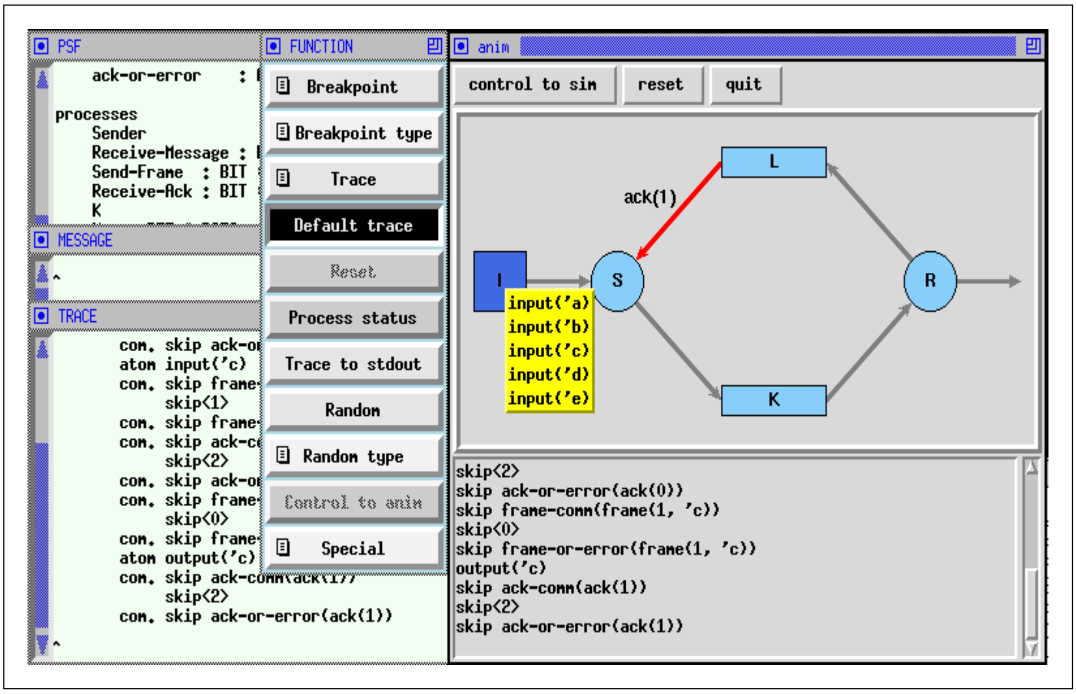
University of Amsterdam
Programming Research Group



Simulation and Animation of Process Algebra Specifications

Bob Diertens

Report P9713 September 1997

University of Amsterdam
Department of Computer Science
Programming Research Group

Simulation and animation of process algebra specifications

Bob Diertens

Report P9713 September 1997

B. Diertens

Programming Research Group
Department of Computer Science
University of Amsterdam

Kruislaan 403
NL-1098 SJ Amsterdam
The Netherlands

tel. +31 20 525 7554
e-mail: bobd@wins.uva.nl

Simulation and animation of process algebra specifications

Bob Diertens

University of Amsterdam
Programming Research Group
e-mail: bobd@wins.uva.nl

ABSTRACT

We present a platform for simulation and animation of process algebra specifications. This platform is built with the use of the ToolBus. To ease the creation of animations, a library of functions has been made. How to use these functions is shown by giving animations for two simple specifications.
The protocol used for interaction between the simulator, animation and the ToolBus is given as a PSF specification. An animation for this specification is also given.

1. Introduction

When simulating a process algebra specification, one easily looses track of the current state of the processes. A visualization of the state seems necessary, especially for larger specifications. We can go even further. By also visualizing the transitions between the states we get an animation of the simulation of our specification.

What do we expect from such an animation? First of all, what our simulator already does, show which actions are performed. We also like to see which processes are active, and how their states are influenced by an action. But above all, we like to see a picture in which we see objects that represent the processes and their connecting communication channels, and in which the formerly mentioned items are visualized.

What was needed for this kind of animation, we did not know. In order to inventorize this, we started by making an animation for a few specifications. First we set up scheme for communication between the simulator and the animation. We used the ToolBus for this with a simple script, and chose Tcl/Tk for the implementation language of the animations. We made an animation for the Alternating Bit Protocol, and used the base of this animation for an animation of a factory.

From our experiments, we identified a bunch of basic functions. We also identified the need to control the simulation through the animation. So we adapted the ToolBus-script, and experimented some more.

At this moment, we had the feeling that we identified all the basic functions that were needed for making an animation for some specification. We implemented a library of these functions, and adapted our animations to make use of this library.

The result is a tool called **simanim**. It actually is a script which controls the execution of the ToolBus, which in turn controls the execution of the simulator and animation.

Overview. In the remainder of this chapter, a short description of PSF, the simulator, and the ToolBus are given. Chapter 2. gives some examples of animation, in chapter 3. some words on the implementation of **simanim** are given, and in chapter 4. we will give a specification in PSF of the interaction between simulator, animation, and the ToolBus.

1.1 PSF

PSF (Process Specification Formalism) is a formal description technique developed for the specification of concurrent systems. A description of PSF can be found in [MauVel90], [MauVel93], [Die94], and [DiePon94].

PSF has been designed as the base for a set of tools to support ACP (Algebra of Communicating Processes) [BerKlo86]. It is very close to the informal syntax normally used in denoting ACP-expressions. The part of PSF that deals with the description of data is based on ASF (Algebraic Specification Formalism) [BerHeeKli89] To meet the modern needs of software engeneering, PSF supports the modular construction of specifications and the parametrization of modules.

1.2 The Simulator

The simulator is part of the PSF-Toolkit. It shows traces of selected items, when it simulates a specification. It is possible to set breakpoints on atoms and processes. The user can choose the actions to perform from a list, but simulation can also be done randomly. The simulator is also provided with a process status, which show the internal status of the simulated terms, and with a history mechanism, that not only makes it possible to go back single steps, but also to jump to a formerly marked state.

1.3 The ToolBus

The ToolBus is a software application architecture developed at the University of Amsterdam by J.A. Bergstra and P. Klint [BerKli95]. It utilizes a scripting language based on process algebra [BaeWeij90] to describe the communication between software tools. A ToolBus script describes a number of processes that can communicate with each other and with tools living outside the ToolBus. A language-dependent adapter that translates between the internal ToolBus data format and the data format used by the individual tools makes it possible to write every tool in the language best suited for the task(s) it has to perform.

2. Animation

To ease the creation of an animation, a library of functions has been made. All handling of input, output, drawing, etc, is done automatically by these functions, which leaves us only with the making of a picture to represent our specification and describing the actions to be performed for the atoms executed by the simulator. A complete description of these functions is given in appendix B..

In the following sections, we give examples on how to make an animation. The first is an animation for the Alternating Bit Protocol, and the second for a small factory. The specification in PSF for these are given in appendix A..

2.1 The Alternating Bit Protocol

First, we have to initialize the windows. The command

```
1 ANIM_windows 440 220 61 10
```

gives us the picture in Figure 2-1. (Line-numbers are there for reference purposes, they are not part of the code.)

We see here three buttons, which are disabled at the moment. Below that a canvas (with width `440` and height `220` in pixels) where the actual animation takes place, and below that a text-window (with width `61` and height `10` in characters) with additional scrollbar. In the text-window, the atoms that are executed by the simulator are displayed (the same as in the TRACE-window of the simulator when tracing is on).

The picture in the canvas is made with the following commands.

```
2 ANIM_create_item recti rect 30 110 20 20 "I"
3 ANIM_create_item ovals oval 120 110 20 20 "S"
4 ANIM_create_item ovalr oval 360 110 20 20 "R"
5 ANIM_create_item rectl rect 240 30 40 10 "L"
6 ANIM_create_item rectk rect 240 190 40 10 "K"
```

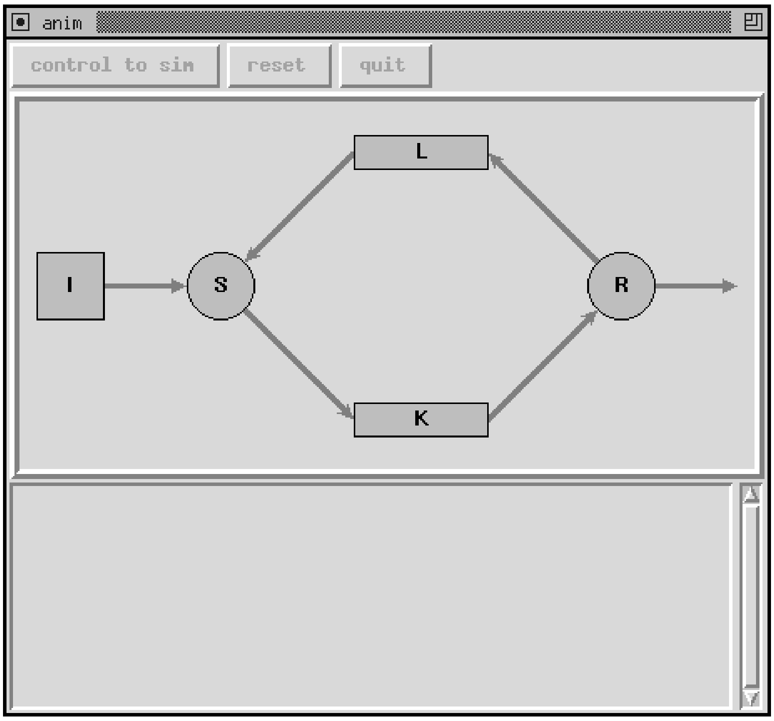


Figure 2-1. screendump of animation window

```
 7 ANIM_create_line toS pos 50 110 item ovals chop -arrow last
 8 ANIM_create_line fromR item ovalr chop pos 430 110 -arrow last

 9 ANIM_create_line StoK item ovals se item rectk w -arrow last
10 ANIM_create_line KtoR item rectk e item ovalr sw -arrow last
11 ANIM_create_line RtoL item ovalr nw item rectl e -arrow last
12 ANIM_create_line LtoS item rectl w item ovals ne -arrow last
```

The command in line 2 creates a rectangle (indicated by the second argument `rect`) at the position `30,110` with width and height both `20`. The actual width and height are twice these sizes. The sizes given here indicate the distance from the position `30,110` to the border of the rectangle. (It is done this way to eliminate rounding of numbers in calculations.)
The first argument is the name of the rectangle, so that it can be referenced later, and the last argument gives the text to be displayed in the item.

The command in line 7 creates a line with name `toS` from position `50,110` (`pos 50 110`) to the border of the item with name `ovals` (`item ovals chop`). And at the end of the line, an arrow is drawn (`-arrow last`).

To display text at some positions later on, we do the following.

```
13 ANIM_textpos_line toS toS s
14 ANIM_textpos_line fromR fromR s
15 ANIM_textpos_line StoK StoK ne
16 ANIM_textpos_item atK rectk s n
17 ANIM_textpos_line KtoR KtoR nw
18 ANIM_textpos_line RtoL RtoL sw
19 ANIM_textpos_item atL rectl n s
20 ANIM_textpos_line LtoS LtoS se
```

The command in line 13 defines a position for text with the name `toS` (the first argument) at line `toS` (the second argument) and with anchor `s` (south), which means that the south of the text will be placed just above the line. The command in line 16 defines a position with name `atK` at the south of item `rectk` with anchor `n` (north).

2.1.1 Passive animation

Now we describe the interpretations for the atoms in the trace of the simulator. We do this by defining the function `ANIM_action` as follows.

```
21 proc ANIM_action {atom} {
```

```
22    if {[regexp {^input\('(.*)\)$} $atom match arg1]} {
23       ANIM_clear recti
24       ANIM_clear ovals
25       ANIM_create_text toS "$arg1"
26       ANIM_activate_line toS
27       ANIM_add_clear ovals {line toS} {text toS}
28    } elseif {[regexp {^skip frame-comm\(frame\((.*), '(.*)\)\)$} \
29          $atom match arg1 arg2]} {
30       ANIM_clear ovals
31       ANIM_create_text StoK "$arg2 ($arg1)"
32       ANIM_activate_line StoK
33       ANIM_add_clear rectk {line StoK} {text StoK}
34    } elseif {[regexp {^skip<(0|1)>$} $atom match]} {
35       ANIM_clear rectk
36       ANIM_create_text atK "$match"
37       ANIM_add_clear rectk {text atK}
38    } elseif {[regexp {^skip frame-or-error\(frame\((.*), '(.*)\)\)$} \
39          $atom match arg1 arg2]} {
40       ANIM_clear rectk
41       ANIM_create_text KtoR "$arg2 ($arg1)"
42       ANIM_activate_line KtoR
43       ANIM_add_clear ovalr {line KtoR} {text KtoR}
44    } elseif {[regexp {^skip frame-or-error\(frame-error\)$} $atom \
45          match]} {
46       ANIM_clear rectk
47       ANIM_create_text KtoR "error"
48       ANIM_activate_line KtoR
49       ANIM_add_clear ovalr {line KtoR} {text KtoR}
50    } elseif {[regexp {^output\('(.*)\)$} $atom match arg1]} {
51       ANIM_clear ovalr
52       ANIM_create_text fromR "$arg1"
53       ANIM_activate_line fromR
54       ANIM_add_clear ovalr {line fromR} {text fromR}
55    } elseif {[regexp {^skip ack-comm\(ack\((.*)\)\)$} $atom match \
56          arg1]} {
57       ANIM_clear ovalr
58       ANIM_create_text RtoL "ack($arg1)"
59       ANIM_activate_line RtoL
60       ANIM_add_clear rectl {line RtoL} {text RtoL}
61    } elseif {[regexp {^skip<(2|3)>$} $atom match]} {
62       ANIM_clear rectl
63       ANIM_create_text atL "$match"
64       ANIM_add_clear rectl {text atL}
65    } elseif {[regexp {^skip ack-or-error\(ack\((.*)\)\)$} $atom \
66          match arg1]} {
67       ANIM_clear rectl
68       ANIM_create_text LtoS "ack($arg1)"
69       ANIM_activate_line LtoS
70       ANIM_add_clear recti {line LtoS} {text LtoS}
71    } elseif {[regexp {^skip ack-or-error\(ack-error\)$} $atom \
72          match]} {
73       ANIM_clear rectl
74       ANIM_create_text LtoS "error"
75       ANIM_activate_line LtoS
76       ANIM_add_clear ovals {line LtoS} {text LtoS}
77    }
78 }
```

We take line 22 as an example of how an atom can be matched. First note that in Tcl the value of a variable with the name *var* is substituted for $*var*.

We first explain the regular expression `^input\('(.*)\)$`. The ^ and $ match with the begin and end of the atom in *atom*, so that we match all of *atom* and not just a part of it. The `\(` and `\)` match with a `(` and a `)` respectively. We use `.*` to match with anything and we put it in between `( )` to save the part it matched (this becomes available in the variable with name *arg1*). The other characters match with themselves. The variable with name *match* will contain everything that has been matched.

So in case the atom is `input('a)` the regular expression will match and variable *arg1* gets the value `a`.

In line 25 we create a text (the value of `arg1`) on the position `toS` created earlier with the use of `ANIM_textpos_line`. The line `toS` is activated in line 26 (on color displays it gets a different color and on monochrome displays it becomes solid).

In line 27 we add the line `toS` and the text `toS` to the clear-list of `ovals`. With the next match of an atom (line 28) we give the order to clear this list for `ovals` (line 10).

Instead of line 27 and 30 we also could have done

```
ANIM_deactivate_line toS
ANIM_delete_text toS
```

directly after line 29.

Now let us look at the result of this. After the simulation of the atoms

input('a)
skip frame-comm(frame(0, 'a))

we get the picture in Figure 2-2.

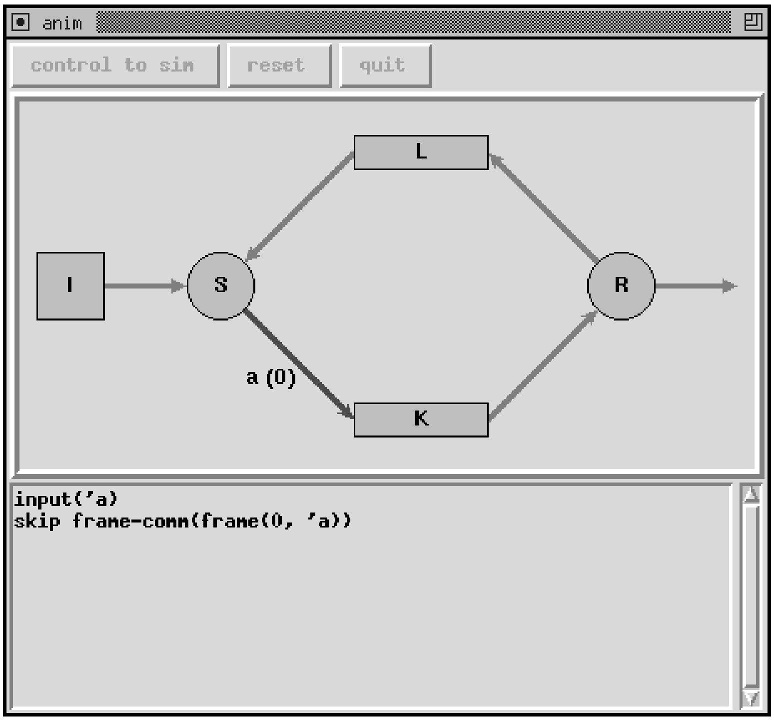


Figure 2-2. alternating bit protocol: passive animation

2.1.2 Active animation

It is also possible to let the animation control the simulation. For this, we have to define a function `ANIM_choose`. When this function is defined, the animation automatically takes control. The buttons [control to sim], [reset], and [quit] are enabled. The first one gives control to the simulator, what gives us passive animation. The simulator then has a button [control to anim] enabled to give control back to the animation. The buttons [reset] and [quit] behave the same as in the simulator.

```
proc ANIM_choose {atom} {
   if {[regexp {^input\('(.*)\)$} $atom match arg1]} {
      ANIM_add_list recti $match
   } elseif {[regexp {^skip frame-comm\(frame\((.*), '(.*)\)\)$} \
         $atom match arg1 arg2]} {
      ANIM_add_list ovals $match
   } elseif {[regexp {^skip<(0|1)>$} $atom match]} {
      ANIM_add_list rectk $match
   } elseif {[regexp {^skip frame-or-error\(frame\((.*), '(.*)\)\)$} \
         $atom match arg1 arg2]} {
      ANIM_add_list rectk $match
   } elseif {[regexp {^skip frame-or-error\(frame-error\)$} $atom \
         match]} {
      ANIM_add_list rectk $match
   } elseif {[regexp {^output\('(.*)\)$} $atom match arg1]} {
      ANIM_add_list ovalr $match
   } elseif {[regexp {^skip ack-comm\(ack\((.*)\)\)$} $atom match \
         arg1]} {
      ANIM_add_list ovalr $match
   } elseif {[regexp {^skip<(2|3)>$} $atom match]} {
      ANIM_add_list rectl $match
   } elseif {[regexp {^skip ack-or-error\(ack\((.*)\)\)$} $atom \
         match arg1]} {
      ANIM_add_list rectl $match
   } elseif {[regexp {^skip ack-or-error\(ack-error\)$} $atom \
         match]} {
```

```
      ANIM_add_list rectl $match
    }
}
```

For each atom in the choose-list of the simulator the above function is called. Each item in the animation has its own choose-list. When there are atoms added to a list with the use of `ANIM_add_list`, the item becomes activated (on color displays it gets a different color and on monochrome displays it becomes stippled). When an activated item is clicked upon with the mouse, a list pops up from which an atom can be selected for execution. Leaving the list with the mouse makes the list disappear. So the lists can be examined without making a selection.

An snapshot of active animation is shown in Figure 2-3.

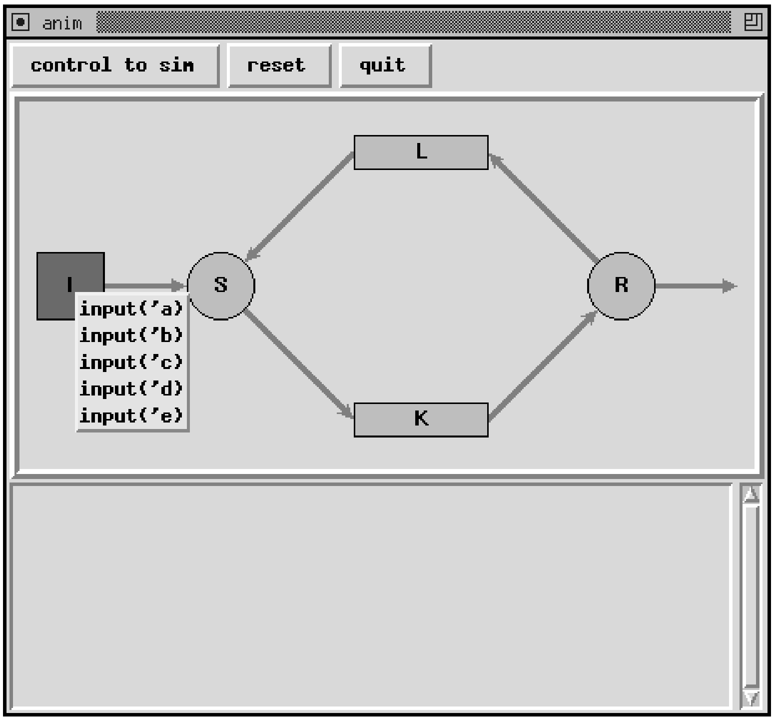


Figure 2-3. alternating bit protocol: active animation

2.2 A small factory

The animation functions shown sofar, are satisfactory for displaying processes and their communications. However, more can be done to make the animations more attractive, such as moving items, queues, display counters on an information panel, etc.

Here, a few features are shown of which the ones mentioned above are the most important. For this, we use a small factory consisting of input, output, some stations and conveyor belts. It produces the products A and B which take slightly different routes through the factory.

We first give the commands for the picture in the canvas of the animation.

```
ANIM_windows 340 200 30 10

ANIM_create_item inp rect 30 30 15 15 "I"
ANIM_create_item s1 rect 30 100 15 15 "1"
ANIM_create_item s2 rect 100 100 15 15 "2"
ANIM_create_item s3 rect 170 100 15 15 "3"
ANIM_create_item s4 rect 240 100 15 15 "4"
ANIM_create_item s5 rect 240 170 15 15 "5"
ANIM_create_item s6 rect 310 170 15 15 "6"
ANIM_create_item out rect 310 100 15 15 "O"

ANIM_create_line ins1 item inp s item s1 n -arrow last
ANIM_textpos_line ins1 ins1 e
ANIM_create_line outs6 item s6 n item out s -arrow last
```

```
13 ANIM_textpos_line outs6 outs6 w

14 ANIM_create_line s1s2 item s1 e item s2 w -width 15
15 ANIM_create_line s2s3 item s2 e item s3 w -width 15
16 ANIM_create_line s3s4 item s3 e item s4 w -width 15
17 ANIM_create_line s3s5 item s3 s pos [ANIM_dim s3 x] [ANIM_dim s5 y] \
18    item s5 w -width 15
19 ANIM_create_line s4s5 item s4 s item s5 n -width 15
20 ANIM_create_line s5s6 item s5 e item s6 w -width 15
```

This gives us the picture in Figure 2-4.

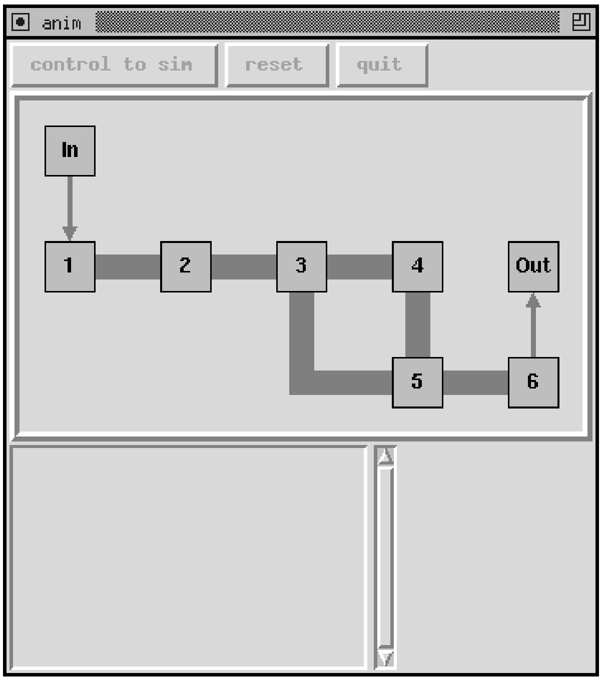


Figure 2-4. factory

In line 17, we see the use of function `ANIM_dim`. It is used the get a dimension from its first argument (here, the x-coordinate of item `s3` and the y-coordinate of item `s5`). The square brackets around it are to let Tcl/Tk know it has to call the function. It is also possible to do more calculations, for example with the use of the Tcl/Tk function `expr` like this

```
[expr [ANIM_dim s3 x] * 2 + 5]
```

which takes the x-coordinate of `s3`, multiplies it by 2 and adds 5 to it.

2.2.1 Moving items

Instead of showing that a product is moved from one station to another by means of an arrow and some text, we actually want to see it moving over the conveyor belt.

We define the function `ANIM_action` as follows.

```
21 proc ANIM_action {atom} {
22    if {[regexp {^input\((.*)\)$} $atom match arg1]} {
23       ANIM_create_text ins1 "$arg1"
24       ANIM_activate_line ins1
25    } elseif {[regexp {^comm-input\((.*)\)$} $atom match arg1]} {
26       ANIM_delete_text ins1
27       ANIM_deactivate_line ins1
28       ANIM_create_item AT1 rect [ANIM_dim s1 x] [ANIM_dim s1 y] \
29          7 7 "$arg1" -free -color 1
30    } elseif {[regexp {^comm-belt\(3, 4, .*\)$} $atom match arg1]} {
31       ANIM_move AT3 rightto [ANIM_dim s4 x] -newid AT4
32    } elseif {[regexp {^comm-belt\(3, 5, .*\)$} $atom match arg1]} {
33       ANIM_move AT3 downto [ANIM_dim s5 y] rightto [ANIM_dim s5 x] \
34          -newid AT5
35    } elseif {[regexp {^comm-belt\(4, 5, .*\)$} $atom match arg1]} {
36       ANIM_move AT4 downto [ANIM_dim s5 y] -newid AT5
```

```
37    } elseif {[regexp {^comm-belt\((.*), (.*), .*\)$} $atom match \
38          arg1 arg2]} {
39       ANIM_move AT$arg1 rightto [ANIM_dim s$arg2 x] -newid AT$arg2
40    } elseif {[regexp {^comm-output\((.*)\)$} $atom match arg1]} {
41       ANIM_destroy_item AT6
42       ANIM_create_text outs6 "$arg1"
43       ANIM_activate_line outs6
44    } elseif {[regexp {^output\((.*)\)$} $atom match arg1]} {
45       ANIM_delete_text outs6
46       ANIM_deactivate_line outs6
47    }
48 }
```

Line 31 shows how we move a product from station 3 to station 4. With the option `-newid` we give it a new name. In this way, we do not have to keep track of which item is at what position (the name of the item indicates its location).

In lines 28 and 29, items are created with the options `-free` and `-color`. The option `-free` indicates that this item has to be freed (destroyed) on a reset. The option `-color` *x* indicates that the color for the item must come from colorset *x*. Where *x* may be either 0 or 1, or a colorset created with the function `ANIM_colorset`.

A snapshot of this passive animation is shown in Figure 2-5

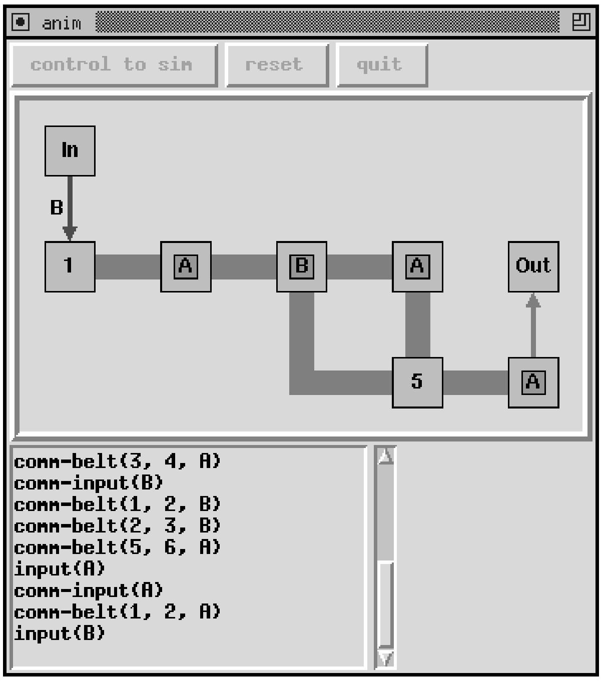


Figure 2-5. factory: passive animation

The function for active animation is given below.

```
proc ANIM_choose {atom} {
   if {[regexp {^input\((.*)\)$} $atom match arg1]} {
      ANIM_add_list inp $match
   } elseif {[regexp {^comm-input\((.*)\)$} $atom match arg1]} {
      ANIM_add_list s1 $match
   } elseif {[regexp {^comm-belt\((.*), (.*), .*\)$} $atom match \
         arg1 arg2]} {
      ANIM_add_list AT$arg1 $match
   } elseif {[regexp {^comm-output\((.*)\)$} $atom match arg1]} {
      ANIM_add_list s6 $match
   } elseif {[regexp {^output\((.*)\)$} $atom match arg1]} {
      ANIM_add_list out $match
   }
}
```

2.2.2 *Queues*

Now, we extend our specification of the factory with input- and output-queues.

In the animation, we replace line 2 with

```
ANIM_create_queue qin 25 30 13 1 -anchor w
```

and line 9 with

```
ANIM_create_queue qout 310 115 1 7 -orient vertical -anchor s
```

This gives us a horizontal input-queue of `13` characters long and `1` character high, at position `25,30.` By using the option `-orient vertical` a vertical output-queue is created.

This is enough for passive animation. However, for active animation we need an item on both sides of the queue in order to control the input and output of the queue. We now replace line 2 with

```
ANIM_create_item qin-out rect 22 30 7 15 ""
ANIM_create_queue qin [ANIM_dim qin-out e,x] 30 10 1 -anchor w
ANIM_create_item qin-in rect [expr [ANIM_dimq qin e,x] + 7] 30 7 15 "In"
```

and line 9 with

```
ANIM_create_item qout-in rect [ANIM_dim s6 x] 107 12 8 ""
ANIM_create_queue qout [ANIM_dim qout-in x] [ANIM_dim qout-in n,y] 1 5 \
   -orient vertical -anchor s
ANIM_create_item qout-out rect [ANIM_dimq qout x] \
   [expr [ANIM_dimq qout n,y] - 8] 12 8 "Out"
```

The code for passive and active animation is given below

```
 63 proc ANIM_action {atom} {
 64    if {[regexp {^q-input\((.*)\)$} $atom match arg1]} {
 65       ANIM_add_queue qin $arg1
 66    } elseif {[regexp {^comm-q-input\((.*)\)$} $atom match arg1]} {
 67       ANIM_sub_queue qin
 68       ANIM_create_text ins1 $arg1
 69       ANIM_activate_line ins1
 70    } elseif {[regexp {^comm-input\((.*)\)$} $atom match arg1]} {
 71       ANIM_delete_text ins1
 72       ANIM_deactivate_line ins1
 73       ANIM_create_item AT1 rect [ANIM_dim s1 x] [ANIM_dim s1 y] \
 74       7 7 "$arg1" -free -color 1
 75    } elseif {[regexp {^comm-belt\(3, 4, .*\)$} $atom match arg1]} {
 76       ANIM_move AT3 rightto [ANIM_dim s4 x] -newid AT4
 77    } elseif {[regexp {^comm-belt\(3, 5, .*\)$} $atom match arg1]} {
 78       ANIM_move AT3 downto [ANIM_dim s5 y] rightto [ANIM_dim s5 x] \
 79          -newid AT5
 80    } elseif {[regexp {^comm-belt\(4, 5, .*\)$} $atom match arg1]} {
 81       ANIM_move AT4 downto [ANIM_dim s5 y] -newid AT5
 82    } elseif {[regexp {^comm-belt\((.*), (.*), .*\)$} $atom match \
 83          arg1 arg2]} {
 84       ANIM_move AT$arg1 rightto [ANIM_dim s$arg2 x] -newid AT$arg2
 85    } elseif {[regexp {^comm-output\((.*)\)$} $atom match arg1]} {
 86       ANIM_destroy_item AT6
 87       ANIM_create_text outs6 "$arg1"
 88       ANIM_activate_line outs6
 89    } elseif {[regexp {^comm-q-output\((.*)\)$} $atom match arg1]} {
 90       ANIM_delete_text outs6
 91       ANIM_deactivate_line outs6
 92       ANIM_add_queue qout $arg1
 93    } elseif {[regexp {^q-output\((.*)\)$} $atom match arg1]} {
 94       ANIM_sub_queue qout
 95    }
 96 }

 97 proc ANIM_choose {atom} {
 98    if {[regexp {^q-input\((.*)\)$} $atom match arg1]} {
 99       ANIM_add_list qin-in $match
100    } elseif {[regexp {^comm-q-input\((.*)\)$} $atom match arg1]} {
101       ANIM_add_list qin-out $match
102    } elseif {[regexp {^comm-input\((.*)\)$} $atom match arg1]} {
103       ANIM_add_list s1 $match
104    } elseif {[regexp {^comm-belt\((.*), (.*), .*\)$} $atom match arg1 arg2]} {
105       ANIM_add_list AT$arg1 $match
106    } elseif {[regexp {^comm-output\((.*)\)$} $atom match arg1]} {
107       ANIM_add_list s6 $match
108    } elseif {[regexp {^comm-q-output\((.*)\)$} $atom match arg1]} {
109       ANIM_add_list qout-in $match
110    } elseif {[regexp {^q-output\((.*)\)$} $atom match arg1]} {
111       ANIM_add_list qout-out $match
112    }
113 }
```

A snapshot of this is shown in Figure 2-6.

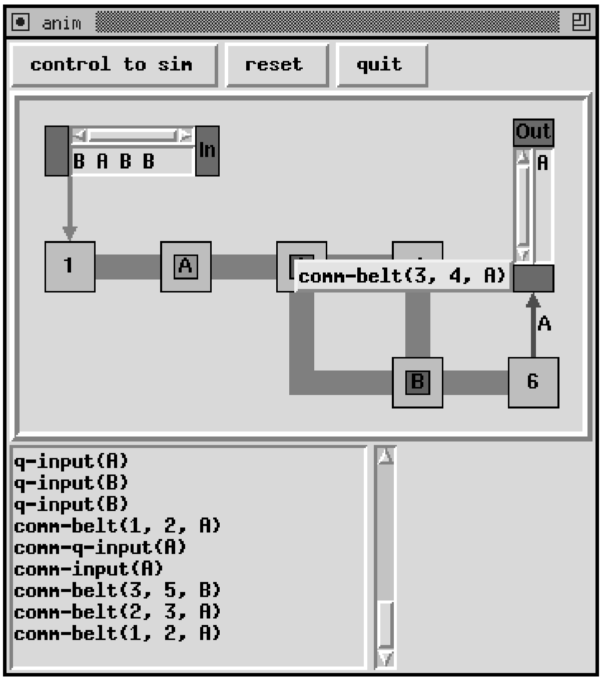


Figure 2-6. factory with queues: active animation

2.2.3 Information panel

In order to get an even better view, support for accounting is added. If we want to display the lengths of the queues and the amount of input and output of the factory, we can add the following code.

```
114 ANIM_create_box info queues -side top -ipadx 1 -ipady 1 -expand -bw 2 \
115    -relief ridge
116 ANIM_create_box queues queueinput -side left
117 ANIM_create_label queueinput inputtext "queue In" -width 9 -anchor w
118 ANIM_create_label queueinput inputvar q-input -var -bw 2 \
119    -relief sunken -width 2
120 ANIM_create_box queues queueoutput -side left
121 ANIM_create_label queueoutput outputtext "queue Out" -width 9 -anchor w
122 ANIM_create_label queueoutput outputvar q-output -var -bw 2 \
123    -relief sunken -width 2

124 ANIM_init_var q-input 0
125 ANIM_init_var q-output 0

126 ANIM_create_box info table -side top -bw 2 -relief ridge
127 ANIM_create_box table header -side left
128 ANIM_create_label header col0 "" -width 6
129 ANIM_create_label header col1 "A" -width 2
130 ANIM_create_label header col2 "B" -width 2
131 ANIM_create_box table row1 -side left
132 ANIM_create_label row1 input input -width 6 -anchor w
133 ANIM_create_label row1 inpA input(A) -var -width 2 -bw 2 \
134    -relief sunken
135 ANIM_create_label row1 inpB input(B) -var -width 2 -bw 2 \
136    -relief sunken
137 ANIM_create_box table row2 -side left
138 ANIM_create_label row2 output output -width 6 -anchor w
139 ANIM_create_label row2 outpA output(A) -var -width 2 -bw 2 \
140    -relief sunken
141 ANIM_create_label row2 outpB output(B) -var -width 2 -bw 2 \
142    -relief sunken

143 ANIM_init_array input [list A 0 B 0]
144 ANIM_init_array output [list A 0 B 0]
```

At line 114, a box is created with the name `queues` and parent `info`. Box `info` is predefined and is normally empty. See appendix B. for an explanation of the options. In that box we create the boxes `queueinput` and `queueoutput`. In box `queueinput` we create two labels, one which contains text and one which will contain the last value assigned to variable `q-input` (this is indicated with the option `-var`).

Variables must be initialized with the use of function `ANIM_init_var,` in order to initialize them again after a reset.

In box `info` also a box `table` is made. In this box we display the arrays `input` and `output`, which must be initialized with function `ANIM_init_array`.

Now, in the function `ANIM_action` one can assign values to these variables with either the **set** or the **incr** command of Tcl. We insert after line 65 the commands

```
incr q-input
incr input($arg1)
```

after line 67

```
incr q-input -1
```

after line 92

```
incr q-output
```

and after line 94

```
incr q-output -1
incr output($arg1)
```

Unfortunately, in Tcl these variables must be declared to be global in the function `ANIM_action`. We do this by inserting

```
global q-input q-output input output
```

after line 63.

How this all looks like can be seen in Figure 2-7.

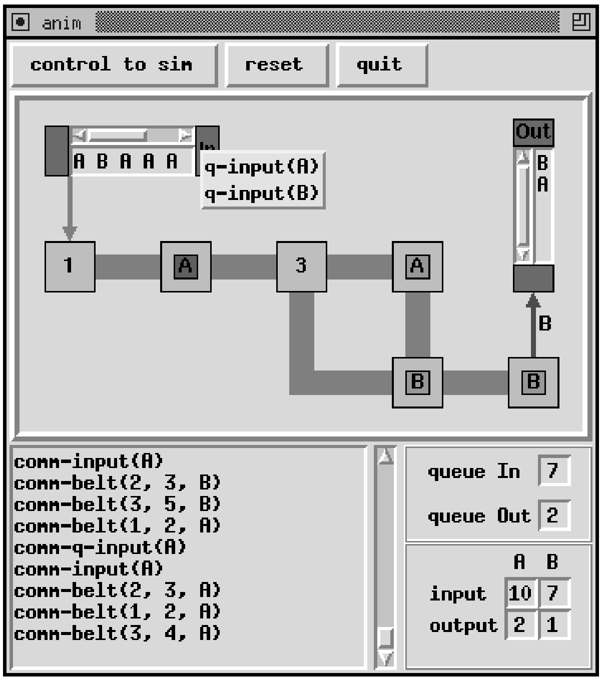


Figure 2-7. factory with info-panel: active animation

2.3 Adding Tcl/Tk code

When the given features do not provide the required functionality, it is always possible to write some additional code. However, care should be taken not to break up the functionality of the existing code. The animation routines only use names starting with `ANIM` and `anim`, and window-paths starting with `.anim`. It is best not to use such names and paths.

3. Specification of simanim

We shall describe simanim with the use of a specification in PSF. Of course, we used simanim to develop this specification. The animation can be found in appendix D..

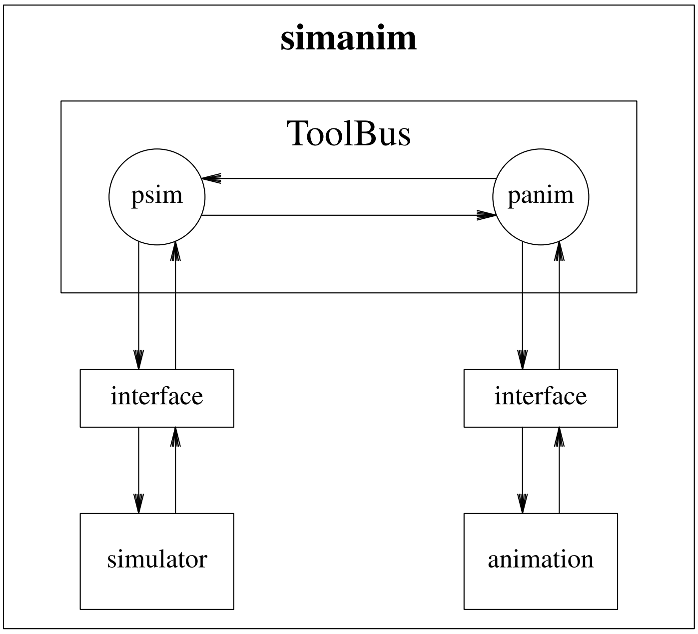


Figure 3-1. processes and communications in simanim

From Figure 3-1, we can distinguish three levels of communication. Between the tools, between the interfaces, and between the processes in the ToolBus. We shall describe each level separately, starting with the tools.

3.1 Level 1: The tools

3.1.1 Description of the simulator

On startup, the simulator first receives information about which tool will be in control and if the other tool is capable of being in control.

When the simulator is in control, it receives the order to send a message. This message can either be an atom that is executed, or the pushing of the buttons reset or quit. After which the simulator must receive an acknowledge before going on. When the animation is capable of being in control, it may also be the pushing of the button control to anim.

When the animation is in control, it receives the order to send a message, which must be the list of atoms that can be executed at that moment. After which the simulator receives a message which atom is selected or a reset, a quit, or the order to take control.
If the end of simulation of the specification is reached, a messages that states that the end is reached must be send, instead of a list of atoms. After which the simulator must receive an acknowledge.

3.1.2 Description of the animation

On startup, the animation first receives the request to send information on which tool should take control and if that tool must keep control.

The animation may now receive an atom that is executed by the simulator, a reset, a list of atoms from which a choice must be made, the message that states that the end of simulation of the specification is reached, or the message that the animation must take control.

Note: it is not necessary that the animation receives a quit from the simulator, because the animation is stopped by the ToolBus in that case (see section 3.3).

When the message is a list of atoms, the animation may send an atom chosen from the list, the pushing of the buttons [reset], [quit], or [control to sim].
When it is a end-message, it may only send the pushing of the buttons [reset], or [quit].
In the other cases, it sends either an acknowledge or an error, depending on the outcome of the actions of the animation.

3.1.3 Data modules for the tools

The module `Atoms` gives us the atoms we can simulate. We use these instead of a specification, since we are only interested in the communications between the tools.

```
data module Atoms
begin
    exports
    begin
        sorts
            ATOM
        functions
            f : -> ATOM
            g : -> ATOM
            h : -> ATOM
    end
end Atoms
```

The following two modules give the types and identifiers for the tools to use.

```
data module Tool-Types
begin
    exports
    begin
        sorts
            Tterm
    end
end Tool-Types

data module Tool-ID
begin
    exports
    begin
        functions
            sim : -> Tterm
            anim : -> Tterm
    end
    imports
        Tool-Types
end Tool-ID
```

The functions given in the module `Tool-Messages`, represent the messages send and received by the tools. We use a single module for this instead of a module for each tool, in which we specify the type of the messages and the messages for that particular tool. In this way, we save a lot of specifying, and it keeps us focused on the interaction between the tools.

```
data module Tool-Messages
begin
    exports
    begin
        functions
            control-info : -> Tterm
            control : Tterm # BOOLEAN -> Tterm
            control : Tterm -> Tterm

            send-message : -> Tterm

            reset : -> Tterm
            quit : -> Tterm
            controltoanim : -> Tterm
            controltosim : -> Tterm
            choose : Tterm -> Tterm
            choice : Tterm -> Tterm
            end-of-spec : -> Tterm

            givecontrol : -> Tterm
```

```
        takecontrol : -> Tterm

        ack : -> Tterm
        error : -> Tterm
    end
    imports
        Booleans,
        Tool-Types
end Tool-Messages
```

The module Booleans that is imported by module Tool-Messages, is taken from the library that comes with the PSF-Toolkit.

The module Tool-data specifies the functions needed for data-manipulation by the tools.

```
data module Tool-data
begin
    exports
    begin
        functions
            atom : ATOM -> Tterm
            control-tool : Tterm -> Tterm
            control-keep : Tterm -> BOOLEAN
            _|_ : Tterm # Tterm -> Tterm
    end
    imports
        Atoms,
        Tool-ID,
        Tool-Messages
    variables
        x : -> Tterm
        y : -> BOOLEAN
    equations
    [1] control-tool(control(x, y)) = x
    [2] control-keep(control(x, y)) = y
end Tool-data
```

3.1.4 Specification of the simulator

The specification of the simulator follows from the description given in section 3.1.1. We use the atom sim for actions inside the simulator and the atoms sim-snd and sim-rec for communication with the outside world. The variable control is used for denoting who is in control (the simulator or the animation), and the variable keep-control to denote if the control may be given to the other tool.

```
process module Simulator
begin
    exports
    begin
        atoms
            sim-snd : Tterm
            sim-rec : Tterm
        processes
            Simulator
    end
    imports
        Tool-data
    atoms
        sim : Tterm
    processes
        Run : Tterm # BOOLEAN
    sets
        of ATOM
            A = { f, g, h }
        of Tterm
            Control-set = { control(c, k) | c in Tterm, k in BOOLEAN }
        of Tterm
            ATOM-set = { atom(a) | a in ATOM }
    variables
        control : -> Tterm
        keep-control : -> BOOLEAN
    definitions
        Simulator = sum(c in Control-set,
                        sim-rec(c) . sim(c) .
                        Run(control-tool(c), control-keep(c))
                    )
        Run(control, keep-control) =
                    [ control = sim ] -> (
                        sim-rec(send-message) . (
```

```
                     (
                        sum(a in A, sim(atom(a)) . sim-snd(atom(a)))
                     +  sim(reset) . sim-snd(reset)
                     +  sim(quit) . sim-snd(quit)
                     ) . sim-rec(ack) . sim(ack) .
                     Run(control, keep-control)
                  +  [ keep-control = false ] -> (
                        sim(controltoanim) . sim-snd(givecontrol) .
                        Run(anim, keep-control)
                     )
                  )
               )
            +  [ control = anim ] -> (
                  sim-rec(send-message) . (
                     sim-snd(choose(atom(f) | atom(g) | atom(h)))
                  +  sim-snd(end-of-spec) . sim-rec(ack) . sim(ack)
                  ) . (
                     (
                        sum(a in ATOM-set, sim-rec(a) . sim(a))
                     +  sim-rec(reset) . sim(reset)
                     +  sim-rec(quit) . sim(quit) . sim-snd(quit)
                     ) . Run(control, keep-control)
                  +  sim-rec(takecontrol) . sim(takecontrol) .
                     Run(sim, keep-control)
                  )
               )
end Simulator
```

3.1.5 Specification of the animation

The specification of the animation follows from the description 3.1.2. Simular to module `Simulator`, we use an atom `anim` for action inside the animation, and the atoms `anim-snd` and `anim-rec` for communication with the outside world. Also, the use of the variables `control` and `keep-control` is the same as in module `Simulator`.

The process `Choose` is used to transform the list of possible atoms, received from the simulator, into a list of alternatives.

```
process module Animation
begin
   exports
   begin
      atoms
         anim-rec : Tterm
         anim-snd : Tterm
      processes
         Animation
   end
   imports
      Tool-data
   atoms
      anim : Tterm
   processes
      Choose : Tterm # Tterm
      Choose : Tterm
      Run : Tterm # BOOLEAN
   sets
      of Tterm
         ATOM-set = { atom(a) | a in ATOM }
      of Tterm
         CHOOSE = { choose(l) | l in Tterm }
      of Tterm
         TOOL = { sim, anim }
      of BOOLEAN
         CONTROL-INFO = { false, true }
   variables
      l : -> Tterm
      b : -> Tterm
      a : -> ATOM
      control : -> Tterm
      keep-control : -> BOOLEAN
   definitions
      Animation = anim-rec(control-info) .
                  sum(t in TOOL, sum(ci in CONTROL-INFO,
                     anim-snd(control(t, ci)) .
                     Run(t, ci)
                  ) )
      Run(control, keep-control) =
```

```
        (
           sum(a in ATOM-set, anim-rec(a) . anim(a))
        +  anim-rec(reset) . anim(reset)
        +  anim-rec(end-of-spec) . anim(end-of-spec) . (
              anim(reset) . anim-snd(reset) .
              Run(control, keep-control)
           +  anim(quit) . anim-snd(quit)
           )
        +  sum(c in CHOOSE, anim-rec(c) . (
                 Choose(c) . Run(control, keep-control)
              +  anim(reset) . anim-snd(reset) .
                 Run(control, keep-control)
              +  anim(quit) . anim-snd(quit)
              +  [ keep-control = false ] -> (
                    anim(controltosim) . anim-snd(givecontrol) .
                    Run(sim, keep-control)
                 )
              )
           )
        ) . (
           anim-snd(ack)
        +  anim-snd(error)
        ) . Run(control, keep-control)
     +  anim-rec(takecontrol) . anim(takecontrol) .
        Run(anim, keep-control)
  Choose(choose(l | b)) = anim(b) . anim-snd(choice(b))
        +  Choose(choose(l))
  Choose(choose(atom(a))) = anim(atom(a)) . anim-snd(choice(atom(a)))
end Animation
```

3.2 Level 2: The interfaces

The main task of the interfaces is to convert the data the messages of the tools to a form the ToolBus can handle and vice versa. For this, we introduce two conversion functions and some functions for deciding the type of the messages.

```
data module Tool-ToolBus-data
begin
   exports
   begin
      functions
         tb-term : Tterm -> TBterm
         conv : Tterm -> TBterm
         conv : TBterm -> Tterm

         tool : TBterm -> TBterm
         get-choice : TBterm -> Tterm

         is-choose : TBterm -> BOOLEAN
         is-choice : TBterm -> BOOLEAN
         is-control : TBterm -> BOOLEAN
         is-atom : TBterm -> BOOLEAN
   end
   imports
      Tool-data,
      ToolBus-Types
   variables
      t : -> Tterm
      n : -> BOOLEAN
      a : -> ATOM
   equations
   [1] conv(t) = tb-term(t)
   [2] conv(tb-term(t)) = t
   [3] tool(tb-term(control(t, n))) = conv(t)
   [4] tool(tb-term(control(t))) = conv(t)
   [5] get-choice(tb-term(choice(t))) = t

   [6] is-choose(tb-term(choose(t))) = true
   [7] is-choice(tb-term(choice(t))) = true
   [8] is-control(tb-term(control(t, n))) = true
   [9] is-control(tb-term(control(t))) = true
   [10] is-atom(tb-term(atom(a))) = true
end Tool-ToolBus-data
```

Now, we can specify the interfaces. These interfaces start up the tools and arrange for the communications with the tools to take place.

3.2.1 The interface of the simulator

The atoms `simtb-snd` and `simtb-rec` are used for communication with the ToolBus, and the atoms `simint-rec` and `simint-snd` are used for communication with the simulator. The atom `simint-comm` represents a communication with data going from the simulator to the interface, and the atom `intsim-comm` a communication with data going the other way.

The main process is `SimInt`, which starts the simulator and interface in parallel and enforces the communications to take place with the use of the encaps operator.

```
process module Sim-Interface
begin
   exports
   begin
      atoms
         simtb-snd : TBterm
         simtb-rec : TBterm
      processes
         SimInt
   end
   imports
      Simulator, Tool-ToolBus-data
   atoms
      simint-rec : Tterm
      simint-snd : Tterm
      simint-comm : Tterm
      intsim-comm : Tterm
   processes
      Interface
   sets
      of atoms
         H = { sim-snd(x), sim-rec(x), simint-rec(x), simint-snd(x)
               | x in Tterm }
      of Tterm
         ATOM-set = { atom(a) | a in ATOM }
      of Tterm
         CHOOSE = { choose(l) | l in Tterm }
   communications
      sim-snd(x) | simint-rec(x) = simint-comm(x) for x in Tterm
      sim-rec(x) | simint-snd(x) = intsim-comm(x) for x in Tterm
   definitions
      Interface =
                  sum(t in TBterm, simtb-rec(t) .
                     (
                        [ is-control(t) = true ] ->
                           simint-snd(conv(t))
                     +  [ conv(t) = send-message ] ->
                           simint-snd(send-message)
                     +  [ is-choice(t) = true ] -> (
                           simint-snd(get-choice(t))
                        )
                     +  [ conv(t) = reset ] ->
                           simint-snd(reset)
                     +  [ conv(t) = quit ] ->
                           simint-snd(quit)
                     +  [ conv(t) = takecontrol ] ->
                           simint-snd(takecontrol)
                     )
                  ) . Interface
               +  sum(a in ATOM-set, simint-rec(a) .
                     simtb-snd(conv(a)) . simtb-rec(conv(ack))
                  ) . simint-snd(ack) . Interface
               +  simint-rec(reset) . simtb-snd(conv(reset)) .
                  simtb-rec(conv(ack)) .  simint-snd(ack) . Interface
               +  simint-rec(quit) . simtb-snd(conv(quit))
               +  simint-rec(end-of-spec) . simtb-snd(conv(end-of-spec)) .
                  simtb-rec(conv(ack)) .  simint-snd(ack) . Interface
               +  simint-rec(givecontrol) . simtb-snd(conv(control(anim))) .
                  Interface
               +  sum(c in CHOOSE, simint-rec(c) . simtb-snd(conv(c))) .
                  Interface
      SimInt = encaps(H, Simulator || Interface)
end Sim-Interface
```

3.2.2 The interface of the animation

The naming of the atoms in the interface for the animation, is done in the same manner as in the interface for the simulator.

```
process module Anim-Interface
begin
    exports
    begin
        atoms
            animtb-snd : TBterm
            animtb-rec : TBterm
        processes
            AnimInt
    end
    imports
        Animation, Tool-ToolBus-data
    atoms
        animint-rec : Tterm
        animint-snd : Tterm
        animint-comm : Tterm
        intanim-comm : Tterm
    processes
        Interface
    sets
        of atoms
            H = { anim-snd(x), anim-rec(x), animint-rec(x), animint-snd(x)
                  | x in Tterm }
    communications
        anim-snd(x) | animint-rec(x) = animint-comm(x) for x in Tterm
        anim-rec(x) | animint-snd(x) = intanim-comm(x) for x in Tterm
    definitions
        Interface =
                sum(t in TBterm,
                    animtb-rec(t) . (
                        [ conv(t) = control-info ] -> (
                          animint-snd(control-info) .
                          sum(c in Tterm, animint-rec(c) .
                             animtb-snd(conv(c)) . Interface
                          )
                        )
                    +   [ is-atom(t) = true ] -> (
                          animint-snd(conv(t))
                        )
                    +   [ conv(t) = reset ] -> (
                          animint-snd(reset)
                        )
                    +   [ conv(t) = end-of-spec ] -> (
                          animint-snd(end-of-spec) .
                          sum(c in Tterm, animint-rec(c) . (
                                [ is-choice(conv(c)) = true ] ->
                                   animtb-snd(conv(c))
                             +  [ c = reset ] -> animtb-snd(conv(c))
                             +  [ c = quit ] -> animtb-snd(conv(c))
                             )
                          ) . Interface
                        )
                    +   [ conv(t) = takecontrol ] -> (
                          animint-snd(takecontrol) . Interface
                        )
                    +   [ is-choose(t) = true ] -> (
                          animint-snd(conv(t)) .
                          sum(c in Tterm, animint-rec(c) . (
                                [ is-choice(conv(c)) = true ] ->
                                   animtb-snd(conv(c))
                             +  [ c = reset ] -> animtb-snd(conv(c))
                             +  [ c = quit ] -> animtb-snd(conv(c))
                             +  [ c = givecontrol ] ->
                                   animtb-snd(conv(control(sim)))
                             )
                          ) . Interface
                        )
                    +   [ is-choice(t) = true ] -> (
                          animint-snd(get-choice(t))
                        )
                    )
                ) . (
                    animint-rec(ack) . animtb-snd(conv(ack))
                +   animint-rec(error) . animtb-snd(conv(error))
                ) . Interface
        AnimInt = encaps(H, Animation || Interface)
end Anim-Interface
```

3.3 Level 3: The ToolBus

For each tool, we use a process in the ToolBus. Before we can specify these processes, we have to specify the primitives for the ToolBus.

The atoms `tb-snd-msg`, `tb-rec-msg`, `tb-snd-eval`, `tb-rec-value`, `tb-snd-do`, and `tb-shutdown`, represent their equivalents in ToolBus-scripts. The atom `tb-comm-msg` represents a communications between `tb-snd-msg` and `tb-rec-msg`.

```
process module ToolBus-primitives
begin
   exports
   begin
      atoms
         tb-snd-msg : TBterm # TBterm
         tb-rec-msg : TBterm # TBterm
         tb-comm-msg : TBterm # TBterm

         tb-snd-eval : TBid # TBterm
         tb-rec-value : TBid # TBterm
         tb-snd-do : TBid # TBterm

         tb-shutdown
   end
   imports
      ToolBus-Types
   communications
      tb-snd-msg(t,m) | tb-rec-msg(t, m) = tb-comm-msg(t, m)
         for t in TBterm, m in TBterm
end ToolBus-primitives
```

Now we can specify the processes, which startup the interfaces in parallel and arrange for the communications with the interfaces to take place. These two processes are run by the ToolBus in parallel.

3.3.1 The ToolBus-process for the simulator

The atom `simtb-comm-snd` is used for a communication with data going to the ToolBus, and `simtb-comm-rec` for a communication with data going to the interface.

When it is necessary for the simulator to receive an acknowledgement after a message is send, this is send immediately to the simulator without waiting for the animation to react on this message. This enables the simulator to perform some tasks, instead of waiting on a message from the animation.

```
process module Process-Sim
begin
   exports
   begin
      atoms
         simtb-comm-snd : TBterm
         simtb-comm-rec : TBterm
      processes
         Process-Sim
   end
   imports
      ToolBus-primitives,
      ToolBus-ID,
      Sim-Interface
   processes
      TB-Sim : TBterm
   sets
      of atoms
         H = { tb-snd-eval(tid, t), tb-rec-value(tid, t), tb-snd-do(tid, t),
               simtb-snd(t), simtb-rec(t) | tid in TBid, t in TBterm }
   communications
      simtb-snd(t) | tb-rec-value(tid, t) = simtb-comm-snd(t)
         for t in TBterm, tid in TBid
      simtb-rec(t) | tb-snd-eval(tid, t) = simtb-comm-rec(t)
         for t in TBterm, tid in TBid
      simtb-rec(t) | tb-snd-do(tid, t) = simtb-comm-rec(t)
         for t in TBterm, tid in TBid
   variables
      t : -> TBterm
   definitions
      Process-Sim =
            encaps(H,
               SimInt
```

```
               || sum(m in TBterm,
                     tb-rec-msg(psim, m) . tb-snd-do(SIM, m) .
                     TB-Sim(tool(m))
                  )
               )
         TB-Sim(t) =  (
               [ t = conv(anim) ] -> (
                  tb-snd-eval(SIM, conv(send-message)) .
                  sum(v in TBterm,
                     tb-rec-value(SIM, v) . (
                        [ is-choose(v) = true ] -> (
                           tb-snd-msg(panim, v)
                        )
                     +  [ v = conv(end-of-spec) ] -> (
                           tb-snd-msg(panim, v) .
                           tb-snd-do(SIM, conv(ack))
                        )
                     )
                  ) .
                  sum(v in TBterm,
                     tb-rec-msg(psim, v) . (
                        [ is-choice(v) = true ] -> (
                           tb-snd-do(SIM, v)
                        )
                     +  [ v = conv(reset) ] -> (
                           tb-snd-do(SIM, v)
                        )
                     +  [ v = conv(quit) ] -> (
                           tb-snd-eval(SIM, v) .
                           tb-rec-value(SIM, v) .
                           tb-shutdown
                        )
                     +  [ is-control(v) = true ] -> (
                           tb-snd-do(SIM, conv(takecontrol)) .
                           TB-Sim(tool(v))
                        )
                     )
                  )
               )
            +  [ t = conv(sim) ] -> (
                  tb-snd-eval(SIM, conv(send-message)) .
                  sum(v in TBterm,
                     tb-rec-value(SIM, v) . (
                        (
                           [ is-atom(v) = true ] -> (
                              tb-snd-msg(panim, v)
                           )
                        +  [ v = conv(reset) ] -> (
                              tb-snd-msg(panim, v)
                           )
                        +  [ v = conv(quit) ] -> (
                              tb-shutdown
                           )
                        ) .
                        tb-snd-do(SIM, conv(ack))
                     +  [ is-control(v) = true ] -> (
                           tb-snd-msg(panim, v) .
                           TB-Sim(tool(v))
                        )
                     )
                  )
               )
            ) . TB-Sim(t)
      end Process-Sim
```

3.3.2 The ToolBus-process for the animation

The atom `animtb-comm-snd` is used for a communication with data going to the ToolBus, and `animtb-comm-rec` for a communication with data going to the interface.

When the animation is in control, it sends the choice made from the choose-list. This choice is send to the ToolBus-process for the simulator, after which the choice is also send to the animation. It is done this way, because we want both the choice made and the result of the animation (an acknowledgement or an error). These two can also be send combined, but there is a possibility that the animation of the choice takes a long time. In the meantime, the simulator now can calculate the next choose-list.

```
      process module Process-Anim
      begin
```

```
exports
begin
   atoms
      animtb-comm-snd : TBterm
      animtb-comm-rec : TBterm
   processes
      Process-Anim
end
imports
   ToolBus-primitives,
   ToolBus-ID,
   Anim-Interface
processes
   TB-Anim : TBterm
sets
   of atoms
      H = { tb-snd-eval(tid, t), tb-rec-value(tid, t), tb-snd-do(tid, t),
            animtb-snd(t), animtb-rec(t) | tid in TBid, t in TBterm }
communications
   animtb-snd(t) | tb-rec-value(tid, t) = animtb-comm-snd(t)
      for t in TBterm, tid in TBid
   animtb-rec(t) | tb-snd-eval(tid, t) = animtb-comm-rec(t)
      for t in TBterm, tid in TBid
   animtb-rec(t) | tb-snd-do(tid, t) = animtb-comm-rec(t)
      for t in TBterm, tid in TBid
variables
   t : -> TBterm
definitions
   Process-Anim =
         encaps(H,
            AnimInt
         || tb-snd-eval(ANIM, conv(control-info)) .
            sum(v in TBterm,
               tb-rec-value(ANIM, v) . (
                  [ is-control(v) = true ] -> (
                     tb-snd-msg(psim, v) .
                     TB-Anim(tool(v))
                  )
               +  [ v = conv(error) ] -> (
                     tb-shutdown
                  )
               )
            )
         )
   TB-Anim(t) = (
         [ t = conv(anim) ] -> (
            sum(v in TBterm,
               tb-rec-msg(panim, v) .
               tb-snd-eval(ANIM, v)  -- choose(..) or end-of-spec
            ) .
            sum(v in TBterm,
               tb-rec-value(ANIM, v) . (
                  [ is-choice(v) = true ] -> (
                     tb-snd-msg(psim, v) .
                     tb-snd-eval(ANIM, v) .
                     (
                        tb-rec-value(ANIM, conv(ack))
                     +  tb-rec-value(ANIM, conv(error)) .
                        tb-shutdown
                     )
                  )
               +  [ v = conv(reset) ] -> (
                     tb-snd-msg(psim, v)
                  )
               +  [ v = conv(quit) ] -> (
                     tb-snd-msg(psim, v)
                  )
               +  [ is-control(v) = true ] -> (
                     tb-snd-msg(psim, v) .
                     TB-Anim(tool(v))
                  )
               )
            )
         )
      +  [ t = conv(sim) ] -> (
            sum(v in TBterm,
               tb-rec-msg(panim, v) . (
                  (
                     [ is-atom(v) = true ] -> (
                        tb-snd-eval(ANIM, v)
                     )
                  +  [ v = conv(reset) ] -> (
                        tb-snd-eval(ANIM, v)
```

```
                              )
                           ) . (
                              tb-rec-value(ANIM, conv(ack))
                           +  tb-rec-value(ANIM, conv(error)) .
                              tb-shutdown
                           )
                        +  [ is-control(v) = true ] -> (
                              tb-snd-do(ANIM, conv(takecontrol)) .
                              TB-Anim(tool(v))
                           )
                        )
                     )
                  )
               ) . TB-Anim(t)
end Process-Anim
```

3.3.3 *Specification of the ToolBus*

Finally, we can specify the ToolBus itself. We use a process `ToolBus-Control` to perform a shutdown when this is requested from one of the processes in the ToolBus. The shutdown is enforced by the use of the disrupt and prio operators.

```
process module ToolBus-SimAnim
begin
   imports
      ToolBus-primitives,
      Process-Sim,
      Process-Anim
   atoms
      application-shutdown
      tbc-shutdown
      tbc-app-shutdown
      TB-Shutdown
      TB-App-Shutdown
   processes
      ToolBus-SimAnim
      ToolBus-Control
      Application
   sets
      of atoms
         H = { tb-snd-msg(t, m), tb-rec-msg(t, m), tbc-shutdown,
               tbc-app-shutdown, tb-shutdown, application-shutdown
               | t in TBterm, m in TBterm}
         P = { TB-Shutdown, TB-App-Shutdown }
   communications
      tb-shutdown | tbc-shutdown = TB-Shutdown
      tbc-app-shutdown | application-shutdown = TB-App-Shutdown
   definitions
      ToolBus-SimAnim =
               encaps(H, prio(P > atoms, ToolBus-Control || Application))
      ToolBus-Control = tbc-shutdown . tbc-app-shutdown
      Application =
               disrupt(Process-Sim || Process-Anim, application-shutdown)
end ToolBus-SimAnim
```

4. *Implementation of simanim*

As mentioned earlier, **simanim** is a script that controls the execution of the ToolBus, which in turn controls the execution of the simulator and the animation. An overview is given in Figure 4-1. The sim-adapter is needed to preserve the capabilities of the simulator that use the standard input and standard output. It mainly sets up two pipes to communicate with the simulator. The sim-adapter is written in Perl, and therefore we need the perl-adapter. The simulator is extended with an interface for communicating over the pipes.

Animations must be written in Tcl/Tk, and are connected to the ToolBus with the use of the tcl-adapter.
The choice for Tcl/Tk is because the capabilities of Tcl/Tk fulfill our needs, but any language that covers the needed capabilities could have been chosen, and perhaps in future, support for other languages will be made.

The script for the ToolBus can be derived from the specification of the processes in the ToolBus in section 3.3. This script can be found in appendix C..

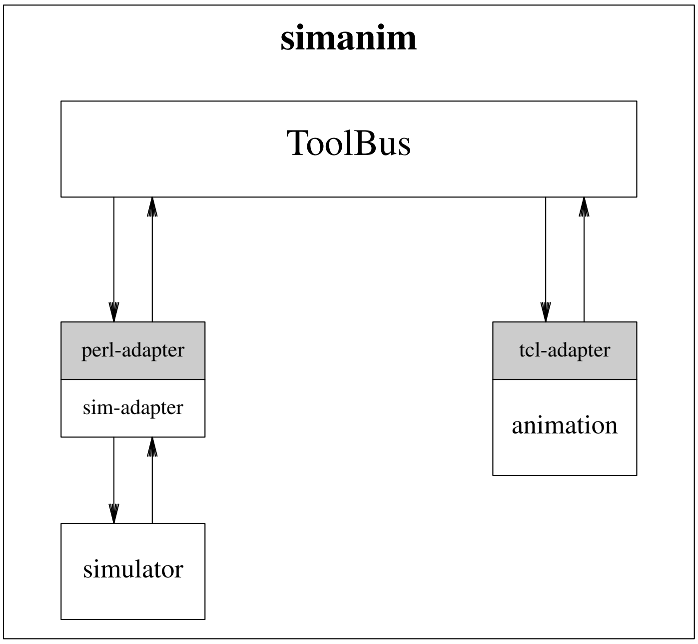


Figure 4-1. overview of simanim

5. *References*

[BaeWeij90] J.C.M. Baeten and W.P. Weijland, *Process Algebra,* Cambridge Tracts in Theoretical Computer Science 18, Cambridge University Press, 1990.

[BerHeeKli89] J.A. Bergstra, J. Heering, and P. Klint, “The Algebraic Specification Formalism ASF,” in *Algebraic Specification*, ed. J.A. Bergstra, J. Heering, P. Klint, ACM Press Frontier Series, pp. 1-66, Addison-Wesley, 1989.

[BerKli95] J.A. Bergstra and P. Klint, “The discrete time ToolBus,” report P9502, Programming Research Group - University of Amsterdam, March 1995.

[BerKlo86] J.A. Bergstra and J.W. Klop, “Process algebra: specification and verification in bisimulation semantics,” in *Math. & Comp. Sci. II*, ed. M. Hazewinkel, J.K. Lenstra, L.G.L.T. Meertens, eds., CWI Monograph 4, pp. 61-94, North-Holland, Amsterdam, 1986.

[Die94] B. Diertens, “New Features in PSF I - Interrupts, Disrupts, and Priorities,” report P9417, Programming Research Group - University of Amsterdam, June 1994.

[DiePon94] B. Diertens and A. Ponse, “New Features in PSF II - Iteration and Nesting,” report P9425, Programming Research Group - University of Amsterdam, October 1994.

[MauVel90] S. Mauw and G.J. Veltink, “A Process Specification Formalism,” in *Fundamenta Informaticae XIII (1990)*, pp. 85-139, IOS Press, 1990.

[MauVel93] S. Mauw and G.J. Veltink (eds.), *Algebraic Specification of Communication Protocols,* Cambridge Tracts in Theoretical Computer Science 36, Cambridge University Press, 1993.

A. PSF specifications

A.1 Alternating Bit Protocol

```
data module Bits
begin
    exports
    begin
        sorts
            BIT
        functions
            0 :-> BIT
            1 :-> BIT
            flip : BIT -> BIT
    end
    equations
    [B1] flip(0) = 1
    [B2] flip(1) = 0
end Bits

data module Data
begin
    exports
    begin
        sorts
            DATA
        functions
            'a :-> DATA
            'b :-> DATA
            'c :-> DATA
            'd :-> DATA
            'e :-> DATA
    end
end Data

data module Frames
begin
    exports
    begin
        sorts
            FRAME
        functions
            frame : BIT # DATA -> FRAME
            frame-error :-> FRAME
    end
    imports
        Data, Bits
end Frames

data module Acknowledgements
begin
    exports
    begin
        sorts
            ACK
        functions
            ack : BIT -> ACK
            ack-error :-> ACK
    end
    imports
        Bits
end Acknowledgements

process module ABP
begin
    imports
        Bits, Data, Frames, Acknowledgements
    atoms
        input : DATA
        send-frame : FRAME
        receive-ack-or-error : ACK
        receive-frame : FRAME
        send-frame-or-error : FRAME
        receive-frame-or-error : FRAME
        output : DATA
        send-ack : ACK
```

```
      receive-ack : ACK
      send-ack-or-error : ACK
      frame-comm : FRAME
      frame-or-error : FRAME
      ack-comm : ACK
      ack-or-error : ACK
   processes
      Sender
      Receive-Message : BIT
      Send-Frame : BIT # DATA
      Receive-Ack : BIT # DATA
      K
      K : BIT # DATA
      Receiver
      Receive-Frame : BIT
      Send-Ack : BIT
      Send-Message : BIT # DATA
      L
      L : BIT
      ABP
   sets
      of atoms
         H = { send-frame(f), receive-frame(f) | f in FRAME }
            + { send-frame-or-error(f), receive-frame-or-error(f)
               | f in FRAME }
            + { send-ack(a), receive-ack(a) | a in ACK }
            + { send-ack-or-error(a), receive-ack-or-error(a) | a in ACK }
         I = { frame-comm(f), frame-or-error(f) | f in FRAME }
            + { ack-comm(a), ack-or-error(a) | a in ACK }
      of BIT
         Bit-Set = { 0, 1 }
   communications
      send-frame(f) | receive-frame(f) = frame-comm(f) for f in FRAME
      send-frame-or-error(f) | receive-frame-or-error(f) =
         frame-or-error(f) for f in FRAME
      send-ack(a) | receive-ack(a) = ack-comm(a) for a in ACK
      send-ack-or-error(a) | receive-ack-or-error(a) =
         ack-or-error(a) for a in ACK
   variables
      f :-> FRAME
      b :-> BIT
      d :-> DATA
      a :-> ACK
   definitions
      Sender = Receive-Message(0)
      Receive-Message(b) = sum(d in DATA, input(d) . Send-Frame(b,d))
      Send-Frame(b,d) = send-frame(frame(b,d)) . Receive-Ack(b,d)
      Receive-Ack(b,d) = (
                  receive-ack-or-error(ack(flip(b)))
               +  receive-ack-or-error(ack-error)
               ) . Send-Frame(b,d)
            +  receive-ack-or-error(ack(b)) . Receive-Message(flip(b))

      K = sum(d in DATA, sum(b in Bit-Set, receive-frame(frame(b,d)) . K(b,d) ))
      K(b,d) = (
               skip . send-frame-or-error(frame(b,d))
            +  skip . send-frame-or-error(frame-error)
            ) . K

      Receiver = Receive-Frame(0)
      Receive-Frame(b) = (
                  sum(d in DATA, receive-frame-or-error(frame(flip(b),d)))
               +  receive-frame-or-error(frame-error)
               ) . Send-Ack(flip(b))
            +  sum(d in DATA, receive-frame-or-error(frame(b,d)) .
                  Send-Message(b,d)
               )
      Send-Ack(b) = send-ack(ack(b)) . Receive-Frame(flip(b))
      Send-Message(b,d) = output(d) . Send-Ack(b)

      L = sum(b in Bit-Set, receive-ack(ack(b)) . L(b) )
      L(b) = (
               skip . send-ack-or-error(ack(b))
            +  skip . send-ack-or-error(ack-error)
            ) . L

      ABP = hide(I, encaps(H, Sender || Receiver || K || L ))
end ABP
```

A.2 *Factory*

```
data module Products
begin
   exports
   begin
      sorts
         PRODUCT
      functions
         A : -> PRODUCT
         B : -> PRODUCT
   end
end Products

data module Stations
begin
   exports
   begin
      sorts
         STATION
      functions
         1 : -> STATION
         2 : -> STATION
         3 : -> STATION
         4 : -> STATION
         5 : -> STATION
         6 : -> STATION
         eq-stat : STATION # STATION -> BOOLEAN
         next : STATION # PRODUCT -> STATION
   end
   imports
      Booleans, Products
   variables
      x : -> STATION
      y : -> STATION
      p : -> PRODUCT
   equations
   [1] eq-stat(x, x) = true
   [2] not(eq-stat(x, y)) = true
   [3] next(1, p) = 2
   [4] next(2, p) = 3
   [5] next(3, A) = 4
   [6] next(3, B) = 5
   [7] next(4, p) = 5
   [8] next(5, p) = 6
end Stations

process module Factory
begin
   exports
   begin
      atoms
         input : PRODUCT
         output : PRODUCT
      processes
         Start
      sets
         of PRODUCT
            PRODUCT-set = { A, B }
   end
   imports
      Stations
   atoms
      read-input : PRODUCT
      send-input : PRODUCT
      comm-input : PRODUCT
      read-output : PRODUCT
      send-output : PRODUCT
      comm-output : PRODUCT
      to-belt : STATION # STATION # PRODUCT
      from-belt : STATION # PRODUCT
      comm-belt : STATION # STATION # PRODUCT
   processes
      Input
      Stations
      Station : STATION
      Output
   sets
      of STATION
         STATION-set = { 1, 2, 3, 4, 5, 6 }
```

```
        of atoms
           H = { send-input(p), read-input(p), send-output(p),
                 read-output(p), to-belt(x, y, p), from-belt(y, p)
                 | p in PRODUCT, x in STATION, y in STATION }
     communications
        send-input(p) | read-input(p) = comm-input(p)
           for p in PRODUCT
        send-output(p) | read-output(p) = comm-output(p)
           for p in PRODUCT
        to-belt(s1, s2, p) | from-belt(s2, p) = comm-belt(s1, s2, p)
           for s1 in STATION, s2 in STATION, p in PRODUCT
     variables
        s : -> STATION
     definitions
        Start = encaps(H, Input || Stations || Output)
        Input = sum(p in PRODUCT-set, input(p) . send-input(p)) . Input
        Stations = merge(s in STATION-set, Station(s))
        Station(s) =
              [eq-stat(s, 1) = true] -> (
                 sum(p in PRODUCT,
                    read-input(p) . to-belt(s, next(s, p), p)
                 ) . Station(s)
              )
           +  [eq-stat(s, 6) = true] -> (
                 sum(p in PRODUCT,
                    from-belt(s, p) . send-output(p)
                 ) . Station(s)
              )
           +  [and(not(eq-stat(s, 1)), not(eq-stat(s, 6))) = true] -> (
                 sum(p in PRODUCT,
                    from-belt(s, p) . to-belt(s, next(s, p), p)
                 ) . Station(s)
              )
        Output = sum(p in PRODUCT, read-output(p) . output(p)) . Output
  end Factory
```

A.3 Factory with Queues

```
  data module S-Products
  begin
     exports
     begin
        functions
           eq-prod : PRODUCT # PRODUCT -> BOOLEAN
           error : -> PRODUCT
     end
     imports
        Products, Booleans
  end S-Products

  process module S-Factory
  begin
     imports
        Factory,
        Sequences {
           Elements bound by [
              ITEM -> PRODUCT,
              eq -> eq-prod,
              error-element -> error
           ] to S-Products
        }
     atoms
        q-input : PRODUCT
        q-output : PRODUCT
        q-send-input : PRODUCT
        q-read-output : PRODUCT
        comm-q-input : PRODUCT
        comm-q-output : PRODUCT
     processes
        Start-Q-Factory
        In-Queue : SEQ
        Out-Queue : SEQ
     sets
        of atoms
           Q-H = { input(p), output(p), q-send-input(p), q-read-output(p)
                 | p in PRODUCT }
     communications
        q-send-input(p) | input(p) = comm-q-input(p) for p in PRODUCT
        q-read-output(p) | output(p) = comm-q-output(p) for p in PRODUCT
```

```
    variables
       q : -> SEQ
    definitions
       Start-Q-Factory = encaps(Q-H,
          In-Queue(empty-sequence) || Start || Out-Queue(empty-sequence))
       In-Queue(q) = sum(p in PRODUCT-set,
                q-input(p) . In-Queue(q ^ p) )
          +  [not(eq(q, empty-sequence))=true] ->
                q-send-input(first(q)) . In-Queue(tail(q))
       Out-Queue(q) = sum(p in PRODUCT-set,
                q-read-output(p) . Out-Queue(q ^ p) )
          +  [not(eq(q, empty-sequence))=true] ->
                q-output(first(q)) . Out-Queue(tail(q))
end S-Factory
```

B. Reference Guide

Here, a description of the available functions is given. The functions are listed in alphabetical order. A function description consist of its name followed by its argument, and a description of its behaviour below this.
Functions are given in **bold** and arguments of functions in *italic*. Arguments in between square brackets are optional. An argument followed by ... indicates that this argument may appear more than once. Arguments of options separated by | means that they are alternatives.

ANIM_activate_item *item*

Activates *item*, which means that it changes color, and when clicked upon, the list with possible actions for this *item* is shown.
Note: this is done automatically when the first action is added to *list* with the function **ANIM_add_list**.

ANIM_activate_line *line*

Activates *line*, which means that the line changes color.

ANIM_activate_list *list*

Activates *list.* When the corresponding item is activated and clicked upon, this list is shown.
Note: this is done automatically when the first action is added to *list* with the function **ANIM_add_list**.

ANIM_add_clear *item duplet ...*

Adds the *duplet*s to the clear-list of *item.* When the function **ANIM_clear** is called for *item*, the lines and texts indicated with the *duplet*s are cleared (deactivated are deleted).
A duplet has the following form

{ *type id* }

type is either **line** or **text,** and *id* is the name of the line or text.

ANIM_add_list *list entry*

Add *entry* to *list*.

ANIM_add_queue *queue string*

Add *string* to *queue*.

ANIM_change_text_item *item string*

Change the text displayed on *item* into *string*.

ANIM_clear *item*

Clear the things found in the clear-list of *item,* added by function **ANIM_add_clear**.

ANIM_colorlistbox *normal select*

Sets the color for listboxes to *normal* and for selected items in the listboxes to *select*.

ANIM_colorset *set type normal active*

Sets the normal and active colors for *type* in colorset *set* to *normal* and *active*. *Type*may*be*one*of* **rect**, **oval**, **line**, or **text**. If colorset *set* does not exist it is made with initial values copied from colorset 0.
Note: Colorsets 0 and 1 are predefined, but values can be changed with this function.

ANIM_create_box *pbox name* [*options*]

Creates a box in the INFO-window with as parent-box *pbox*, and with id *name*. The top parent-box is created by default and is called **info**.
Options:
-**side** top | bottom | left | right

Specify to which side of the box the children (boxes and labels) will be placed.
Default top.

-**fill** none | x | y | both

If a child of this box is smaller than the available space for this child, it is stretched according the the value given for this option.

none No stretching.

x Stretch the children horizontally to fill the entire width of the available space for the children.

y Stretch the children vertically to fill the entire height of the available space for the children.

both Stretch the children both horizontally and vertically.

Default none.

-relief flat | groove | raised | ridge | sunken
The type of border (3D-effect) to be drawn around the box.
Default flat.

-bw *width*
The width of the border.
Default 0.

-ipadx *pixels*
Pixels specifies how much horizontal space to leave on each side of the children of the box.
Default 0.

-ipady *pixels*
Pixels specifies how much vertical space to leave on each side of the children of the box.
Default 0.

-expand
If this option is given, and if there is still space left unoccupied by the children, the children are expanded. Extra horizontal space is added to the children for which the **-side** option has the value **left** or **right**, and extra vertical space is added to the children for which the **-side** option has the value **top** or **bottom**.

ANIM_create_item *item type x y w h string* [*options*]
Creates an item of *type* with the center at x, y and with width $2 * w$ and height $2 * h$. *type* can either be **rect** for a rectangle or **oval** for an oval. *string* is centered on item. *Item* serves as the name in order to make references to it in calls to the other functions.
Options:

-nolist
There will be no list associated with *item*.

-free
Registers the item as to be freed on a reset.

-color *setname*
Setname specifies the name of the colorset to be used for *item*. By default there are 2 colorsets available named **0** and **1.** New colorsets can be made with **ANIM_colorset**.
Default 0.

ANIM_create_label *box name string* [*options*]
Creates a label with *string* as text in *box*.
Options:

-var
string is treated as the name of a variable, which means that the value of the variable is displayed instead of *string*. When the variable is updated, the label is updated too.

-relief flat | groove | raised | ridge | sunken
The type of border (3D-effect) to be drawn around the label.
Default flat.

-bw *width*
The borderwidth of the label.
Default 2.

-width *width*
The width of the label.

Default 0.

-**anchor** *anchor*

Specifies how *string* is to be displayed in the space for the label. Possible values are **n**, **ne**, **e**, **se**, **s**, **sw**, **w**, **nw**, or **center**. For example, **nw** means display the top-left corner of *string* at the top-left corner of the label.
Default center.

-**padx** *pixels*

Extra space on the left and right of the label.
Default 1.

ANIM_create_line *line triplet triplet ...* [*options*]

Draws a line from *triplet* to *triplet* to ... . A triplet has one of the following forms

pos *x y*

to indicate position **x,y**, and

item *name anchor*

to indicate the position of *anchor* of item **name**. Possible values for *anchor* are

n e s w nw ne se sw ce chop

Chop means that the line is chopped at the border of item.
Options:

-**width** *width*

The width of *line*.
Default 3.

-**arrow** none | first | last | both

Specify on which ends of the line to draw arrows.
Default none.

-**color** *setname*

Setname specifies the name of the colorset to be used for *item*. By default there are 2 colorsets available named **0** and **1.** New colorsets can be made with **ANIM_colorset**.
Default 0.

-**nolower**

Normally a line is lowered so that all items are displayed on top of this line. This option turns this off.

ANIM_create_queue *queue x y w h* [*options*]

Creates a queue consisting of a text-window and a scrollbar. It is positioned with the anchor given by the option -**anchor** on the position *x*,*y*. The text-window has the width *w* and the height *h*. The queue-items in a horizontal queue are separated by a space, and in a vertical queue they appear one a line. Options:

-**orientation** horizontal | vertical

Specifies the orientation of *queue*.
Default horizontal.

-**anchor** *anchor*

Specifies the point of *queue* that will be put on the position *x*,*y*. Possible values are **n**, **ne**, **e**, **se**, **s**, **sw**, **w**, **nw**, or **center**.
Default center.

ANIM_create_text *id string*

Create a text *string* on the text-position indicated by *id* formerly created by either **ANIM_textpos**, **ANIM_textpos_item**, **ANIM_textpos_line**.

ANIM_deactivate_item *item*

Opposite of **ANIM_activate_item**.

ANIM_deactivate_line *line*

Opposite of **ANIM_activate_line**.

ANIM_delete_text *id*

Deletes text from the text-position indicated by *id*, formerly created with **ANIM_create_text**.

ANIM_destroy_item *item*

Destroys the item indicated by *item* and formerly created by **ANIM_create_item**.

ANIM_dim *id dimension*

Returns the dimension *dimension* of item *id*. Possible values for *dimension* are

x y ce n e s w nw ne se sw n,y s,y e,x w,x wid ht

ANIM_diml *id dimension*

Returns the dimension *dimension* of line *id*. Possible values for *dimension* are

start end start,x start,y end,x end,y

ANIM_dimq *id dimension*

Returns the dimension *dimension* of queue *id*. Possible values for *dimension* are

x y ce n e s w nw ne se sw n,y s,y e,x w,x wid ht

ANIM_init_array *name list*

Initializes array *name* with indices and values taken from *list*. *list* must consist of a list of index value index value ... separated by spaces. On a reset, this initialization is also performed.

ANIM_init_var *name value*

Initializes variable *name* with *value*. On a reset, this initialization is also performed.

ANIM_move *item movement ...* [*options*]

Moves *item* along the path given by the *movement*s, where a *movement* is one of the following

left *d*
right *d*
up *d*
down *d*
leftto *x*
rightto *x*
upto *y*
downto *y*

Here, *d* is a distance in pixels, *x* is a x-coordinate, and *y* is a y-coordinate.

Options:

-newid *id*

Changes the id of the item into *id*.

ANIM_sub_queue *queue*

Removes the first queue-item from *queue*.

ANIM_textpos *id x y anchor* [*options*]

Creates a text-position with name I id with *anchor* on position *x*,*y*. Possible values are **n**, **ne**, **e**, **se**, **s**, **sw**, **w**, **nw**, or **center**.

Options:

-noreset

Indicates that the text on this position is not be deleted on a reset.

ANIM_textpos_item *id item corner anchor*

Creates a text-position with name I id with *anchor* on *corner* of *item*. Possible values are **n**, **ne**, **e**, **se**, **s**, **sw**, **w**, **nw**, or **center**.

ANIM_textpos_line *id line anchor* [*options*]

Creates a text-position with name I id with *anchor* somewhere along *line* according to the options or their defaults. Possible values are **n**, **ne**, **e**, **se**, **s**, **sw**, **w**, **nw**, or **center**.

Options:

-d *distance*

Gives the distance from the beginning of the line for the position of the anchor. *distance* must be given as a fraction (from 0.0 to 1.0) of the length of the line (or segment)

Default 0.5.

-s *segment*

segment indicates to which part of the line the calculation for the position of the anchor are done.

Default 1.

ANIM_windows *canvasw canvash textw texth*

Initializes the windows with *canvasw*and*canvash* for the width and height of the canvas, and with *textw*and*texth* for the width and height, in number of characters, of the text-window.

It also creates a box next to the text-window with name **info**, to be used as parent-box for function **ANIM_create_box**.

C. ToolBus script

```
process PSIM is
let SIM : sim,
   S : str,
   A : str,
   T : str,
   N : int
in
   execute(sim, SIM?) .
   rec-msg(sim, control(T?, N?)) .
   snd-do(SIM, control(T, N)) .
   (
      if equal(T, "anim") then
         snd-eval(SIM, get-text) .
         (
            rec-value(SIM, choose(S?)) .
            snd-msg(anim, choose(S))
         +  rec-value(SIM, end) .
            snd-msg(anim, end) .
            snd-do(SIM, ack)
         ) .
         (
            rec-msg(sim, choice(A?)) .
            snd-do(SIM, choice(A))
         +  rec-msg(sim, reset) .
            snd-do(SIM, reset)
         +  rec-msg(sim, quit) .
            snd-eval(SIM, quit) .
            rec-value(SIM, quit) .
            shutdown("")
         +  rec-msg(sim, control(T?)) .
            snd-do(SIM, take-control)
         )
      else
         snd-eval(SIM, get-text) .
         (
            (
               rec-value(SIM, atom(S?)) .
               snd-msg(anim, atom(S))
            +  rec-value(SIM, reset) .
               snd-msg(anim, reset)
            +  rec-value(SIM, quit) .
               shutdown("")
            ) . snd-do(SIM, ack)
         +  rec-value(SIM, control(T?)) .
            snd-msg(anim, control(T))
         )
      fi
   ) * delta
endlet

#define ANIM_DONE_OR_ERROR \
   ( \
      rec-value(ANIM, ack) \
   +  rec-value(ANIM, error) . \
      shutdown("error") \
   )

process PANIM is
let ANIM : anim,
   S : str,
   A : str,
   T : str,
   N : int
in
   execute(anim, ANIM?) .
   snd-eval(ANIM, control-info) .
   (
      rec-value(ANIM, control(T?, N?))
   +  rec-value(ANIM, error) .
      shutdown("")
   ) .
   snd-msg(sim, control(T, N)) .
   (
      if equal(T, "anim") then
         (
            rec-msg(anim, choose(S?)) .
            snd-eval(ANIM, choose(S))
```

```
        +  rec-msg(anim, end) .
           snd-eval(ANIM, end)
        ) . (
           rec-value(ANIM, choice(A?)) .
           snd-msg(sim, choice(A)) .
           snd-eval(ANIM, action(A)) .
           ANIM_DONE_OR_ERROR
        +  rec-value(ANIM, reset) .
           snd-msg(sim, reset) .
           snd-eval(ANIM, reset) .
           rec-value(ANIM, ack)
        +  rec-value(ANIM, quit) .
           snd-msg(sim, quit)
        +  rec-value(ANIM, control(T?)) .
           snd-msg(sim, control(T))
        )
      else
           (
              rec-msg(anim, atom(S?)) .
              snd-eval(ANIM, action(S))
           +  rec-msg(anim, reset) .
              snd-eval(ANIM, reset)
           ) . ANIM_DONE_OR_ERROR
        +  rec-msg(anim, control(T?)) .
           snd-do(ANIM, take-control)
      fi
   ) * delta
endlet

tool sim is {command = SIM_ADAPTER }
tool anim is {command = ANIM_ADAPTER }

toolbus(PSIM, PANIM)
```

D. Animation of simanim

```
ANIM_windows 480 220 45 10

ANIM_create_item TSIM rect 140 30 20 15 ""
ANIM_create_item TANIM rect 340 30 20 15 ""
ANIM_create_item ISIM rect 140 100 20 15 ""
ANIM_create_item IANIM rect 340 100 20 15 ""
ANIM_create_item SIM rect 140 170 20 15 "SIM"
ANIM_create_item ANIM rect 340 170 20 15 "ANIM"

ANIM_create_line TSIMtoISIM pos [expr [ANIM_dim TSIM x] - 8] \
   [ANIM_dim TSIM s,y] pos [expr [ANIM_dim ISIM x] - 8] [ANIM_dim ISIM n,y] \
   -arrow last
ANIM_create_line ISIMtoTSIM pos [expr [ANIM_dim ISIM x] + 8] \
   [ANIM_dim ISIM n,y] pos [expr [ANIM_dim TSIM x] + 8] [ANIM_dim TSIM s,y] \
   -arrow last
ANIM_create_line ISIMtoSIM pos [expr [ANIM_dim ISIM x] - 8] \
   [ANIM_dim ISIM s,y] pos [expr [ANIM_dim SIM x] - 8] [ANIM_dim SIM n,y] \
   -arrow last
ANIM_create_line SIMtoISIM pos [expr [ANIM_dim SIM x] + 8] \
   [ANIM_dim SIM n,y] pos [expr [ANIM_dim ISIM x] + 8] [ANIM_dim ISIM s,y] \
   -arrow last

ANIM_create_line TANIMtoIANIM pos [expr [ANIM_dim TANIM x] - 8] \
   [ANIM_dim TANIM s,y] pos [expr [ANIM_dim IANIM x] - 8] \
   [ANIM_dim IANIM n,y] -arrow last
ANIM_create_line IANIMtoTANIM pos [expr [ANIM_dim IANIM x] + 8] \
   [ANIM_dim IANIM n,y] pos [expr [ANIM_dim TANIM x] + 8] \
   [ANIM_dim TANIM s,y] -arrow last
ANIM_create_line IANIMtoANIM pos [expr [ANIM_dim IANIM x] - 8] \
   [ANIM_dim IANIM s,y] pos [expr [ANIM_dim ANIM x] - 8] [ANIM_dim ANIM n,y] \
   -arrow last
ANIM_create_line ANIMtoIANIM pos [expr [ANIM_dim ANIM x] + 8] \
   [ANIM_dim ANIM n,y] pos [expr [ANIM_dim IANIM x] + 8] [ANIM_dim IANIM s,y] \
   -arrow last

ANIM_create_line TSIMtoTANIM pos [ANIM_dim TSIM e,x] \
   [expr [ANIM_dim TSIM y] + 8] pos [ANIM_dim TANIM w,x] \
   [expr [ANIM_dim TANIM y] + 8] -arrow last
ANIM_create_line TANIMtoTSIM pos [ANIM_dim TANIM w,x] \
   [expr [ANIM_dim TANIM y] - 8] pos [ANIM_dim TSIM e,x] \
   [expr [ANIM_dim TSIM y] - 8] -arrow last

ANIM_textpos toolbus 5 30 w -noreset
ANIM_textpos interfaces 5 100 w -noreset
ANIM_textpos tools 5 170 w -noreset
ANIM_create_text toolbus ToolBus
ANIM_create_text interfaces interfaces
ANIM_create_text tools tools

ANIM_textpos_item SIM SIM s n
ANIM_textpos_item ANIM ANIM s n

ANIM_textpos_line TSIM-ISIM TSIMtoISIM e
ANIM_textpos_line ISIM-TSIM ISIMtoTSIM w
ANIM_textpos_line ISIM-SIM ISIMtoSIM e
ANIM_textpos_line SIM-ISIM SIMtoISIM w

ANIM_textpos_line TANIM-IANIM TANIMtoIANIM e
ANIM_textpos_line IANIM-TANIM IANIMtoTANIM w
ANIM_textpos_line IANIM-ANIM IANIMtoANIM e
ANIM_textpos_line ANIM-IANIM ANIMtoIANIM w

ANIM_textpos_line TSIM-TANIM TSIMtoTANIM n
ANIM_textpos_line TANIM-TSIM TANIMtoTSIM s

proc ANIM_action {atom} {
   if {[regexp {^sim\(control\((.*)\)\)$} $atom match arg1]} {
      ANIM_delete_text ISIM-SIM
      ANIM_deactivate_line ISIMtoSIM
   } elseif {[regexp {^sim\(ack\)$} $atom match]} {
      ANIM_delete_text ISIM-SIM
      ANIM_deactivate_line ISIMtoSIM
   } elseif {[regexp {^sim\((.*)\)$} $atom match arg1]} {
      ANIM_delete_text SIM
      ANIM_delete_text ISIM-SIM
      ANIM_create_text SIM "$arg1"
   } elseif {[regexp {^simint-comm\((.*)\)$} $atom match arg1]} {
```

```
      ANIM_delete_text ISIM-SIM
      ANIM_deactivate_line ISIMtoSIM
      ANIM_delete_text SIM
      ANIM_create_text SIM-ISIM "$arg1"
      ANIM_activate_line SIMtoISIM
   } elseif {[regexp {^intsim-comm\((.*)\)$} $atom match arg1]} {
      ANIM_delete_text TSIM-ISIM
      ANIM_deactivate_line TSIMtoISIM
      ANIM_create_text ISIM-SIM "$arg1"
      ANIM_activate_line ISIMtoSIM
   } elseif {[regexp {^simtb-comm-snd\(tb-term\((.*)\)\)$} $atom match arg1]} {
      ANIM_delete_text SIM-ISIM
      ANIM_deactivate_line SIMtoISIM
      ANIM_create_text ISIM-TSIM "$arg1"
      ANIM_activate_line ISIMtoTSIM
   } elseif {[regexp {^simtb-comm-rec\(tb-term\((.*)\)\)$} $atom match arg1]} {
      if {! [regexp {^ack$} $arg1 match]} {
         ANIM_delete_text TANIM-TSIM
         ANIM_deactivate_line TANIMtoTSIM
      }
      ANIM_create_text TSIM-ISIM "$arg1"
      ANIM_activate_line TSIMtoISIM
   } elseif {[regexp {^tb-comm-msg\(panim, tb-term\((.*)\)\)$} $atom match arg1]} {
      ANIM_delete_text ISIM-TSIM
      ANIM_deactivate_line ISIMtoTSIM
      ANIM_create_text TSIM-TANIM "$arg1"
      ANIM_activate_line TSIMtoTANIM
   } elseif {[regexp {^tb-comm-msg\(psim, tb-term\((.*)\)\)$} $atom match arg1]} {
      ANIM_delete_text IANIM-TANIM
      ANIM_deactivate_line IANIMtoTANIM
      ANIM_create_text TANIM-TSIM "$arg1"
      ANIM_activate_line TANIMtoTSIM
   } elseif {[regexp {^animtb-comm-rec\(tb-term\((.*)\)\)$} $atom match arg1]} {
      # clean ack from animint
      ANIM_delete_text IANIM-TANIM
      ANIM_deactivate_line IANIMtoTANIM
      ANIM_delete_text TSIM-TANIM
      ANIM_deactivate_line TSIMtoTANIM
      ANIM_create_text TANIM-IANIM "$arg1"
      ANIM_activate_line TANIMtoIANIM
   } elseif {[regexp {^animtb-comm-snd\(tb-term\((.*)\)\)$} $atom match arg1]} {
      if {[regexp {^control\(.*\)$} $arg1 match]} {
         ANIM_delete_text TANIM-IANIM
         ANIM_deactivate_line TANIMtoIANIM
      } else {
         ANIM_delete_text ANIM-IANIM
         ANIM_deactivate_line ANIMtoIANIM
      }
      ANIM_delete_text ANIM-IANIM
      ANIM_deactivate_line ANIMtoIANIM
      ANIM_create_text IANIM-TANIM "$arg1"
      ANIM_activate_line IANIMtoTANIM
   } elseif {[regexp {^intanim-comm\((.*)\)$} $atom match arg1]} {
      ANIM_delete_text TANIM-IANIM
      ANIM_deactivate_line TANIMtoIANIM
      ANIM_create_text IANIM-ANIM "$arg1"
      ANIM_activate_line IANIMtoANIM
   } elseif {[regexp {^animint-comm\((.*)\)$} $atom match arg1]} {
      # clean after a reset
      ANIM_delete_text IANIM-ANIM
      ANIM_deactivate_line IANIMtoANIM

      ANIM_delete_text ANIM
      ANIM_create_text ANIM-IANIM "$arg1"
      ANIM_activate_line ANIMtoIANIM
   } elseif {[regexp {^anim\((.*)\)$} $atom match arg1]} {
      ANIM_delete_text IANIM-ANIM
      ANIM_deactivate_line IANIMtoANIM
      ANIM_delete_text ANIM
      ANIM_create_text ANIM "$arg1"
   } elseif {[regexp {^TB-Shutdown$} $atom match]} {
      ANIM_delete_text ISIM-TSIM
      ANIM_deactivate_line ISIMtoTSIM
      ANIM_delete_text IANIM-TANIM
      ANIM_deactivate_line IANIMtoTANIM
   }
}

proc ANIM_choose {atom} {
   if {[regexp {^sim\((.*)\)$} $atom match arg1]} {
      ANIM_add_list SIM $match
```

```
    } elseif {[regexp {^simint-comm\((.*)\)$} $atom match arg1]} {
      ANIM_add_list SIM $match
    } elseif {[regexp {^intsim-comm\((.*)\)$} $atom match arg1]} {
      ANIM_add_list ISIM $match
    } elseif {[regexp {^simtb-comm-snd\(tb-term\((.*)\)\)$} $atom match arg1]} {
      ANIM_add_list ISIM $match
    } elseif {[regexp {^simtb-comm-rec\(tb-term\((.*)\)\)$} $atom match arg1]} {
      ANIM_add_list TSIM $match
    } elseif {[regexp {^tb-comm-msg\(panim, tb-term\((.*)\)\)$} $atom match arg1]} {
      ANIM_add_list TSIM $match
    } elseif {[regexp {^tb-comm-msg\(psim, tb-term\((.*)\)\)$} $atom match arg1]} {
      ANIM_add_list TANIM $match
    } elseif {[regexp {^animtb-comm-rec\(tb-term\((.*)\)\)$} $atom match arg1]} {
      ANIM_add_list TANIM $match
    } elseif {[regexp {^animtb-comm-snd\(tb-term\((.*)\)\)$} $atom match arg1]} {
      ANIM_add_list IANIM $match
    } elseif {[regexp {^intanim-comm\((.*)\)$} $atom match arg1]} {
      ANIM_add_list IANIM $match
    } elseif {[regexp {^animint-comm\((.*)\)$} $atom match arg1]} {
      ANIM_add_list ANIM $match
    } elseif {[regexp {^anim\((.*)\)$} $atom match arg1]} {
      ANIM_add_list ANIM $match
    } elseif {[regexp {^TB-Shutdown|TB-App-Shutdown$} $atom match]} {
      ANIM_add_list TSIM $match
    }
}
```

CONTENTS

LIST OF FIGURES

Technical Reports of the Programming Research Group

Note: These reports can be obtained using the technical reports overview on our WWW site
(`http://www.wins.uva.nl/research/prog/reports/`) or by anonymous ftp to `ftp.wins.uva.nl`,
directory `pub/programming-research/reports/`.

[P9713] B. Diertens. *Simulation and Animation of Process Algebra Specifications.*

[P9711] L. Moonen. *A Generic Architecture for Data Flow Analysis to Support Reverse Engineering.*

[P9710] B. Luttik and E. Visser. *Specification ofRewriting Strategies.*

[P9709] J.A. Bergstra and M.P.A. Sellink. *An Arithmetical Module for Rationals and Reals.*

[P9707] E. Visser. *Scannerless Generalized-LR Parsing.*

[P9706] E. Visser. *A Family of Syntax Definition Formalisms.*

[P9705] M.G.J. van den Brand, M.P.A. Sellink, and C. Verhoef. *Generation of Components for Software Renovation Factories from Context-free Grammars.*

[P9704] P.A. Olivier. *Debugging Distributed Applications Using a Coordination Architecture.*

[P9703] H.P. Korver and M.P.A. Sellink. *A Formal Axiomatization for Alphabet Reasoning with Parametrized Processes.*

[P9702] M.G.J. van den Brand, M.P.A. Sellink, and C. Verhoef. *Reengineering COBOL Software Implies Specification of the Underlying Dialects.*

[P9701] E. Visser. *Polymorphic Syntax Definition.*

[P9618] M.G.J. van den Brand, P. Klint, and C. verhoef. *Re-engineering needs Generic Programming Language Technology.*

[P9617] P.I. Manuel. *ANSI Cobol III in SDF + an ASF Definition of a Y2K Tool.*

[P9616] P.H. Rodenburg. *A Complete System of Four-valued Logic.*

[P9615] S.P. Luttik and P.H. Rodenburg. *Transformations of Reduction Systems.*

[P9614] M.G.J. van den Brand, P. Klint, and C. Verhoef. *Core Technologies for System Renovation.*

[P9613] L. Moonen. *Data Flow Analysis for Reverse Engineering.*

[P9612] J.A. Hillebrand. *Transforming an ASF+SDF Specification into a ToolBus Application.*

[P9611] M.P.A. Sellink. *On the conservativity of Leibniz Equality.*

[P9610] T.B. Dinesh and S.M. Üsküdarlı. *Specifying input and output of visual languages.*

[P9609] T.B. Dinesh and S.M. Üsküdarlı. *The VAS formalism in VASE.*

[P9608] J.A. Hillebrand. *A small language for the specification of Grid Protocols.*

[P9607] J.J. Brunekreef. *A transformation tool for pure Prolog programs: the algebraic specification.*

[P9606] E. Visser. *Solving type equations in multi-level specifications (preliminary version).*

[P9605] P.R. D'Argenio and C. Verhoef. *A general conservative extension theorem in process algebras with inequalities.*

[P9602b] J.A. Bergstra and M.P.A. Sellink. *Sequential data algebra primitives (revised version of P9602).*

[P9604] E. Visser. *Multi-level specifications.*

[P9603] M.G.J. van den Brand, P. Klint, and C. Verhoef. *Reverse engineering and system renovation: an annotated bibliography.*

[P9602] J.A. Bergstra and M.P.A. Sellink. *Sequential data algebra primitives.*

[P9601] P.A. Olivier. *Embedded system simulation: testdriving the ToolBus.*

[P9512] J.J. Brunekreef. *TransLog, an interactive tool for transformation of logic programs.*

[P9511] J.A. Bergstra, J.A. Hillebrand, and A. Ponse. *Grid protocols based on synchronous communication: specification and correctness.*

[P9510] P.H. Rodenburg. *Termination and confluence in infinitary term rewriting.*

[P9509] J.A. Bergstra and Gh. Stefanescu. *Network algebra with demonic relation operators.*

[P9508] J.A. Bergstra, C.A. Middelburg, and Gh. Stefanescu. *Network algebra for synchronous and asynchronous dataflow.*