Appendix A

Constructing a simple clock model with definitive scripts

Note by Meurig Beynon

When creating a model by using a computer, the normal practice is to decide the precise functionality of the model in advance, and to implement from a functional specification. Modelling activity in EM is closer in spirit to creative work in the arts such as making a sculpture or composing a piece of music. The interaction between the artist's state of mind and the work they are creating is dynamic, and the meaning of the work of art is shaped as it is being developed. A simple exercise in modelling a clock can be used to illustrate this idea. The basis for this model is the file of definitions clock.d that comprises a definitive script in which the variables correspond to familiar observables associated with an analogue clock, such as the marks that indicate the time and the lengths and positions of the hour and minute hands. Features of this file include the dependencies that link the positions of the hour and minute hands to the current time. Notice how these are specified in such as way that both the position of the minute hand and the hour hand depend on the time via independent definitions. An alternative way to express this dependency that might more aptly describe the physical relationship between the hands of a mechanical clock would express the position of the minute hand as linked to the position of an internal mechanism, and derive the position of the hour hand by a definition representing the chain of cogs that might connect the hour hand to the minute hand. The file clock.d generates a clock face where the hands are yet to be displayed since the time (clock/t) is yet to be specified. In specifying a time for the clock face, the modeller can adopt many different viewpoints. For instance:

· the modeller may act as a user, setting the clock to the current time

· the modeller may act as a designer, seeking to place the hands in a significant configuration

· the modeller may act as the clockmaker who inserts the clock mechanism. Possible definitions to complement clock.d that correspond to each of these scenarios are given in ext0.d. There are many other instances of potential redefinitions in the file ext0.d. These effect very simple changes to the generated display, but ones that nevertheless can correspond to rich thought processes and changes of perspective on the part of the modeller. In the role of a user, the modeller will consider such issues as starting and stopping the clock, or setting the time to reflect a new time zone. In the role of designer, the modeller may consider the appearance of the clock face, the possibility of adding a second hand, or changing the colour of the hands. The modeller can also act in a role that is outside the scope of either the designer or the user, as when reconfiguring the display to a convenient size for demonstration, or adding physically unrealistic features to the clock. The motivation for making these modifications is to highlight two fundamental ideas behind Empirical Modelling:

1.the construction and structure of scripts mirrors the way in which the modeller construes state-change to occur

2.the modeller's perspective on the script is subject to change from moment to moment, and involves internal human activity (relating to thought processes, situation and agency) that is much richer and more complex than the external computer-based change.

Note how traditional computer programs, in contrast, being optimised to serve particular functions and operate in specific situations, are constructed in ways that do not necessarily give any insight into the fashion in which the programmer construes the world (though this is recognised to be highly relevant to the process of identifying a requirement). Moreover, they are generally intended to exploit the computer's capacity for performing exceedingly complex state-change, and to make the role of the user as clearly defined and as simple to enact as possible.

/*clock.d*/

```
%scout

integer clockwinScale;
point clockwinNW;
point clockwinSE;
point clockwinPos;

window clockwin = {
   type: DONALD
   box: [clockwinNW, clockwinSE]
   pict: “CLOCK”
   xmin: 30
   ymin: 30
   xmax: 370
   ymax: 370
   bgcolor: “white”
   border: 1
};

clockwinPos = {100, 100};
clockwinScale = 2;
point clockwinNW = clockwinPos + {20 * clockwinScale, 20 * clockwinScale};
point clockwinSE = clockwinPos + {180 * clockwinScale, 180 * clockwinScale};
display screen = <clockwin>;

%donald
viewport CLOCK

openshape clock

within clock {
   real sixthpi
   line eleven, ten, nine, eight, seven, six, five, four, three, two, one
   line noon
   point centre
   real radius
   circle edge

   sixthpi = 0.523599
   radius = 150.0
   eleven = rot(noon, centre, -11 * sixthpi)
   ten = rot(noon, centre, -10 * sixthpi)
   nine = rot(noon, centre, -9 * sixthpi)
   eight = rot(noon, centre, -8 * sixthpi)
   seven = rot(noon, centre, -7 * sixthpi)
   six = rot(noon, centre, -6 * sixthpi)
   five = rot(noon, centre, -5 * sixthpi)
   four = rot(noon, centre, -4 * sixthpi)
   three = rot(noon, centre, -3 * sixthpi)
   two = rot(noon, centre, -2 * sixthpi)
   one = rot(noon, centre, -sixthpi)
   noon = [centre+{0, 0.9*radius}, centre+{0,radius}]
   centre = {200, 200}
   edge = circle(centre, radius)

 line minHand, hourHand
 real minAngle, hourAngle
 int t
 minAngle = (pi div 2.0) - float (t mod 60) * (pi div 30.0)
```

```
  hourAngle = (pi div 2.0) - float (t mod 720) * (pi div 360.0)

  minHand = [centre + {0.75*radius @ minAngle}, centre]
  hourHand = [centre + {0.5*radius @ hourAngle}, centre]
  centre = {200, 200}

}

# clock/t is a donald int variable
# representing time elapsed in minutes
# from midnight (1440 per day)
```

/*autotime.e*/

```
/* This eden file illustrates the concept of automating a pattern
of observation. The underlying construal is:
The watch automatically consults the real-world time (via
the action new_time = gettime(); in chk_time), and updates
the number of seconds elapsed since 1/1/70 (recorded in tn)
accordingly.
Issues:
How fast does chk_time act and schedule actions (todo())?
“Accordingly” means “if the time recorded in new_time changes”?
How much real-time elapses between tn = gettime(); and the
initiation of the clock mechanism with first invocation of
inc_time();?
*/

tn = gettime();

/* use a day long absolute time for testing purposes */
func abs_time {
auto h,m,s;
h=$1[3]; m=$1[2]; s=$1[1];
return h*3600+m*60+s;
}

func ext_time {
    auto h,m,s;
h = $1/3600;
m = ($1%3600)/60;
s = $1%60;
return [s,m,h];
}

/* inc_time updates the time second by second */

proc inc_time {
auto tnt, tailtn;
tailtn = [tn[4],tn[5],tn[6],tn[7]];
tnt = abs_time(tn);
for (i=1; i<=$1; i++) {
tnt++; tn = ext_time(tnt)//tailtn;
}
}

old_time = gettime();
proc chk_time : new_time {
todo(“new_time = gettime();”);
```

```
if (old_time[1] != new_time[1])
{
inc_time(1);
old_time = new_time;
}
}

new_time = old_time;
/* need to initialise this in order to trigger chk_time */

/* the role of chk_time as controlling the update of tn through invocations of
inc_time(1)
is  made more apparent if we contrast it with the following variant:

proc chk_time: tn {
 todo("inc_time(1);");
}

which effectively updates the clock as fast as the eden queuing and processing speed
permits
*/

tnsecs is abs_time(tn);


/*ext0.d*/

%scout
# meta-interaction: action by the modeller to change the
# state of the computer model e.g. in order to make more
# space on the screen, by relocating image of clock and
# by shrinking the clock image.

%scout
mag = 1;
clockwinPos = {10,10};

%donald
# extension 0

# clock/t is a donald int variable
# representing time elapsed in minutes
# from midnight (1440 per day)

clock/t = 134
clock/t = 670

# semantics of adding time?
# should a design for a watch have a particular time on it?

# potentially represents aspect of watch state that is beyond user control
# - reflecting actual user agency over time in real-world context

%eden
include("autotime.e");
/* this maintains a variable tnsecs that records number of seconds
elapsed in real-time from an arbitrary initial time: demo by
proc wtnsecs: tnsecs { writeln(tnsecs); }
*/
```

```
_clock_t is abs_time(tn) / 60;/* integer division, so integer result */

/* user agency over whether watch is running, realised by "Interrupt"
on tkeden input interface (construe as e.g. "remove battery") and
touch(&new_time);
(which we can construe as e.g. "insert battery")
*/

/* also have two kinds of watch: those where operation is perceptible
(e.g. via audible tick) and those where operation indiscernible
proc wtnsecs {};
*/

%donald
clock/t = 0
# potentially represents a user of the watch - setting the time
# many possible reasons for setting the time (e.g. watch is fast,
# user is in new time zone, designer wants to view hands overlapping).

%eden
uk_time is tnsecs / 60;
japan_time is uk_time + 480;
_clock_t is japan_time;

%donald
# extension 1

# edge is the circle that defines the outer rim of the clock face
# adding inner circle can be construed as a designer's action

within clock {
circle inner_edge
real width_edge
inner_edge = circle(center, radius - width_edge)
 width_edge = 20.0

# aesthetic issue of where to place the inner edge

width_edge = 10.0
}

# note that such action is construed as changing the watch
# as opposed to changing the state of the watch

# extension 2

# adding a second hand: potentially a design decision, but could also
# be construed as taking account of a hitherto neglected observable

%donald
within clock {
line secHand
real secAngle, size_secHand
secHand = [center + {size_secHand*radius @ secAngle}, center]
size_secHand = 0.8
}

# no position for second hand yet, so not on display
# before we give proper definition of dependency on time, can experiment
# with arbitrary values (meta-interaction by the model constructor
```

```
clock / secAngle = pi

# also might introduce explicit observables for size_minHand etc here
within clock {
real size_minHand
size_minHand = 0.75
minHand = [center + {size_minHand * radius @ minAngle}, center]
}

# now specify the position of the second hand as function of time
%eden
sec_mod_60 is tnsecs % 60;
/* determine how many seconds elapsed since last whole minute time */

%donald
within clock {
secAngle = (pi div 2.0) - float (sec_mod_60!)* (pi div 30.0)
}

%eden
A_clock_secHand = “color=red”;

/* construed as a design feature */

/* compare the following implausible associations of observables */

_clock_size_secHand is (float(sec_mod_60) / 60.0) * _clock_size_minHand;

%donald
# the minute hand comes loose
clock / minAngle = - pi div 2

# repair is difficult, unless have observed correct configuration first
```