, which are given as parameter *P2*, ..., *P4*.
- *SetIconName*: The Icon of an object, which is the parameter *P1*, is switched. The parameter *P2* is the icon name. Optionally you can change the position of the *FSM*. Enter for this the x-position or y-position as parameters *P3* and *P4,* respectively.
- *SetIcon*: The position of an object, which is the parameter *P1*, is set. The parameter *P2* and *P3* are the position. The parameters *P4,* is the rotation angle of the icon.

Experienced simulation specialists can add further actions the object *ActorCtrl*. For this purpose you must do the following steps:

The new action must be added in the column action type of the table *ActionList* in the class of the object. For this purpose you must change the format of the column action type and add here the new action to the existing format string. In the method *getMessage* you must added the new action in the *inspect* instruction. You can orient at that already existing structure. In order to maintain the structure of the object *ActorCtrl*, you insert a new method object for the new action, which is called by the method *getMessage* with the transferred parameter.

Modeling assistance

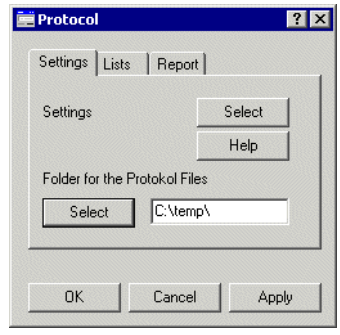
The ideas of the Finite Staete Machine (*FSM*) are simple to understand, but nevertheless the modeling of complex dynamic material flow systems is, like robots, is not simply. The library supports you with the modeling by different tools for the model analysis. Plausibility checks of the tables secure that the correct components are entered. Absolute paths are replaced by relative in tables, so that contents are to be read more simply. Please note that table entries become invalid by renaming objects.

The Protocol object

With the help of the protocol object you can observe the function of the *FSM*. Insert for this purpose the object into the root frame.

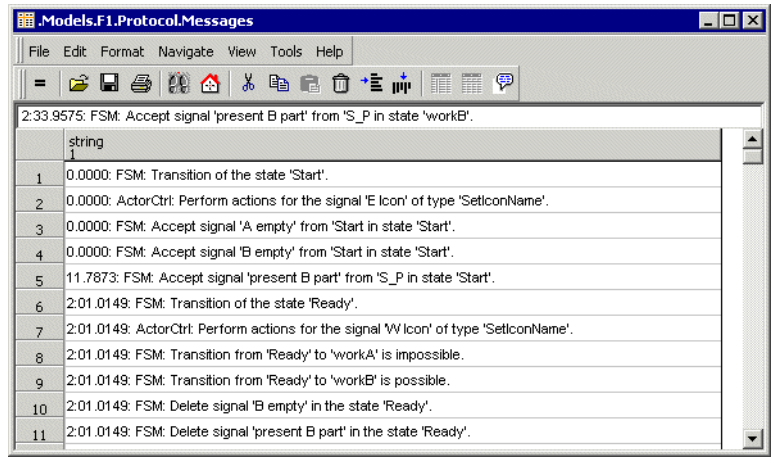
Symbol of the protocol object: 

You want to desplay all actions with simulation time. For this purpose you open the table of the settings and set all values in the row regist *true*.



Einstellungen des Protokollbausteins

In the table Messages, which you can open on the tab Lists, you all changes of states see as well as all tries of the *FSM* to change the state. All performed actions of the object *ActorCtrl* are registered.



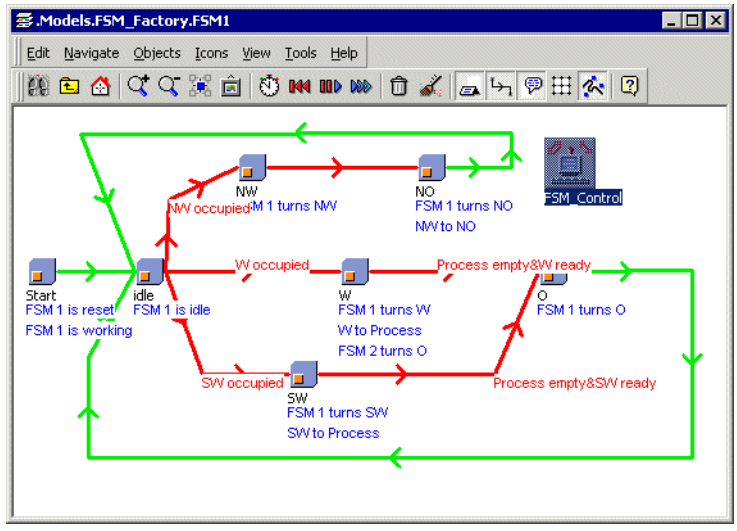
The table of messages of the protocol object.

All tables of the protocol object can open by the context menu.

Animation of signals

A further modeling assistance is the animation of the signals in the model frame, as it is shown in the picture Animation of the signals with special marking of the signals to the object ActorCtrl. A doubleclick on the colored connections opens a table with the corresponding signals, which were exchanged between the objects.

In addition the signals to the states and their transitions can be animated in the frame of the *FSM* . The object *FSM_Control* has corresponding menu and context menu entries Animate signals and Delete animation. The red signals are used in conditions in the connections of the state objects. The blue signals are sent signals, when the state of the *FSM* was changed.



Signals of the *FSM* from the picture States and transitions of the FSM

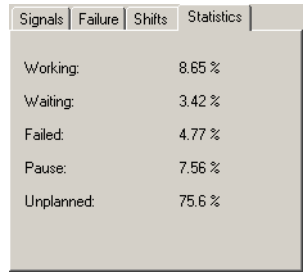
Evaluations

A FSM can be in the following situations:

- *Ungeplannt*: Outside of the shifts, which are specified by the calendar object, is the *FSM* ungeplannt.
- *Pausiert*: During of the shifts pauses occur, which are specified by the calendar object.
- *Failed*: By own failures or by failures of another *FSM* in a protection circle a *FSM* cannot be available.
- *Working*: If the *FSM* contains states with time consumption and if the *Token* is on such a state, then the *FSM* is working during this time. The *Token* is green represented.
- *Wartend*: If the conditions for a state change are not fulfilled, then the *FSM* must wait for the appropriate signals. The *Token* is yellow represented

These time values are represented on the tab Statistics and can be shown also by a diagram object. For this purpose you drag an diagram object onto the *FSM*, of which you would like to illustrate these statistic values. It opens a diagram. By pressing the button Apply of the dialog of the *FSM* control the statistical values in the diagram and on the tab Statistics are updated. This dialog can be opened by the context menu entry Open control.

Note that the actual availabilities can differ from the values, which you entered on the tab Failure, if propagation of failures is desired.



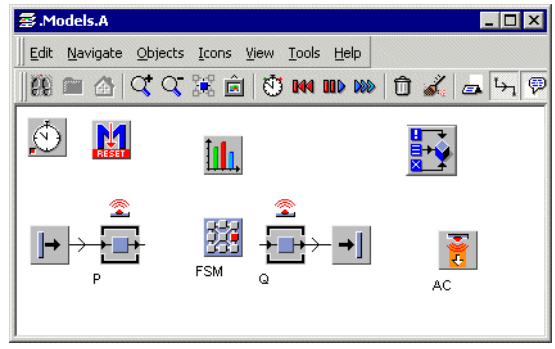
Statistics of the *FSM*

A simple example

In this chapter a simple *FSM* is described, which oscillates between two states. This example shows the principal functionality of a *FSM*.

The Scenario

On the basic object *P* movable elements arrive which can be worked. A robot turns to *P*, takes the part and puts it to the basic object *Q*. The movable element is taken by the robot only, if *Q* is empty. This ensures that the movable element can be put down on *Q*. The movement of the robot from *Q* to *P* needs a negligible time, since the robot does not have to transport anything. The movement of the movable element from *P* to *Q* needs one minute.



A robot takes movable elements from *P* to *Q*.

States

To model a robot we start to recognize all states, which the robot can take. The robot remains for the following two reasons a certain time in a state:

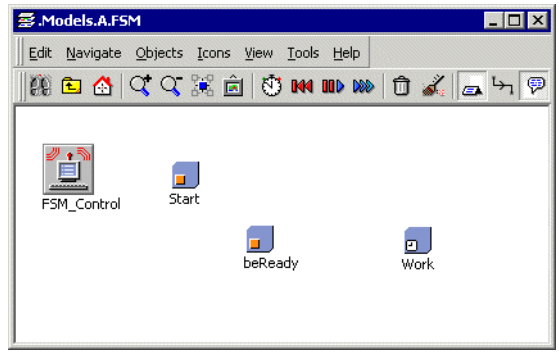
1. The robot is working. In this case we assume that its state does not change, even if the robot is moving during this time.

For the modeling of the work time you use an state object with time consumption. In our example the transport of the part from *P* to *Q* is modelled by a state *Work*.

2. The robot waits for signals, in order to change into the next state. These signals indicate the occurrence to a certain situation.

In our example we need a further state *beReady*, which indicates that the robot can continue working, if the next part arrives on *P*. If *Q* is empty and if the work of the robot is finished, then it goes into the state *beReady*.

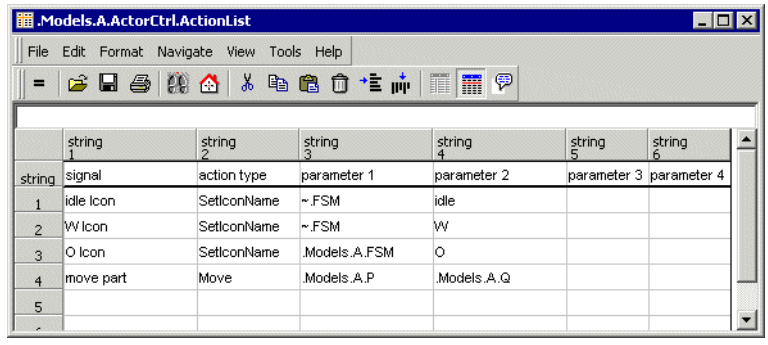
Open the object *FSM* and insert two state objects *beReady* and *Work*. In *Work* you enter the duration of 60 seconds.



The states of the *FSM*

Signals of FSM activate actions

When a state occurs then an corresponding actions are activated. Typically the icon of the *FSM* is switched. For this purpose the *FSM* sends signals to the object *ActorCtrl*. If in our example the state *Work* occurs, then the part from *P* is relocated to *Q*. Insert the object *ActorCtrl* into the root frame and fill out the list of the actions, as follows.

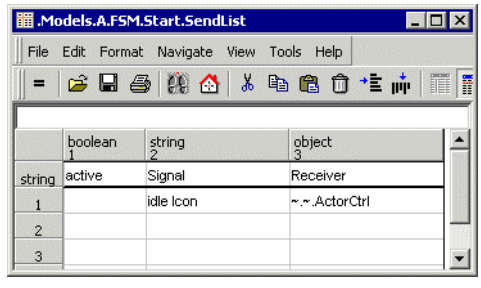


	string 1	string 2	string 3	string 4	string 5	string 6
string	signal	action type	parameter 1	parameter 2	parameter 3	parameter 4
1	idle Icon	SetIconName	~.FSM	idle		
2	W Icon	SetIconName	~.FSM	W		
3	O Icon	SetIconName	.Models.A.FSM	O		
4	move part	Move	.Models.A.P	.Models.A.Q		
5						

List of actions of the object *ActorCtrl*

Object entries can be made by Drag&Drop. After closing and opening of this table relative paths are automatically registered.

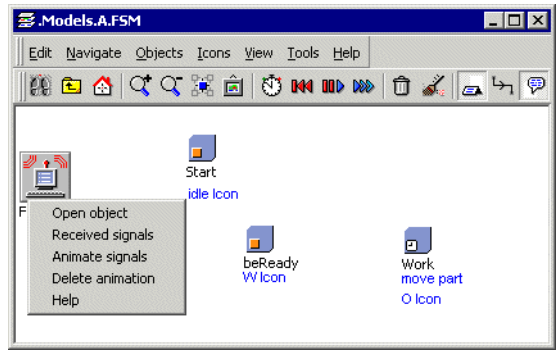
The signals entered in the first column are sent by sate objects. Open the table of the outgoing signals of the state objects. You can use the context menu entry Outgoing signals of the object *State*.



	boolean 1	string 2	object 3
string	active	Signal	Receiver
1		idle Icon	~.~.ActorCtrl
2			
3			

Outgoing signals, which are sent by the state *Start*.

As a check of the entered signals you can desplay the sent signals of the states by the context menu entry Received Signals of the object *FSM_Control*.

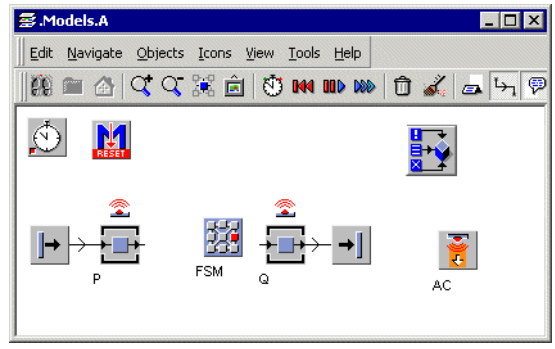


Showing signals, which activated actions.

Material flow objects send signals to the FSM

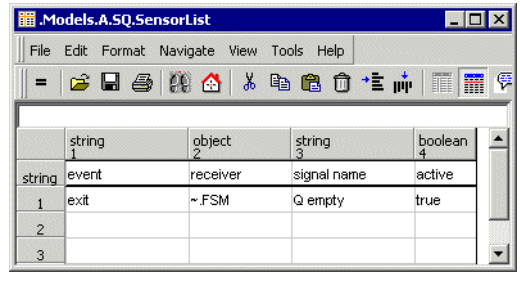
Sensors send signals to *FSM* , which describe certain material flow events.

If a part is ready on *P*, then the *FSM* must become active. You insert an object *SensorCtrl* and drag the object *P* onto the object *SensorCtrl*. The dialog of the object *SensorCtrl* opens. In the sensor list you select *ready* in the first column. Into the second column you drag the *FSM*. After closing and opening this table the relative path ~.*FSM* is registered here. Finally you enter the signal name *P ready* into the third column.



Sensor control on the objects*P* and *Q*

In addition the event must be registered, if *Q* becomes empty. Assign to *Q* a sensor control, which sends the signal *Q empty* to the *FSM*, if *Q* becomes empty.



The sensor list of the sensor control on the singleProc *Q*

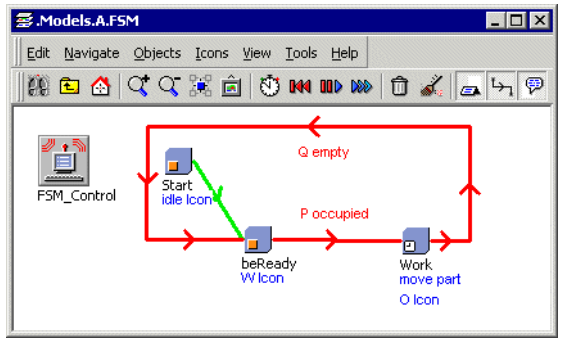
Transitions between states

Only after you defined the states of the *FSM*, you can think about the conditions for transitions between the states. At the beginning of the simulation the *FSM* is in the state *Start*. In our example the *FSM* can change immediately into the state *beReady* at the beginning.

If the *FSM* is in the state *beReady*, then it can change into the state *Work* , if there is a part on *P* and it is ready. This situation is indicated by the signal *P occupied*.

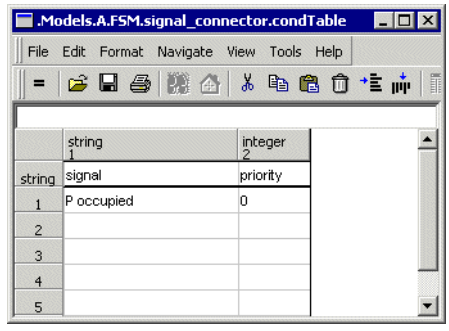
If the FSM is in the state *Work* and if *Q* is empty then it can change into the state *beReady*. This situation is indicated by the signal *Q empty*.

So that the conditions for the transitions between the states are examined, insert connections of the object *signal_connector* between the states. Only transitions between states can occur, which are connected.



Transitions between states with conditions are displayed red.

In order to specify a condition for a transition, you doubleclick a connection. It opens a table, in which you enter the signal, which is necessary for the transition.

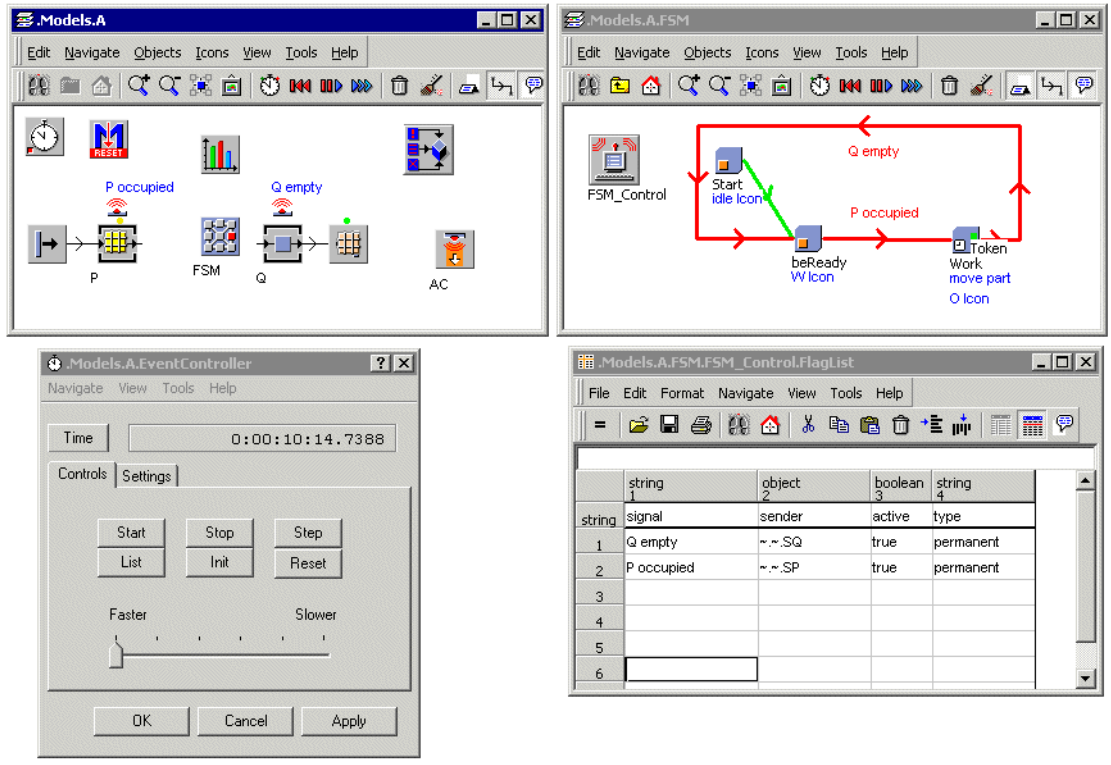


Conditions for the transitions from the state *beReady* to the state *Work*

Validating the function of the FSM

Now you observe the function of the *FSM*. Opening it the *FSM*, in order to see the current state. Doubleclick the object *FSM_Control*. Open the table of the current signals by the button Current signals of the dialog Finite State Machine Control and observe the arrival of signals from material flow events and the disappearance of signals after state changes.

Note that at the end of the duration of the state *Work*, the SingleProc *Q* can be empty or occupied.



Examination of the correct functionality of the *FSM*

In the representation you see the *FSM* in the state *Work*. The duration of this state *Work* yet did not run off, but *Q* is already empty. The transition to the state *beReady* is impossible because of the entered duration of the state *Work*.

It is also possible that *Q* is not empty, but the duration of the state *Work* ran off. In this case the *FSM* announces: 'The transition of *Work* to *beReady* is impossible.' You can see this message in the message list of the protocol object.

About Siemens PLM Software

Siemens PLM Software, a division of Siemens Automation and Drives (A&D), is a leading global provider of product lifecycle management (PLM) software and services with 4.6 million licensed seats and 51,000 customers worldwide. Headquartered in Plano, Texas, Siemens PLM Software's open enterprise solutions enable a world where organizations and their partners collaborate through Global Innovation Networks to deliver world-class products and services. For more information on Siemens PLM Software products and services, visit www.siemens.com/plm.

SIEMENS

Division headquarters

United States
Granite Park One
5800 Granite Parkway
Suite 600
Plano, TX 75024
972 987 3000
Fax 972 987 3398

Regions

Americas
Granite Park One
5800 Granite Parkway
Suite 600
Plano, TX 75024
800 498 5351
Fax 972 987 3398

Europe
Norwich House Knoll Road
Camberley, Surrey
GU15 3SY
United Kingdom
44 1276 702000
Fax 44 1276 705150

Asia-Pacific
Suites 6804-8, 68/F., Central Plaza
18 Harbour Road, WanChai
Hong Kong
852 2230 3333
Fax 852 2230 3210

10/07