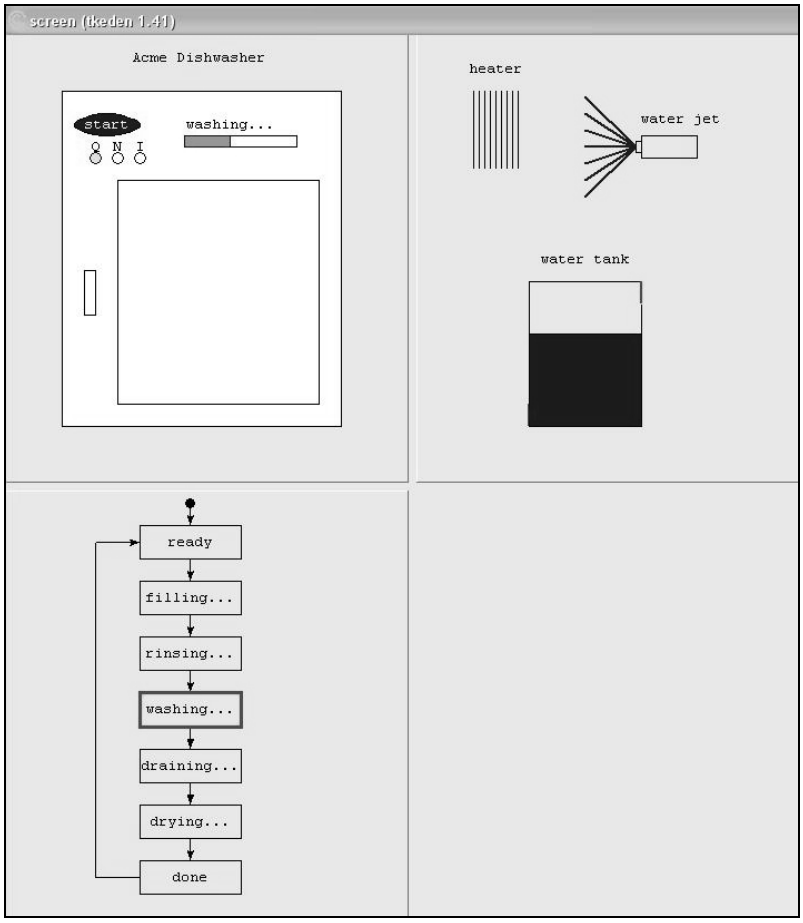
Appendix A: The Dishwasher EM model




```
/* dishwasherModel3-3.s

   The Dishwasher Model

*/


/*************************
     ROLE: A CLOCK
*************************/
%eden
stopCLK = 1;
nextClock = iClock = 0;
proc clocking : iClock, stopCLK {
     if (!stopCLK)
     {
          nextClock++;
          todo("iClock = nextClock;");
     }
}

proc stopClk{
     stopCLK = 1;
}

proc startClk{
     stopCLK = 0;
}
```

```
proc resetClk{
     iClock = nextClock = 0;
}

/*******************************
     ROLE: A MAINTENANCE ENGINEER
*******************************/

/*  unfinished */

/************************
     ROLE: A USER
************************/
%eden
proc startJob{
   startProcess();
}

proc changeMode{
   para char;
   if(char=='Q'){

      mode is quick;
   } else if(char=='N'){
      mode is normal;
   } else if(char=='I'){
      mode is intensive;
   }
}

proc openDoor{
    door is open;
}

proc closeDoor{

    door is closed;
}

/************************
   ROLE: AN ENGINEER
************************/
%eden
/* door */
open is 0;
closed is 1;
door is closed;

/* wash mode */
quick is 1;
normal is 2;
intensive is 3;

mode is normal;
rinseTime is (mode == quick)?1:((mode == normal)?2:8);
washTime is (mode == quick)?1:((mode == normal)?2:8);
dryTime is (mode == quick)?1:((mode == normal)?2:8);

/* washing progress */
ready is "ready";
go is "filling...";
filled  is "rinsing...";
rinsed is "washing...";
washed  is "draining...";
drained is "drying...";
dried   is "done";

progress is ready;

proc startProcess{
    startClk();
    startTime is time();
    progress is go;
}

proc monitorDoor: door{
```

```
    if(door==open){
         writeln("door opened.");
         if(progress==dried)
           progress is ready;
    }
    else{
         startTime is time();
         writeln("door closed.");
    }
}

/*
proc doneWash: iClock{
    if(progress==dried && door==closed){
        progress is done;
        writeln("done.");
    }
}
*/

/* tank */
tankEmpty is 0;
tankFull is 1;
tankSomewater is 2;
valveClosed is 0;
valveOpen is 1;

tankCapacity is 20000;
waterFlowPerSecond is 50;
waterLevel is 1;
waterPercent is float(waterLevel) / float(tankCapacity) * 100;
waterLevelStatus is
(waterLevel==0)?tankEmpty:((waterLevel>=tankCapacity)?tankFull:tankSomewater);
drainValve is (progress==washed && door==closed &&
waterLevelStatus!=tankEmpty)?valveOpen:valveClosed;
fillValve is (progress==go && door==closed &&
waterLevelStatus!=tankFull)?valveOpen:valveClosed;

/*
proc monitorTank: waterLevel, waterFlowPerSecond, drainValve, fillValve {

     writeln("waterlevel:", waterLevel," waterFlowPerSecond:",waterFlowPerSecond,"
drainValve:",drainValve," fillValve:",fillValve);
}
*/

proc waterSimulation: iClock{

     if(fillValve==valveOpen){
         execute("waterLevel is eval(waterLevel+waterFlowPerSecond);");
     }


     if(drainValve==valveOpen && waterLevelStatus!=tankEmpty){
         execute("waterLevel is eval(waterLevel-waterFlowPerSecond);");
         if(waterLevel<0)
             waterLevel is 0;
     }


}



proc fillTank: iClock{
    if(progress==go && door==closed && waterLevelStatus==tankFull){
        progress is filled;
        writeln("filled.");
    }
}

proc drainTank: iClock{
    if(progress==washed && door==closed && waterLevelStatus==tankEmpty){
        progress is drained;
     writeln("drained.");
    }
}
```

```
/* jet */

jetRinseWaterPerSecond is 2;
jetWashWaterPerSecond is 4;
jetSecondPerPulse is 1;

jetSpraying is 0;
jetPulsing is 1;
jetOff is 2;

jetLeft is -1;
jetRight is 1;
jetMiddle is 0;

jetStatus is (progress==filled)?jetSpraying:((progress==rinsed)?jetPulsing:jetOff);
jetDirection is (jetStatus==jetSpraying
||jetStatus==jetOff)?jetMiddle:((jetStatus==jetPulsing &&
(iClock/jetSecondPerPulse)%2==1)?jetRight:jetLeft);

/*
proc monitorJetDirect: jetDirection{
   write(jetDirection);
}
*/

proc rinseDish: iClock{
    if(progress==filled && door==closed){

       execute("waterLevel is eval(waterLevel-jetRinseWaterPerSecond);");
       if(waterLevel<0)waterLevel is 0;


       if(time()-startTime>rinseTime){
         progress is rinsed;
         startTime is time();
         writeln("rinsed.");
       }
    }
}

proc washDish: iClock{

    if(progress==rinsed &&  door==closed){

       execute("waterLevel is eval(waterLevel-jetWashWaterPerSecond);");
       if(waterLevel<0)waterLevel is 0;

       if(time()-startTime>washTime){
     progress is washed;
        startTime is time();
        writeln("washed.");
       }
    }
}

/* heater */

proc dryDish: iClock{

    if(progress==drained && time()-startTime>washTime && door==closed){
     progress is dried;
        startTime is time();
        writeln("dried.");
    }
}

/*****************************************
      DISHWASHER COMPONENTS VISUALISATIONS
*****************************************/
%donald
viewport visualisation

# tank visualisation
rectangle theTank
theTank = rectangle({100,50},{200,50+waterPercent!})
?A_theTank = "color=blue,fill=solid";
```

```
line theTankTop, theTankLeft, theTankRight, theFillValve, theDrainValve
theTankTop = [{100,180},{200,180}]
theTankLeft = [{100,70},{100,180}]
theTankRight = [{200,50},{200,160}]
?theFillValveX is (fillValve==valveOpen)?180:200;
?theDrainValveX is (drainValve==valveOpen)?80:100;
theFillValve = [{theFillValveX!,160},{200,180}]
?A_theFillValve = "color=red,linewidth=2";
theDrainValve = [{theDrainValveX!,50},{100,70}]
?A_theDrainValve = "color=red,linewidth=2";

label theTankLabel
theTankLabel = label("water tank", {150,200})

# heater visualisation
int bargap, barx,bary
line bar1,bar2,bar3,bar4,bar5,bar6,bar7,bar8,bar9
bargap = 5
barx = 50
bary = 350

bar1 = [{barx,bary },{barx,bary-70}]
bar2 = [{barx+bargap,bary },{barx+bargap,bary-70}]
bar3 = [{barx+bargap*2,bary },{barx+bargap*2,bary-70}]
bar4 = [{barx+bargap*3,bary },{barx+bargap*3,bary-70}]
bar5 = [{barx+bargap*4,bary },{barx+bargap*4,bary-70}]
bar6 = [{barx+bargap*5,bary },{barx+bargap*5,bary-70}]
bar7 = [{barx+bargap*6,bary },{barx+bargap*6,bary-70}]
bar8 = [{barx+bargap*7,bary },{barx+bargap*7,bary-70}]
bar9 = [{barx+bargap*8,bary },{barx+bargap*8,bary-70}]

?A_bar1 is (progress==drained)?"linewidth=3,color=red":"";
?A_bar2 is A_bar1;
?A_bar3 is A_bar1;
?A_bar4 is A_bar1;
?A_bar5 is A_bar1;
?A_bar6 is A_bar1;
?A_bar7 is A_bar1;
?A_bar8 is A_bar1;
?A_bar9 is A_bar1;

label theHeaterLabel
theHeaterLabel = label("heater", {barx+20,bary+20})

#jet visualisation
int jetx, jety

jetx = 200
jety = 300

rectangle theJet, theJetHead
theJet = rectangle({jetx,jety+10},{jetx+50,jety-10})
theJetHead = rectangle({jetx-5,jety+5},{jetx,jety-5})

line water0, water1, water2, water3, water4, water5, water6

water0 = [{jetx-5,jety}, {jetx-50,jety+45} ]
water1 = [{jetx-5,jety}, {jetx-50,jety+30} ]
water2 = [{jetx-5,jety}, {jetx-50,jety+15} ]
water3 = [{jetx-5,jety}, {jetx-50,jety} ]
water4 = [{jetx-5,jety}, {jetx-50,jety-15} ]
water5 = [{jetx-5,jety}, {jetx-50,jety-30} ]
water6 = [{jetx-5,jety}, {jetx-50,jety-45} ]

?A_water0 is (progress==rinsed && door==closed &&
jetDirection==jetLeft)?"linewidth=2,color=blue":"color=grey";
?A_water1 is A_water0;
?A_water5 is (progress==rinsed && door==closed &&
jetDirection==jetRight)?"linewidth=2,color=blue":"color=grey";
?A_water6 is A_water5;


?A_water2 is ((progress==filled || progress==rinsed) && door==closed &&
jetStatus==jetSpraying)?"linewidth=2,color=blue":"color=grey";
?A_water3 is A_water2;
?A_water4 is A_water2;
```

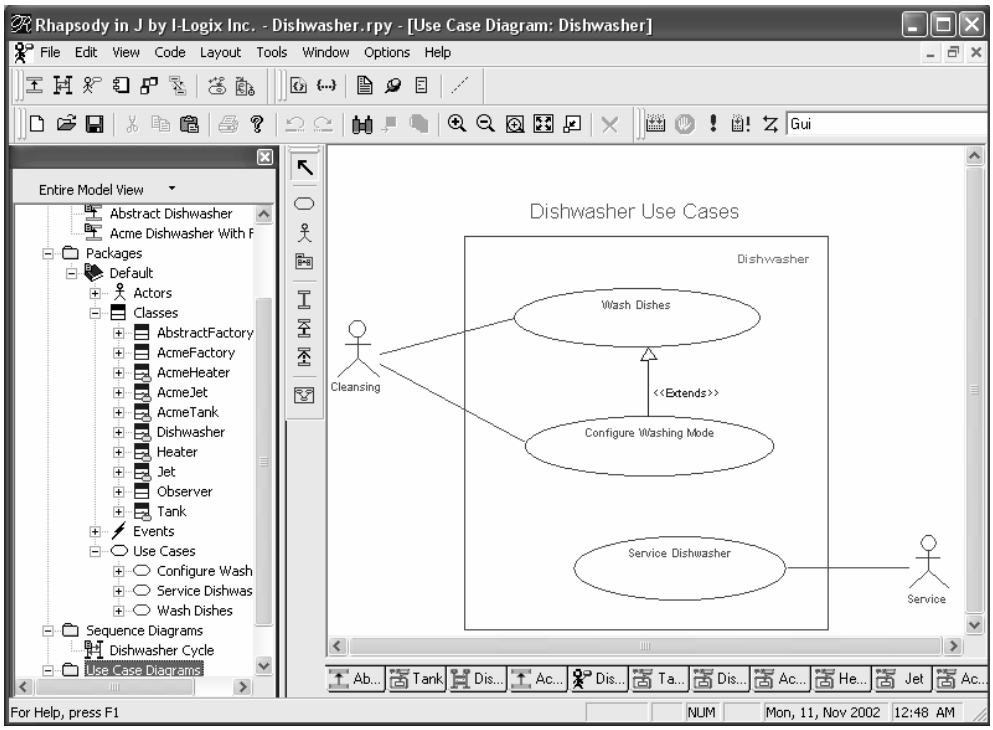
```
label theJetLabel
theJetLabel = label("water jet", {jetx+35,jety+25})


%scout
window visualisationWindow;

visualisationWindow = {
     box: [{370,0}, {730,400}],
     pict:     "visualisation",
     type:     DONALD,
     xmin:     0,
     ymin:     0,
     xmax:     360,
     ymax:     400,
     border:   1,
     sensitive: ON
};

%eden

/****************************************
      STATECHART VISUALISATIONS
****************************************/
%donald
viewport statechart

rectangle stateReady, stateFilling, stateRinsing, stateWashing, stateDraining,
stateDrying, stateDone
label fillingLabel,rinsingLabel,washingLabel,drainingLabel,dryingLabel, readyLabel,
doneLabel

int startx, starty, height, width, gap
point readyin,readyout,
fillin,fillout,rinsein,rinseout,washin,washout,drainin,drainout,dryin,dryout,donein,
doneout
line readytofill, filltorinse, rinsetowash, washtodrain, draintodry, drytodone

startx = 120
starty = 370
gap = 20
height = 30
width = 90


stateReady = rectangle({startx,starty},{startx+width, starty-height})
?A_stateReady is (progress==ready)?"outlinecolor=red,linewidth=3":"";
readyLabel = label(ready!, {startx + width div 2, starty - height div 2})
readyin = {startx + width div 2, starty}
readyout = {startx + width div 2, starty-height}

stateFilling = rectangle({startx,starty-(height+gap)},{startx+width,
starty-(height+gap)-height})
?A_stateFilling is (progress==go)?"outlinecolor=red,linewidth=3":"";
fillingLabel = label(go!, {startx + width div 2, starty-(height+gap) - height div 2})
fillin = {startx + width div 2, starty-(height+gap)}
fillout = {startx + width div 2, starty-(height+gap)-height}

stateRinsing = rectangle({startx,starty-(height+gap)*2},{startx+width,
starty-(height+gap)*2-height})
?A_stateRinsing is (progress==filled)?"outlinecolor=red,linewidth=3":"";
rinsingLabel = label(filled!, {startx + width div 2, starty-(height+gap)*2 - height div
2})
rinsein = {startx + width div 2, starty-(height+gap)*2}
rinseout = {startx + width div 2, starty-(height+gap)*2-height}

stateWashing = rectangle({startx,starty-(height+gap)*3},{startx+width,
starty-(height+gap)*3-height})
?A_stateWashing is (progress==rinsed)?"outlinecolor=red,linewidth=3":"";
washingLabel = label(rinsed!, {startx + width div 2, starty-(height+gap)*3 - height div
2})
washin = {startx + width div 2, starty-(height+gap)*3}
washout = {startx + width div 2, starty-(height+gap)*3-height}

stateDraining = rectangle({startx,starty-(height+gap)*4},{startx+width,
starty-(height+gap)*4-height})
```

```
?A_stateDraining is (progress==washed)?"outlinecolor=red,linewidth=3":"";
drainingLabel = label(washed!, {startx + width div 2, starty-(height+gap)*4 - height div
2})
drainin = {startx + width div 2, starty-(height+gap)*4}
drainout = {startx + width div 2, starty-(height+gap)*4-height}

stateDrying = rectangle({startx,starty-(height+gap)*5},{startx+width,
starty-(height+gap)*5-height})
?A_stateDrying is (progress==drained)?"outlinecolor=red,linewidth=3":"";
dryingLabel = label(drained!, {startx + width div 2, starty-(height+gap)*5 - height div
2})
dryin = {startx + width div 2, starty-(height+gap)*5}
dryout = {startx + width div 2, starty-(height+gap)*5-height}

stateDone = rectangle({startx,starty-(height+gap)*6},{startx+width,
starty-(height+gap)*6-height})
?A_stateDone is (progress==dried)?"outlinecolor=red,linewidth=3":"";
doneLabel = label(dried!, {startx + width div 2, starty-(height+gap)*6 - height div 2})
donein = {startx + width div 2, starty-(height+gap)*6}
doneout = {startx + width div 2, starty-(height+gap)*6-height}

readytofill = [readyout, fillin]
?A_readytofill ="arrow=last";

filltorinse = [fillout, rinsein]
?A_filltorinse ="arrow=last";

rinsetowash = [rinseout, washin]
?A_rinsetowash ="arrow=last";

washtodrain = [washout, drainin]
?A_washtodrain ="arrow=last";

draintodry = [drainout, dryin]
?A_draintodry ="arrow=last";

drytodone = [dryout, donein]
?A_drytodone ="arrow=last";

# the init dot and arrow
point stateInit
stateInit = {readyin.1,readyin.2+20}

circle initCircle
initCircle = circle(stateInit, 4)
?A_initCircle="fill=solid";

line initLine
initLine = [stateInit,readyin]
?A_initLine ="arrow=last";

#the return routine
line l1,l2,l3

l1 = [{startx, starty-(height+gap)*6 - height div 2}, {startx-40, starty-(height+gap)*6
- height div 2}]
l2 = [{startx-40, starty-(height+gap)*6 - height div 2}, {startx-40,starty-height div
2}]
l3 = [{startx, starty- height div 2}, {startx-40, starty- height div 2}]
?A_l3 ="arrow=first";

%scout
window statechartWindow;

statechartWindow = {
     box: [{0,410}, {360,810}],
     pict:     "statechart",
     type:     DONALD,
     xmin:     0,
     ymin:     0,
     xmax:     360,
     ymax:     400,
     border:   1,
     sensitive: OFF
};

%eden
```

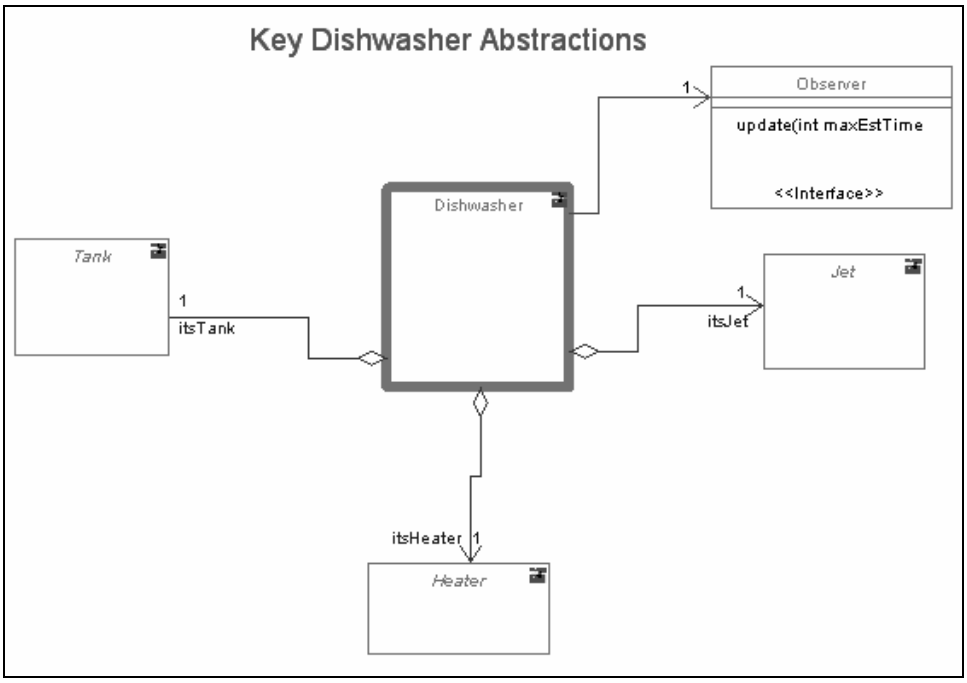
```
/*******************************
  ROLE: AN INTERFACE DESIGNER
*******************************/

%donald
viewport interface

#interface title(not part of the interface)
label titleLabel
titleLabel=label("Acme Dishwasher", {173,380})

#shape of the machine
rectangle machineShape
machineShape=rectangle({50,50},{300,350})
?A_machineShape = "color=white,fill=solid";

#the start button
ellipse startButton
startButton=ellipse({90,320},{90,330},{120,320})
?A_startButton = "outlinecolor=black,color=blue,fill=solid";

label startLabel
startLabel=label("start", {90,320})
?A_startLabel = "color=white";

#the door
rectangle theInside
theInside=rectangle({100,70},{280,270})
?A_theInside = "color=grey50,fill=solid";

rectangle theDoor
?theDoorX is (door==closed)?100:270;
theDoor=rectangle({theDoorX!,70},{280,270})
?A_theDoor = "color=white,fill=solid";

rectangle theHandle
theHandle=rectangle({70,150},{80,190})


#the progress bar

%eden
func convertProgressNum{
     auto result;
     result = 0;

     if($1==filled)result=1;
     else if($1==rinsed)result=2;
     else if($1==washed)result=3;
     else if($1==drained)result=4;
     else if($1==dried)result=5;

     return result;
}
progressNum is convertProgressNum(progress);

%donald
rectangle theBarBackground
theBarBackground=rectangle({160,300},{260,310})

rectangle theBar
theBar=rectangle({160,300},{160+20*progressNum!,310})
?A_theBar = "color=green,fill=solid";

#the progress label
label progressLabel
progressLabel=label(progress!, {200,320})
?A_progressLabel = "color=black";

#mode buttons
circle quickButton
quickButton = circle({80,290},5)
?A_quickButton is (mode==quick)?"color=yellow,fill=solid":"";

label qLabel
qLabel = label("Q",{80,300})
```

```
circle normalButton
normalButton = circle({100,290},5)
?A_normalButton is (mode==normal)?"color=yellow,fill=solid":"";

label nLabel
nLabel = label("N",{100,300})

circle intensiveButton
intensiveButton = circle({120,290},5)
?A_intensiveButton is (mode==intensive)?"color=yellow,fill=solid":"";

label iLabel
iLabel = label("I",{120,300})

%eden

proc mouseMonitor:interfaceWindow_mouse{
    auto x,y;
    if(interfaceWindow_mouse[2]==4){
         x=interfaceWindow_mouse[4];
         y=interfaceWindow_mouse[5];
         if(x>=60 && x<=120 && y>=310 && y<=330)
          startJob();
         /* door handle*/
         else if(x>=70 && x<=80 && y>=150 && y<=190){
          if(door==closed) door is open;
                else door is closed;
               }
         /*quick button*/
         else if(x>=75 && x<=85 && y>=285 && y<=295){
              mode is quick;
              }
         /*normal button*/
         else if(x>=95 && x<=105 && y>=285 && y<=295){
              mode is normal;
              }
         /*intensive button*/
         else if(x>=115 && x<=125 && y>=285 && y<=295){
              mode is intensive;
              }
         else
            write("mouse x:",int(x)," y:",int(y),"\n");

    }
}

%scout
window interfaceWindow;

interfaceWindow = {
     box: [{0,0}, {360,400}],
     pict:     "interface",
     type:     DONALD,
     xmin:     0,
     ymin:     0,
     xmax:     360,
     ymax:     400,
     border:   1,
     sensitive: ON
};

%eden
/*****************************
   COMBINE DONALD VIEWPORTS
*****************************/
%scout

display d;
d=<interfaceWindow/visualisationWindow/statechartWindow>;
screen= d;

%eden
```

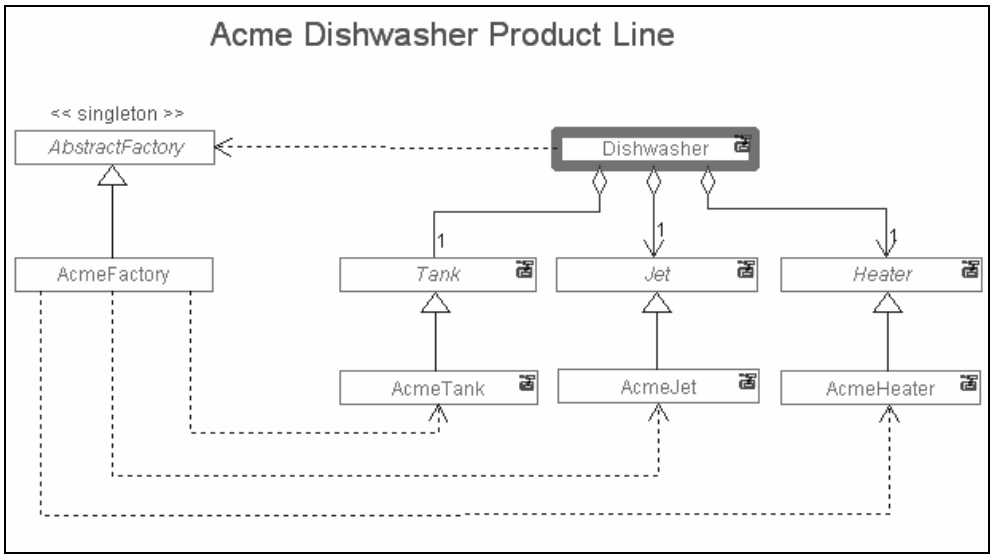
Appendix B: The Dishwasher UML model



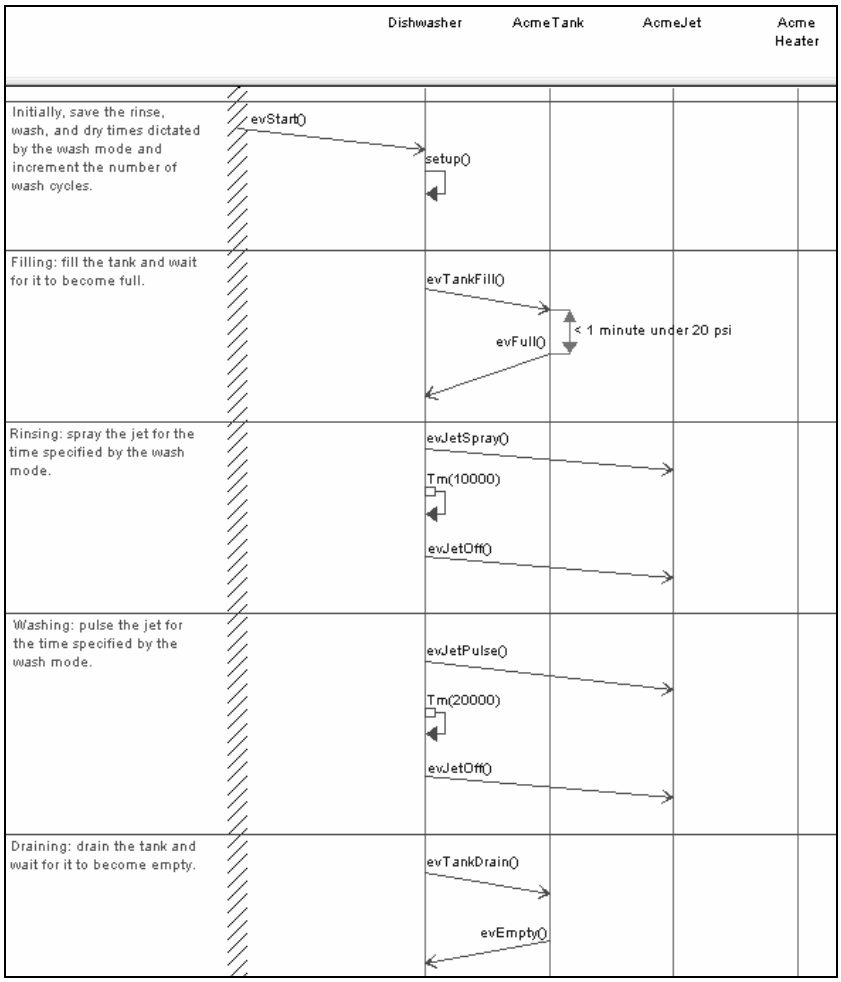
Rhapsody interface with the dishwasher Use Case



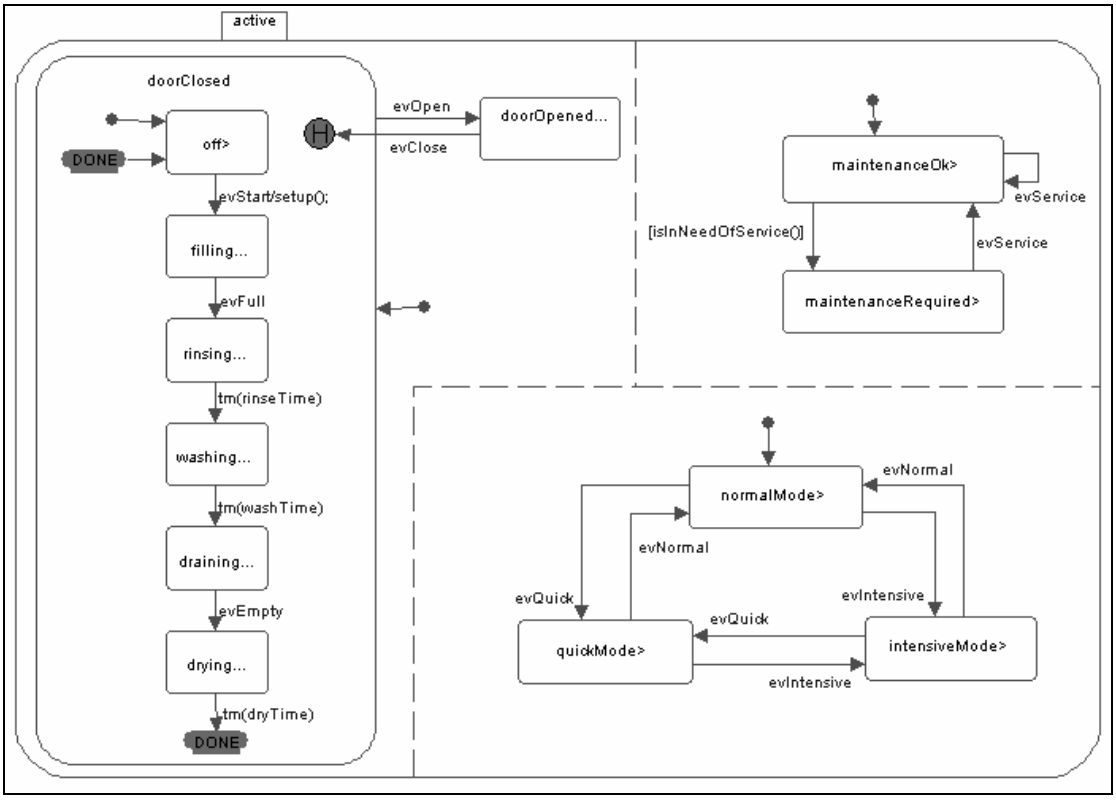
Class Diagram



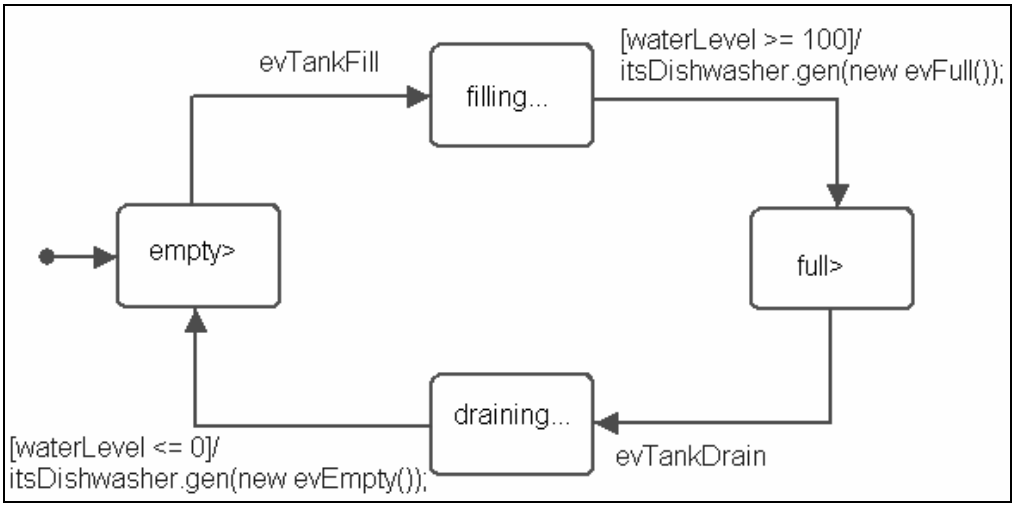
Class Diagram



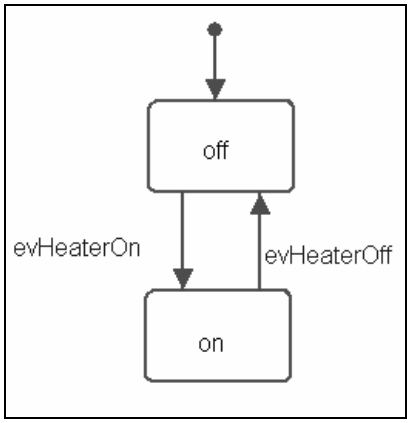
Sequence Diagram



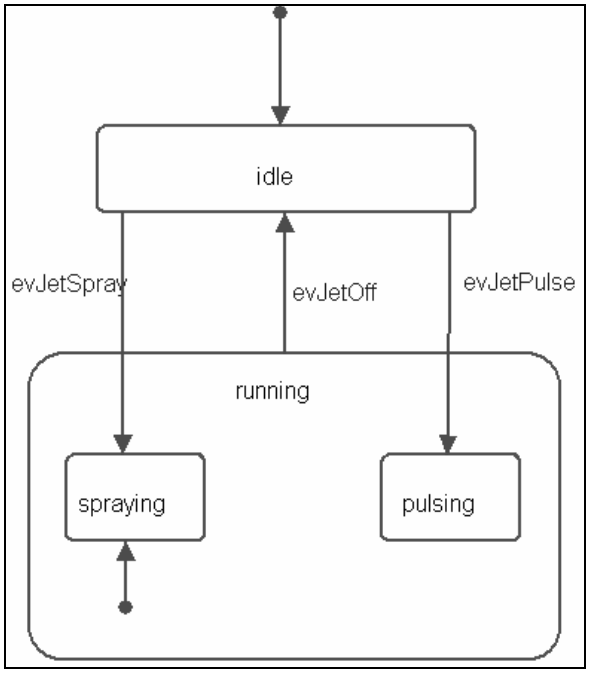
State Chart Diagram for the Dishwasher Class



State Chart Diagram for the Tank Class



State Chart Diagram for the Heater class



State Chart Diagram for the Jet class

Appendix C: An LSD account for the Dishwasher model

```
/*****************************************
    An LSD account for the dishwasher model
*****************************************/

Types:

     modeType  : enum(quick,normal,intensive)
     doorStatusType : enum(open,closed)
     progressType   : enum(go,filled,rinsed,washed,drained,dried,finished)
     tankStatusType : enum(tankEmpty,tankFull,tankSomewater)
     valveType : enum(valveClosed,valveOpen)
     jetStatusType  : enum(jetSpraying,jetPulsing,jetOff)
     jetDirectionType: enum(jetLeft,jetRight,jetMiddle)

agent Clock{
  state:
     (second)  time
     (bool)         pause
  protocol:
     pause == false -> time = time + 1
}

agent Dishwasher{
  state:
     (modeType)     mode
     (progressType) progress = go
     (second)  rinseTime
     (second)  washTime
     (second)  dryTime
     (second)  startTime = time
  oracle:
     time, door, waterLevelStatus, waterLevel, progress, mode
  derivate:
     rinseTime = (mode == quick)?1:((mode == normal)?2:8)
     washTime = (mode == quick)?1:((mode == normal)?2:8)
     dryTime = (mode == quick)?1:((mode == normal)?2:8)
     LIVE = progress != finished
  protocol:
     door == open && progress == dried -> progress = finished
     door == closed -> startTime = time
     progress==go && door==closed && waterLevelStatus==tankFull -> progress

= filled
     progress==washed && door==closed && waterLevelStatus==tankEmpty ->

progress = drained
     progress==filled && door==closed && (time-startTime) > rinseTime ->

progress = rinsed; startTime = time
     progress==rinsed && door==closed && (time-startTime) > rinseTime ->

progress = washed; startTime = time
     progress==drained && door==closed && (time-startTime) > rinseTime ->

progress = dried; startTime = time
}

agent Tank{
  state:
     (real)    tankCapacity
     (real)    waterFlowPerSecond
     (real)    waterLevel
     (real)    waterPercent
  oracle:
```

```
     progress, door, time
  derivate:
     waterPercent = waterLevel / tankCapacity * 100
     waterLevelStatus = (waterLevel==0)?tankEmpty:((waterLevel>=

tankCapacity)?tankFull:tankSomewater)
     drainValve = (progress==washed && door==closed && waterLevelStatus!=

tankEmpty)?valveOpen:valveClosed
     fillValve = (progress==go && door==closed && waterLevelStatus!=

tankFull)?valveOpen:valveClosed
  protocol:
     fillValve == valveOpen -> waterLevel = waterLevel + waterFlowPerSecond
     drainValve == valveClose && waterLevel > 0 -> waterLevel = waterLevel

- waterFlowPerSecond

}


agent Jet{
  state:
     (real)    jetRinseWaterPerSecond
     (real)    jetWashWaterPerSecond
     (real)    jetSecondPerPulse
     (jetStatusType)          jetStatus
     (jetDirectionType)  jetDirection

  oracle:
     progress, time, door
  derivate:
     jetStatus = (progress==filled)?jetSpraying:((progress==rinsed)?

jetPulsing:jetOff)
     jetDirection = (jetStatus==jetSpraying ||jetStatus==jetOff)?jetMiddle

:((jetStatus==jetPulsing && (time/jetSecondPerPulse)%2==1)?jetRight:jetLeft)
  protocol:
     progress==filled && door==closed -> waterLevel-jetRinseWaterPerSecond
     progress==rinsed && door==closed -> waterLevel-jetWashWaterPerSecond
}

agent Heater{
  state:
     (bool)    heaterOn
  oracle:
     progress
  derivate:
     heaterOn = progress == drained
}

agent Door{
  state:
     (doorStatusType)    door
  handle:
     pause
  protocol:
     door == open -> pause = true
     door == closed -> pause = false
}

agent User{
  oracle:
     Dishwasher.LIVE
  handle:
     mode, door
  protocol:
     !Dishwasher.LIVE -> Dishwasher()
     true -> mode = quick
     true -> mode = normal
     true -> mode = intensive
     true -> door = open
     true -> door = closed
}
```

Appendix D: The CPPS for a Crossnumber problem

```
/* CPPS for the Crossnumber problem */

grid =  [['x',' ',' ',' ',' '],
         [' ',' ',' ','x',' '],
         [' ','x',' ','x',' '],
         [' ','x',' ',' ',' '],
         [' ',' ',' ',' ','x']
        ];

numbers is ["1234","2125","26242","3992","4998","356","478","15","75"];

a1 is grid[1][1]; a2 is grid[1][2]; a3 is grid[1][3]; a4 is grid[1][4]; a5 is grid[1][5];
b1 is grid[2][1]; b2 is grid[2][2]; b3 is grid[2][3]; b4 is grid[2][4]; b5 is grid[2][5];
c1 is grid[3][1]; c2 is grid[3][2]; c3 is grid[3][3]; c4 is grid[3][4]; c5 is grid[3][5];
d1 is grid[4][1]; d2 is grid[4][2]; d3 is grid[4][3]; d4 is grid[4][4]; d5 is grid[4][5];
e1 is grid[5][1]; e2 is grid[5][2]; e3 is grid[5][3]; e4 is grid[5][4]; e5 is grid[5][5];

blocks  is [a1,b4,c2,c4,d2,e5];

position1 is [a2,a3,a4,a5];
position2 is [b1,b2,b3];
position3 is [d3,d4,d5];
position4 is [e1,e2,e3,e4];
position5 is [b1,c1,d1,e1];
position6 is [a2,b2];
position7 is [a3,b3,c3,d3,e3];
position8 is [d4,e4];
position9 is [a5,b5,c5,d5];

ok1 is containsString(numbers, digitsToString(position1));
ok2 is containsString(numbers, digitsToString(position2));
ok3 is containsString(numbers, digitsToString(position3));
ok4 is containsString(numbers, digitsToString(position4));
ok5 is containsString(numbers, digitsToString(position5));
ok6 is containsString(numbers, digitsToString(position6));
ok7 is containsString(numbers, digitsToString(position7));
ok8 is containsString(numbers, digitsToString(position8));
ok9 is containsString(numbers, digitsToString(position9));


set1 is findNumberWithConstraints(numbers, digitsToString(position1));
set2 is findNumberWithConstraints(numbers, digitsToString(position2));
set3 is findNumberWithConstraints(numbers, digitsToString(position3));
set4 is findNumberWithConstraints(numbers, digitsToString(position4));
set5 is findNumberWithConstraints(numbers, digitsToString(position5));
set6 is findNumberWithConstraints(numbers, digitsToString(position6));
set7 is findNumberWithConstraints(numbers, digitsToString(position7));
set8 is findNumberWithConstraints(numbers, digitsToString(position8));
set9 is findNumberWithConstraints(numbers, digitsToString(position9));


stuck is set1#==0 || set2#==0 || set3#==0 || set4#==0 || set5#==0 || set6#==0 || set7#==0
|| set8#==0 || set9#==0;
solved is ok1 && ok2 && ok3 && ok4 && ok5 && ok6 && ok7 && ok8 && ok9;

proc monitorOks: ok1, ok2, ok3, ok4, ok5, ok6, ok7,ok8,ok9 {
     writeln("oks:",ok1,ok2,ok3,ok4,ok5,ok6,ok7,ok8,ok9);
}

proc monitorProgress:solved, stuck{
     writeln("stuck:", stuck);
     writeln("solved:", solved);
}

proc monitorSets: set1,set2,set3,set4,set5,set6,set7,set8,set9{
```

```
    writeln("ok:",ok1," set1:",set1);
    writeln("ok:",ok2," set2:",set2);
    writeln("ok:",ok3," set3:",set3);
    writeln("ok:",ok4," set4:",set4);
    writeln("ok:",ok5," set5:",set5);
    writeln("ok:",ok6," set6:",set6);
    writeln("ok:",ok7," set7:",set7);
    writeln("ok:",ok8," set8:",set8);
    writeln("ok:",ok9," set9:",set9);

}

proc monitorGrid: grid{
    writeln("--------");
    writeln(grid[1]);
    writeln(grid[2]);
    writeln(grid[3]);
    writeln(grid[4]);
    writeln(grid[5]);
}

proc monitorNumbers: numbers{
    writeln("numbers:",numbers);
}

func containsString{
    para numbers, thestring;
    auto i;
    for(i=1;i<=numbers#;i++){
      if(numbers[i]==thestring)return 1;
    }
    return 0;
}

func digitsToString{
     para digits;
     auto result, i;

     result="";
     for(i=1;i<=digits#;i++){
        result = result // digits[i];
     }

     return result;
}


proc assign{
    para row, column, direction, number;

    auto i;
    autocalc=0;
    for(i=0;i<number#;i++){
       if(direction==0) grid[row][column+i] = number[i+1];
       else if(direction==1) grid[row+i][column] = number[i+1];
    }
    autocalc=1;
}

/* observations of the given set of numbers */
lengthStatistics is extractLengthStatistics(numbers);

func extractLengthStatistics{
     para numbers;
     auto result,i;

     result=[0,0,0,0,0];
     for(i=1;i<=numbers#;i++){
        result[numbers[i]#]++;
     }
     return result;
}

proc monitorLengthStatistics: lengthStatistics{
     writeln("lengthStatistics:", lengthStatistics);
}
```

```
/* strategies */

func findNumberWithConstraints{
     para numbers, constraints;
     auto result,i,j,statisfied;

     result=[];

     for(i=1;i<=numbers#;i++){
        if(numbers[i]# == constraints#){

             statisfied=1;

             for(j=1;j<=numbers[i]#;j++){
                if(constraints[j]!=' '){
                   if(numbers[i][j]!=constraints[j]){
                       statisfied=0;
                       break;
                   }
                }
             }
             if(statisfied==1)
                result = result // [numbers[i]];
        }
     }
     return result;
}
```

Appendix E: Some useful information from the literature

Table 1: Assumptions and ideals of methodical and amethodical texts

[Tru00] Table 1: Assumptions and ideals of methodical and amethodical texts.

Privileged methodical text	**Marginalized amethodical text**
1. Information systems development is a managed, controlled process idealizing logical decomposition reductionism	2. information systems development is random, opportunistic process driven by accident idealizing holism creativity
3. Information systems development is a linear, sequential process idealizing temporal causal chain	4. Information systems development processes are simultaneous, overlapping and there are gaps idealizing fragmentation parallelism disconnectedness
5. Information systems development is a replicable, universal process idealizing generalization consistency formalisms	6. Information systems development occurs in completely unique and idiographic forms idealizing choice change adhocracy
7. Information systems development is a rational, determined, and goal-driven process idealizing goal predetermination process predetermination human cooperation	8. Information systems development is negotiated, compromised and capricious idealizing conflict social constructivism human independence

Table 2: The rational problem solving paradigm and the reflect-in-action paradigms summarized

[Dor95] Figure 1: The rational problem solving paradigm and the reflect-in-action paradigms summarized.

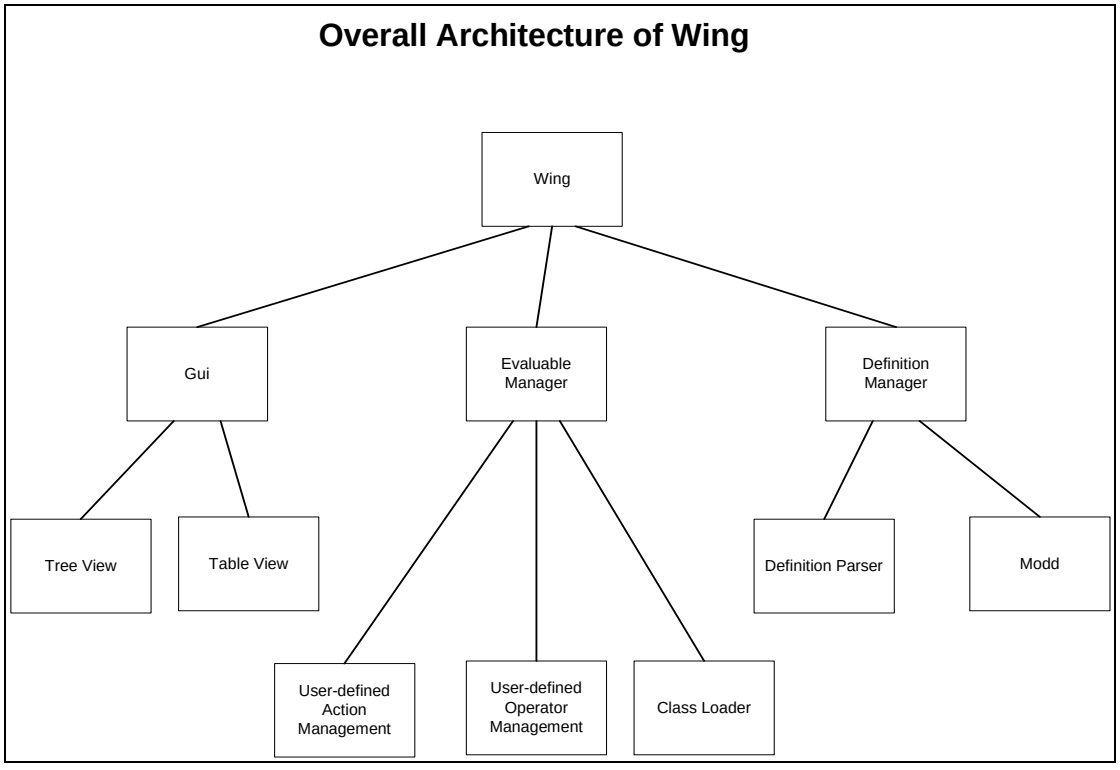
	Rational problem solving	**Reflection-in-action**
Designer	information processor (in an objective reality)	person constructing his/her reality
Designer problem	ill defined, unstructured	essentially unique
Design process	a rational search process	a reflective conversation
Design knowledge	knowledge of design procedures and 'scientific' laws	artistry of design: when to apply which procedure/piece of knowledge
Example/Model	optimization theory, the natural sciences	art/the social sciences

Appendix F: Technical overview of WING

We use Java (JDK1.4) as the programming language to develop WING because of its cross-platform capability and richness of graphics and user-interface library. The interpreter is developed by using JavaCC 0.7. The windowing and graphics functionality is implemented by using Java Foundation Class 1.1.

1. Overall architecture

The diagram showed below is an abstract view of the overall architecture of WING. Each rectangle box represents an abstract functionality to its lower level down to the hierarchical structure.



From the above diagram, WING can be divided into three main components namely *GUI*, *Definition Manager* and *Evaluable Manager*. The *GUI* component provides a user-friendly graphic user interface for user to interact with the system. The most important parts of are:

- *Tree View* – provides a directory tree-like view of definitions and containers, and

allows the user to select a definition or a containers through navigating the tree structure by using mouse pointer.

- *Table View* – visualises definitions in the current container as a table of cells like a spreadsheet. It allows interactive definition editing by just double-clicking the corresponding cell.

The *Evaluable Manager* provides the following functions:

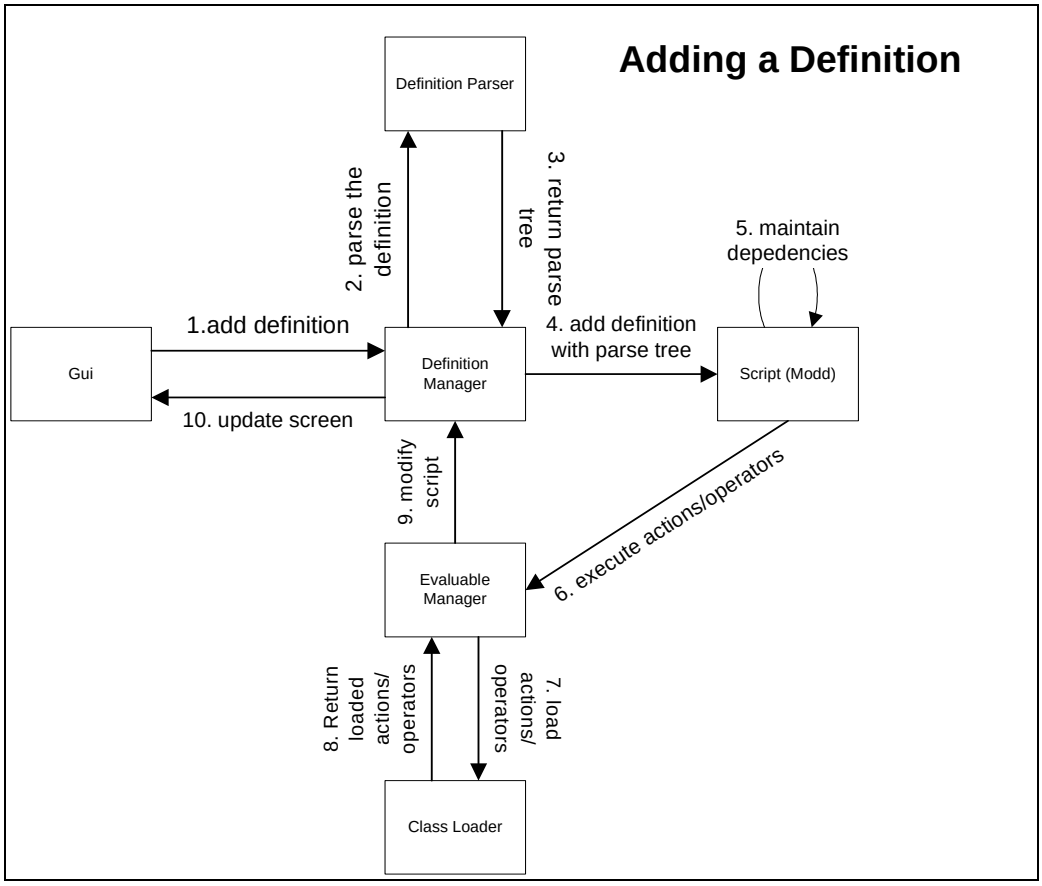
- *User-defined Action Management* – allows user to add, change, delete, compile, load and save actions.
- *User-defined Operator Management* – allows user to add, change, delete, compile, load and save user-defined operators.
- *Class Loader* – allows user to compile and bind Java actions and operators dynamically to the system.

The *Definition Manage*r provides basic functions to manipulate the definitions. It contains the following important components:

- Definition Parser – builds parse tree for each definition. A parse tree is the main communication entity between the system and Modd.
- Modd (Maintainer of Dynamic Dependency) API – maintains dependencies between definitions (developed by Gehring in the EM research group [Geh98])

2. Adding a definition

Referring to the architecture described in the last section, here is a scenario of adding a definition into the system:



As shown in the above diagram, The typical steps of adding a definition into the system are:

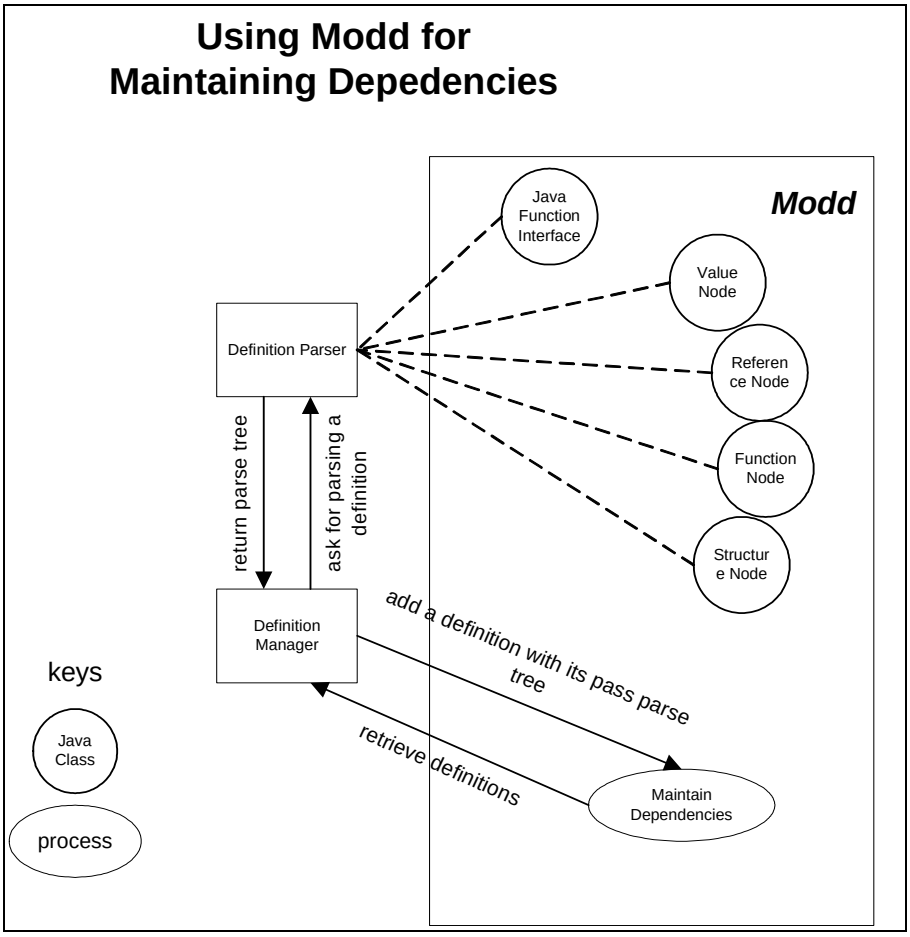
1. GUI passes the definition to the Definition Manager.
2. Definition Manager passes the definition to the Definition Parser
3. Definition Parser builds a parse tree for the definition and returns it to Definition Manager. (Assuming that there is no parse error for this definition)
4. Definition Manager adds the definition with its parse tree to the Script(the most import class in Modd)
5. Script maintains the dependencies among all definitions by evaluating a set of definitions that should be updated (the algorithm determining which definition should be updated in Modd implemented by Gehring)
6. Evaluating the definitions in Modd triggers some user-defined actions and operators that will be executed by Evaluable Manager.
7. Evaluable Manager asks ClassLoader to load the actions and operators from files.
8. ClassLoader returns the loaded actions and operators to Evaluable Manager.
9. Evaluable Manager executes the actions and operators. Actions will be executed

according to their priorities. Some of the actions may request further modification of the script. The requests are passed to Definition Manager.

10. GUI updates the screen in react to changes of the script.

3. Using Modd for maintaining dependencies

WING uses Modd to maintain the dependencies among definitions. The following diagram shows the main entities of Modd:



As shown in the above diagram, the Definition Parser uses a variety of parse tree nodes provided by Modd to build a parse tree for each definition. The tree nodes are implemented as Java classes. Functionalities of different kinds of node are:

- Value node – stores the value of a certain data type.
- Reference node – stores a reference to a definition.
- Function node – stores a function implementation. In WING, it is either an action

or an operator.

- Structure node – stores a list structure similar to EDEN. Since Structure node is still under development, WING implements its own list structure called vectors.

All parse trees that built by using the nodes above are passed to Modd from Definition Manger at run-time. Modd uses the parse tree actions stored in Function node to update the definitions, i.e. to maintain dependencies of definitions.

The parse tree actions stored in Function Node are implemented by using Java Function Interface. Therefore, the task that should be done before we could use Modd is to implement Function Interface for every operator of all data types. All Java files started with prefix "F_" in Appendix of code listing are for this purpose.

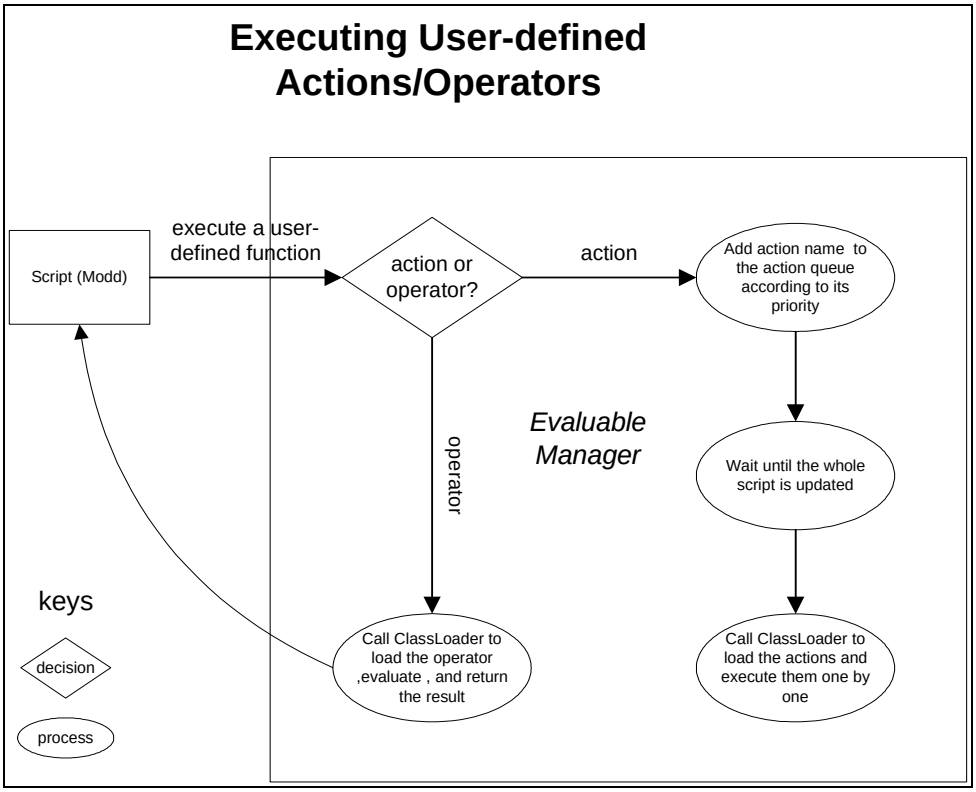
A guide for how to use Modd written by the author of Modd could be found in [Geh98]. However, we summaries the steps to use Modd related to the implementation of WING are as follows:

1. Identify data types and operators.
2. Implement data types using Java.
3. Implement operators by using Function Interface provided by Modd.
4. Implement parse actions in Definition Parser to build parse trees using:
   - Data type implemented in step 2.
   - Function Interface implemented in step 3.
   - Tree nodes provide by Modd.

At run-time, WING's Definition Manager passes parse trees generated by Definition Parser to the Modd for maintaining the dependencies among definitions.

4. Executing user-defined actions/operators

One of the unique features of WING is its capability to allow the user to implement new actions and operators using Java. The following diagram shows procedures to execute an action or an operator:



We summarise the essential points in the diagram as follows (note that Script is a class in Modd that maintains dependencies between definitions):

- When Script encounters a user-defined function name, it will pass the name and corresponding parameters to Evaluable Manager of WING for executing the function.
- Evaluable Manager determines whether the function is an action or an operator.
- If it is an operator, Evaluable Manager calls ClassLoader to load the operator, then execute the operator. The result is returned back to the Script.
- If it is an action, Evaluable Manager adds the action name to the actions queue according to the priority of the action. The Evaluable Manager waits until all definitions are updated. Then, it calls ClassLoader to execute actions in the queue one by one.

In the next two sections, two very useful Java programming techniques are documented. They are the methods used by WING for dynamic compiling and loading of actions/operators. They could be served as reference to implement similar functionality for other Java definitive systems in the future.

5. Dynamic compiling of Java code

To compile Java code at run-time, a class needs to have the following line in its implementation:-

```
import sun.tools.javac.Main;
```

Then, given name of the Java file, we could compile it by using following codes:-

```
boolean   state;
Main compiler=new Main(System.out, "javac " +path);
state = compiler.compile(filename);
```

where `path` is the directory name where the Java file is located. After execution of above codes, the value of state indicates whether the compilation is successful or not. If the value of state is true, the compilation is successful. If the value of state is false, there is parser error. The error messages is displayed in the standard output.

6. Dynamic loading of Java class

Techniques for dynamic loading of Java class is difficult, however McManis, in [Mcm97], provides a template to implement a Java class loader. WING modifies the template to achieve dynamic loading of actions and operators. The implementation of WINGClassLoader could be found in the file WINGClassLoader.java. The main steps for implementing a class loader are summarised as below:

1. extend the java.lang.ClassLoader.
2. implement the abstract method loadClass(). The flow of this method is as follows:
   - check if the class name is valid or not
   - check if the class has already been loaded
   - check if the class is a "system class"
   - fetch the class from class loader's repository if possible
   - define the class for the Virtual Machine
   - resolve the class
   - return the class to the caller

For details, please refer to WINGClassLoader.java.

7. BNF of the definition notation of WING

The DefinitionParser of WING is generated by using JavaCC. The BNF of it is as follows: -

```
one_line  ::=  logical <EOL>
               |<EOL>
               |<EOF>
logical   ::=  relation ( ( <AND> | <OR> | <XOR> ) relation )*
relation  ::=  sum ( ( <SMALLER> | <SMALLER_EQUAL> | <EQUAL> |
     <GREATER>
               | <GREATER_EQUAL> | <NOT_EQUAL> ) sum )*
vector    ::=  "{" logical ( "," logical )* "}"
string    ::=  <STRING>
sum       ::=  term ( ( <PLUS> | <MINUS> ) term )*
term      ::=  exp ( ( <MULTIPLY> | <DIVIDE> ) exp )*
exp       ::=  unary ( <EXP> exp )*
unary          ::=  <MINUS> select
               |select
select    ::=  element ( <SELECT> select )*
element   ::=  <CONSTANT>
               |<TRUE>
               |<FALSE>
               |function
               |"(" logical ")"
               |vector
               |string
function  ::=  <ID> "(" ( logical ( "," logical )* )? ")"
               |<ID>
```

The functionality of DefinitionParser is only to parse the left hand side of a definition and build parse tree by using nodes provided by Modd. Other parts of notation syntax such as for variable name is done by GUI of WING.

8. Reference

[Geh98] D. K. Gehring, *Modd documentation*, Online at: http://www.dcs.warwick.ac.uk/~gehring/modd, 1998.

[Mcm97] C. McManis, The basics of Java class loaders, *Java In Depth*, Online at: http://www.javaworld.com, 1997.

Appendix G: Technical Overview of EME

EME is a MS Windows application developed using Visual C++ 6.0. It uses the Microsoft Foundation Classes (MFC). Many classes in the implementation are derived from MFC.

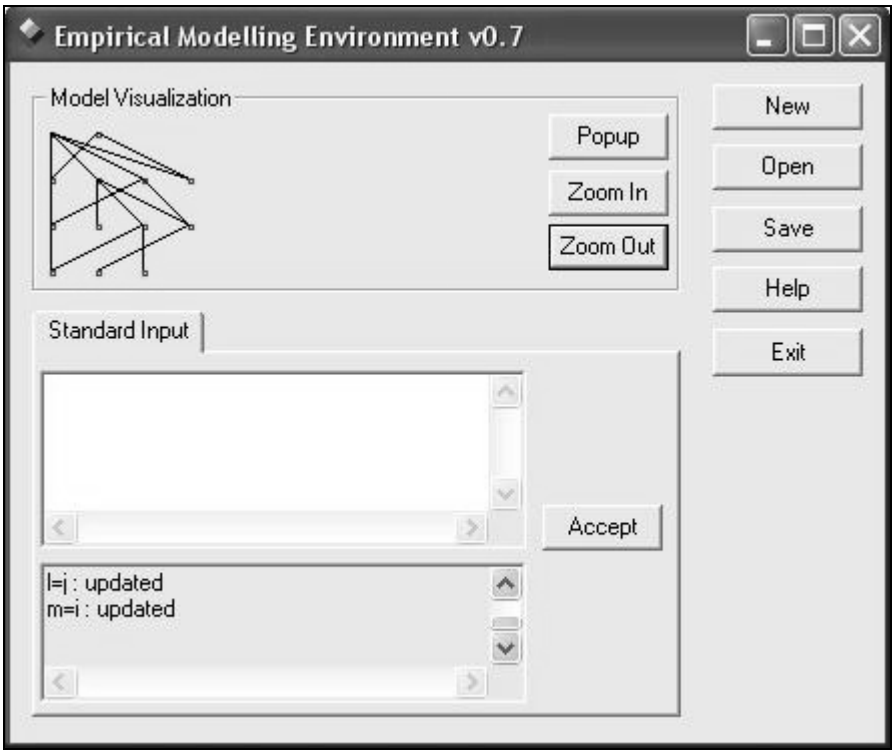
The core part of EME is an interpreter. The interpreter contains two parsers that are generated by using Parser Generator 1.11 from Bumble-bee Software Ltd. The Parser Generator generates C++ parser classes according to grammar specifications very similar to Lex and Yacc in Unix. For the visualisation of the model, EME makes use of the LEDA library. It is a library of the data types and algorithms of combinatorial computing. LEDA simplifies the process of implementing graph algorithms that are used for displaying the dependency graphs, checking cyclic dependencies and determine order of the evaluations.

1. Overall architecture

EME contains about 25 classes. They can be categorised into the following six groups:

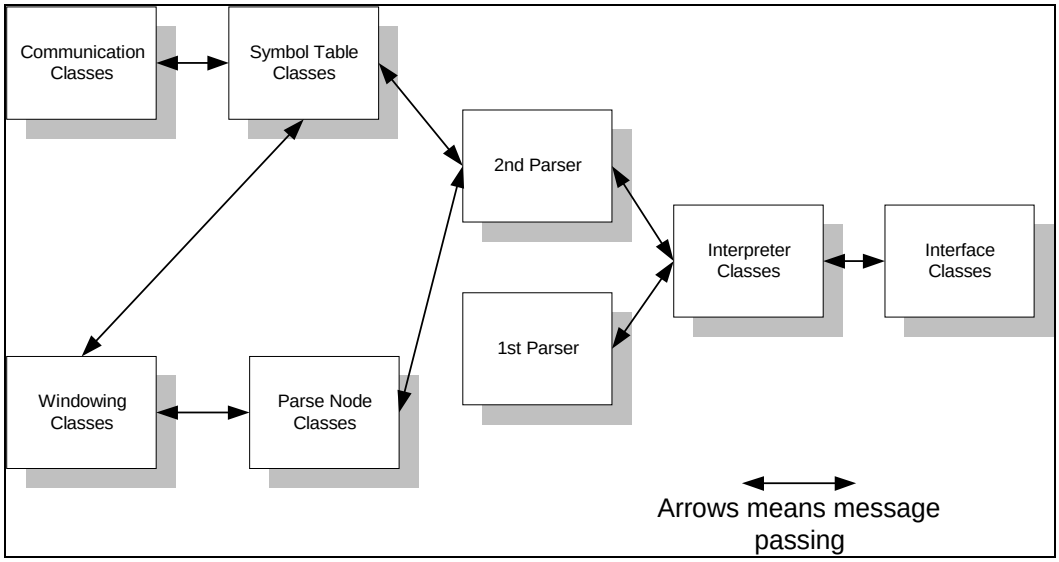
- The interface classes
- Interpreter classes
- Symbol table classes
- Parse node classes
- Windowing classes
- Communication classes

The **interface classes** implement a simple graphical user interface (as shown in the screen capture below) that a user can interact with the tool. The interface can be divided into two areas. The top area shows a model visualisation of data dependency graph with buttons of adjusting the size of the graph, and a popup button which is for opening a bigger window showing the graph in a more detail. The bottom area is for script input similar to the ‘input window’ of TkEden.



The **interpreter classes** implement one Lexer and two Yacc parsers. Two parsers use the same Lexer to get tokens from the input script but process the tokens in different ways that will be discussed in the following sections. The whole interpretation process of a script contains two passes, each pass uses a different parser.

The **symbol table classes** implement how definitions are stored in the memory. It also contains evaluation algorithms and graph layout algorithms. The **parse node classes** implement all the data types and their operations. The interpreter creates parse trees by using these nodes. **Windowing classes** are classes derived from MFC windowing classes that help to manage the side effects triggered by using windowing components. For example, a button should introduce a “button-clicked” definition to the symbol table when it is clicked. Finally, the **communication classes** are responsible for the peer-to-peer communication among EME application instances when we build a distributed model. The figure below shows the overall architecture of EME with arrows denoting interaction between classes.



2. The interpreter

The underlying interpreter contains one tokeniser and two parsers. All of them are generated by a C++ compiler compiler called Parser Generator. The whole interpreting process can be divided into two passes. Each parser is responsible for one pass of the process.

In the 1st pass, the syntax of inputting script is checked. All the branching positions in script are also determined and saved for later use in the 2nd pass. In the 2nd pass, the parser builds parse trees for definitions and calls symbol table to evaluate and store the definitions. It uses the branching information saved from the 1st pass to jump over a block of script if needed. For example, the logical expression of an if statement is checked. If the expression turns out to be false, we need a jump to the end of the if statement without executing anything enclosed in the if brackets. To understand this, we could imagine there is a high level program counter in the tokeniser that scans input token by token and each token is consumed and executed by the parser. The position of the program counter is at the beginning of next token. The parser tells the tokeniser when a jump over a block of scripts is needed. The position where the program counter will be jumped to is determined in the 1st pass of the interpretation process.

Here is the grammar used for both parsers:

```
lines     : lines line
| /* empty */
     ;
```

```
line : statement
     | error '\n'
     | '\n'
     ;

statement : definition
          | systemcommand
     | control
          | procedurespec
          ;
procedurespec  : ACTION var '{' lines '}'
          | FUNCTION var '{' lines '}'
              ;

definition     : var '=' rhs ';'
          ;

systemcommand  : '?' var ';'
          | UNDEFINE var ';'
     | LISTEN '(' NUMBER ')' ';'
          | CONNECT '(' expr ',' expr ',' expr ',' NUMBER ')' ';'
     | TESTSEND '(' expr ',' expr ')' ';'
          ;

control        : REPEAT '(' logicalexpr ')' '{' lines '}'
          | IF '(' logicalexpr ')' '{' lines '}'
          | IF '(' logicalexpr ')' '{' lines '}' else '{' lines '}'
          ;

var       : id
          | var '@' id
;

id        : ID
          | '<' expr '>'
     | '[' expr ']'
     | id '\\' NUMBER
| id '\\' ID
     | id '<' expr '>'
     | id ID
     | id '[' expr ']'
          ;

rhs       : expr
          ;

expr      : expr '+' expr
          | expr '-' expr
     | expr '*' expr
          | expr '/' expr
     | '(' expr ')'
     | '{' expr '}'
     | logicalexpr
          | NUMBER
     | USERTEXT
     | var
          | SIN '(' expr ')'
     | var '(' parameterlist ')'
     | var '(' ')'
          ;

parameterlist  : expr
          | parameterlist ',' expr
          ;

logicalexpr    : expr EQUAL expr
          | expr NOTEQUAL expr
     | expr SMALLEREQUAL expr
          | expr GREATEREQUAL expr
     | expr SMALLER expr
     | expr GREATER expr
     | NOT expr
          ;
```

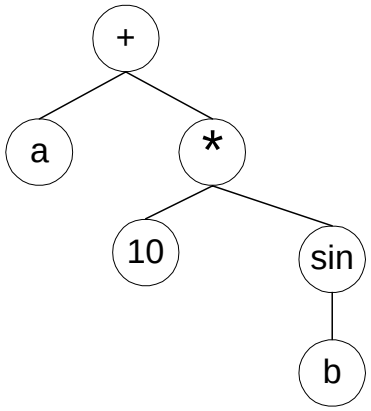
The grammar above is provisional. For example, in the current implementation, only Sin function among all the common trigonometric functions that can be implemented. The grammar here is for testing the functionalities of EME. There will be a big enhancement of the grammars in the release version of EME.

It is worth noting that, along the process of interpreting a definition, it is very important to know all the determinants that the definition is dependent on. The determinants are references to other definition variables. In a definitive system like EME, the set of determinants of a definition is dynamic. For example, in the definition x=a[i]; , the determinants of this definition is dependent on the current value of definition i. Therefore, in EME, the set of determinants of a definition will be determined not in interpreting time but every time when the definition is evaluated.

3. Building parse trees

The main purpose of the interpreter is to build parse trees that can be evaluated every time when the value of definitions need to be updated. A parse tree consists of links of parse nodes of many different types. In the current version of EME, there are about 25 types of parse nodes. They can be grouped into terminal nodes and non-terminal nodes. Typically, terminal nodes are nodes that contain primitive values of EME data types; non-terminal nodes contain operation instructions on the primitive nodes. Here gives an example of a parse tree of the definition:

x = a + 10 * sin(b);



In the above example, the terminal nodes are a,10 and b, whereas non-terminal nodes as +, * and sin function.

The advantage of building parse trees and storing them with the definition is that a script needs to be scanned once only. There is no need to interpret a definition every time when the value of it needs to be updated. The disadvantage is parse nodes are objects that are created dynamically and stored in memory. Therefore, memory usage in a large model will be an issue.

The evaluation of parse trees is implemented in the parse tree classes. The type of a parse node is represented as a unique integer number. Evaluation of a parse tree, typically consist of a sequence of recursive calls to evaluation operation implemented in parse node class. Along with the evaluation process, all the variable expressions are evaluated and variable references are obtained. The variables referenced in this process are recorded and stored as determinants of the definitions.

Implementation of most of the data types and their operations, like number and its arithmetic operations, is straightforward. However, when it comes to implementing data types with side effects, extra care should be made. Examples are the windowing types. A windowing type, such as a push button on the screen, creates two kinds of side effect – the side effect of displaying it on the screen and the side effect of event handling. To deal with these side effects, a wrapper class is created for each windowing type. The wrapper classes contain codes of putting and removing the windowing component on the screen and generating definitions when a user triggers a window event.

4. The symbol table

The **symbol table** contains a MFC implementation of hash table of **symbols**. Each of these symbols contains the following information:

- Name – variable name of the definition.
- A parse tree – parse tree of the definition created by the interpreter.
- Up-to-date flag – indicates whether the value of this symbol is up-to-date or not.
- Value – stores a parse node representing the value returned by evaluation of

its parse tree.

- String representation – a string representation of the definitions
- Dependents – array of variable names that are dependent on this definition.
- Determinants – array of variable names that this definition refers to.

Therefore, each symbol encapsulates detailed information about a definition, which is needed in the whole process of definition evaluation. The symbol table also owns:

- A dependency graph object – a LEDA graph object that stores the data dependency graph for visualisation.
- A communication sever object – responsible for network communication among instances of EME application.
- An action queue – stores a queue of actions generated in the process of evaluation. Actions in this queue will be executed one by one after all the values of definitions are up-to-date.

To help understand how the symbol table works, we list out briefly steps involved in defining a definition:

1. A definition is entered as a string to the Standard input.
2. The 1st pass of the interpreter makes sure there is no syntax error.
3. The 2nd pass of the interpreter builds parse tree while scanning the definition string.
4. The interpreter passes the variable name and its parse tree to the symbol table.
5. The symbol table evaluates the parse tree to determine all the determinants of this definition.
6. The symbol table does a topological sort on the dependency graph with the newly created determinant information. This checks cyclic dependency and also obtains a topological order of variables according to their dependencies. If there is cyclic dependency, the process stops and an error message will be generated.
7. The symbol table registers this definition as dependents of the determinants that are obtained in step 5.
8. The symbol table evaluates, in topological order, all the definitions that are registered at dependents of the definition.
9. During the evaluation process, actions are added to the action queue.

10. After all values of definitions are updated, actions in the action queue are executed one by one.

The symbol table also updates visualisation of the data dependency graph during the above process.

5. Algorithms applied

In this section, we will describe three algorithms applied in different stages of execution in EME. They are topological sort algorithm, one-way constrain satisfaction algorithm and graph layout algorithm.

a) Topological sort algorithm

We have used a modified version of topological sort algorithm from [Näh99] This algorithm is for checking cyclic dependency and determining evaluation order. The algorithm is listed in form of LEDA C++ code:

```
bool CSymboltable::TopologicalSort(GRAPH<string,int>* G, CMap<CString,LPCSTR,int,int>
&toporder)
{

     /* initialization:
     determine the indegree of all nodes and initialize a queue
     of nodes of indegree zero*/
     node_array<int> INDEG(*G);
     queue<node> ZEROINDEG;
     node v;
     forall_nodes(v,*G)
     if ( (INDEG[v] = (*G).indeg(v)) == 0 ) ZEROINDEG.append(v);

     /* removing nodes of indegree zero:
     consider the nodes of indegree zero in turn. When
     a vertex v is considered we number it and we decrease
     the indegrees of all adjacent nodes by one. Nodes whose
     indegree becomes zero are added to the rear of ZEROINDEG*/
     node w;
     edge e;
     int count = 0;

     while (!ZEROINDEG.empty())
     {
          v = ZEROINDEG.pop();
          toporder[(*G)[v]]=++count;
          forall_out_edges(e,v)
               {    w = (*G).target(e);
                    if ( --INDEG[w] == 0 ) ZEROINDEG.append(w);
               }
     }
     return (count == (*G).number_of_nodes());
}
```

This operation returns false if a cyclic dependency is detected. If it returns true, the toporder map variable containing a mapping from variable name to an integer

stores the topological order of the variable.

b) Evaluation algorithm

During the evaluation process, we have applied a modified version of one-way constrain satisfaction algorithm shown below. The algorithm updates values of the definitions in topological order. More detail explanation of this algorithm can be found in [Zan01].

```
BOOL CSymboltable::UpdateAllinTopologicalOrder(CSymbol* start)
{
     CMap<CString,LPCSTR,int,int> toporder;

     p_queue<int,CSymbol*> Q; // a priority queue
     int order,n,i;
     CSymbol *s, *d;
     pq_item item;

     Q.insert(1,start);

     while(!Q.empty()){
          item=Q.find_min();
          s=Q.inf(item);

          if(!Update(s,toporder)) return FALSE;
          n=s->m_dependents.GetSize();
          for(i=0;i<n;i++){
               d=Lookup(s->m_dependents[i]);
               ASSERT(d!=NULL);
               if(d->m_uptodate==true)d->m_uptodate=false;
               toporder.Lookup(d->m_name,order);
               Q.insert(order,d);

          }

          Q.del_item(item);

     }

     return TRUE;
}
```

c) drawing of dependency graph

The difficulty of drawing a dependency graphs is how to reduce crossings of edges and arrange the position of the nodes so that the graph drawn is more comprehensible. There are many graph layout research on drawing acyclic graphs. One of the famous algorithms is called Sugiyama Algorithm [Sug81]. We list out the step on performing the algorithm as below.

- Step 1: Convert the graph in into a "proper" hierarchy. If the hierarchy has long span edges, it is converted into a proper hierarchy by adding dummy nodes and edges.
- Step 2: The number of crossings of edges in the proper hierarchy is reduced by permuting orders of nodes in each level.
- Step 3: Horizontal positions of nodes are determined according to a given set of rules.
- Step 4: The graph is drawn with the dummy nodes removed.

The above steps are very brief description of what the algorithm does. For details of this algorithm, please refer to Sugiyama's paper [Sug81]. The progress of implementing the algorithm, at the time of writing this report, is still on the first step. At the moment, the data dependency graph drawn by EME is without edge crossing minimisations. Therefore, the data dependency graph drawn is only comprehensible for defining a small set of definitions.

6. Network communication

The implementation of network communication contains two classes. One implements a server and the other implements a client. They are all derived from CSocket in MFC. CSocket provides basic primitives for synchronise communication in TCP/IP network. EME is designed to use peer-to-peer communication model where a centralised server is not necessary. Each EME application instance is both a server and a client.

The sequences of making a connection between two EME application instances are shown below. Suppose we have two instances of EME running on two different machines connected in a computer network. We call them A and B. Just like telephone a call, one of these instances should be the caller and the other should be the receiver. However, after the connection is made, there is no difference between the caller and receiver - both can communicate freely through the connection, and if one of them closes the connection, the whole communication session will be closed.

If A calls B to make a connection, these steps of setting up the connection will start:

1. Initialise B's server to listen a port – it can be done by the command

listen(4000);. This initialises B to listen the port number 4000.

2. A client of A calls the server at B by the connect command – for example connect(“apple”,”banana”,”gem.dcs.warwick.ac.uk”,4000); where the first two arguments are logical names of A and B for this connection, i.e. A recognises this connection is made to “banana” and B recognises the connection is made to “apple”.
3. The server at B receives the connection request from a client from A.
4. The server at B creates a new client object that serves the client from A.
5. When the connection is made, both A and B are using a client to communicate with each other.

After the connection is made, A and B can send messages to each other. The types of these messages are ranging from simple string messages to symbol objects that store information on definitions. For example, when the determinants of a definition are obtained during the evaluation of the definitions, and one of the determinants is a remote variable that contains a @ character, the system will send an add-dependent message to the remote system. This add-dependent message registers the variable as a dependent of the remote variable so that when the remote variable’s changed, this variable will be informed. This mechanism forms part of procedures in dependency maintenance of a distributed model.

8. Saving a model

In EME, we save a model into a persistent storage as states of objects from the memory. Instead of saving scripts, the system saves the whole symbol table that contains all the states and dependencies of the model. This includes saving of the names, up-to-date statuses, parse trees, string representation of the definitions in the model. Therefore, loading the model does not involve reinterpreting everything again. EME uses the Serialisation facilities in MFC library to archive this.

9. References

[Näh99] S. Näher, K. Mehlhorn, *LEDA A Platform for Combinatorial and Geometric Computing*, Cambridge University Press, Germany, 1999.

[Sug81] K. Sugiyama, S. Tagawa, M. Toda, Methods for Visual Understanding of Hierarchical Systems Structures, In *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. Smc-11, No. 2, February 1981.

[Zan01] B. T. V. Zanden, R. Halterman, B. A. Myers, R. McDaniel, R. Miller, P. Szekely, D. A. Giuse, D. Kosbie, Lessons learned about one-way, dataflow constraints in the Garnet and Amulet graphical toolkits, In *ACM Transactions on Programming Languages and Systems (TOPLAS)*, v23n6, November 2001.

Appendix H: DMT interface menu reference

Model (all menu options relate to DMT models)

- New - Closes the existing model and creates a new one.
- Load - Loads a model file.
- Save As - Saves a model file. Suggested extension of the file is ".dmt".
- Statistics - Displays some statistics of the current model.
- Print - Outputs the model to a printer. Currently, it can only print one page. You are encouraged to use zoom functions to reduce the size of a large model before printing.
- Exit - Exits the program

Script (all options refer to Eden scripts)

- Input window - Allows user to type in Eden scripts.
- Direct import - Imports an Eden script directly into the current model.
- Export selection - Exports the selected definitions to a text window.

Layout

- Hierarchical-up - Layouts the model in natural evaluation order. Leaves are at the bottom of the layout.
- Hierarchical-down - Layouts the model in natural evaluation order. Leaves are at the top of the layout.
- Random - Randomly places the definitions.

Zoom

- Enlarge 10% - Enlarges 10% of the current graph size.
- Reduce 10% - Reduces 10% of the current graph size.
- Normal 100% - Sets back the graph size to original size.
- Fit to window - Zoom the current graph to fit the window.

Help

- Help - Displays this page.
- About - Displays version information and support details.

Appendix I: Definitions for lines in an OXO model

```
allsquares is
[s1,s2,s3,s4,s5,s6,s7,s8,s9,s10,s11,s12,s13,s14,s15,s16,s17,s18,s19
,s20,s21,s22,s23,s24,s25,s26,s27,s28,s29,s30,s31,s32,s33,s34,s35,s3
6,s37,s38,s39,s40,s41,s42,s43,s44,s45,s46,s47,s48,s49,s50,s51,s52,s
53,s54,s55,s56,s57,s58,s59,s60,s61,s62,s63,s64];
nofsquares is allsquares#;
lin1 is [s1,s2,s3,s4];
lin2 is [s1,s5,s9,s13];
lin3 is [s1,s6,s11,s16];
lin4 is [s2,s6,s10,s14];
lin5 is [s3,s7,s11,s15];
lin6 is [s4,s7,s10,s13];
lin7 is [s4,s8,s12,s16];
lin8 is [s5,s6,s7,s8];
lin9 is [s9,s10,s11,s12];
lin10 is [s13,s14,s15,s16];
lin11 is [s1,s17,s33,s49];
lin12 is [s1,s18,s35,s52];
lin13 is [s2,s18,s34,s50];
lin14 is [s3,s19,s35,s51];
lin15 is [s4,s19,s34,s49];
lin16 is [s4,s20,s36,s52];
lin17 is [s1,s21,s41,s61];
lin18 is [s1,s22,s43,s64];
lin19 is [s2,s22,s42,s62];
lin20 is [s3,s23,s43,s63];
lin21 is [s4,s23,s42,s61];
lin22 is [s4,s24,s44,s64];
lin23 is [s5,s21,s37,s53];
lin24 is [s5,s22,s39,s56];
lin25 is [s6,s22,s38,s54];
lin26 is [s7,s23,s39,s55];
lin27 is [s8,s23,s38,s53];
lin28 is [s8,s24,s40,s56];
lin29 is [s9,s25,s41,s57];
lin30 is [s9,s26,s43,s60];
lin31 is [s10,s26,s42,s58];
lin32 is [s11,s27,s43,s59];
lin33 is [s12,s27,s42,s57];
lin34 is [s12,s28,s44,s60];
lin35 is [s13,s25,s37,s49];
lin36 is [s13,s26,s39,s52];
lin37 is [s14,s26,s38,s50];
lin38 is [s15,s27,s39,s51];
lin39 is [s16,s27,s38,s49];
lin40 is [s16,s28,s40,s52];
lin41 is [s13,s29,s45,s61];
lin42 is [s13,s30,s47,s64];
lin43 is [s14,s30,s46,s62];
lin44 is [s15,s31,s47,s63];
lin45 is [s16,s31,s46,s61];
lin46 is [s16,s32,s48,s64];
lin47 is [s17,s18,s19,s20];
lin48 is [s17,s21,s25,s29];
lin49 is [s17,s22,s27,s32];
lin50 is [s18,s22,s26,s30];
lin51 is [s19,s23,s27,s31];
lin52 is [s20,s23,s26,s29];
lin53 is [s20,s24,s28,s32];
lin54 is [s21,s22,s23,s24];
lin55 is [s25,s26,s27,s28];
lin56 is [s29,s30,s31,s32];
lin57 is [s33,s34,s35,s36];
lin58 is [s33,s37,s41,s45];
```

```
lin59 is [s33,s38,s43,s48];
lin60 is [s34,s38,s42,s46];
lin61 is [s35,s39,s43,s47];
lin62 is [s36,s39,s42,s45];
lin63 is [s36,s40,s44,s48];
lin64 is [s37,s38,s39,s40];
lin65 is [s41,s42,s43,s44];
lin66 is [s45,s46,s47,s48];
lin67 is [s49,s50,s51,s52];
lin68 is [s49,s53,s57,s61];
lin69 is [s49,s54,s59,s64];
lin70 is [s50,s54,s58,s62];
lin71 is [s51,s55,s59,s63];
lin72 is [s52,s55,s58,s61];
lin73 is [s52,s56,s60,s64];
lin74 is [s53,s54,s55,s56];
lin75 is [s57,s58,s59,s60];
lin76 is [s61,s62,s63,s64];
alllines is
[lin1,lin2,lin3,lin4,lin5,lin6,lin7,lin8,lin9,lin10,lin11,lin12,lin
13,lin14,lin15,lin16,lin17,lin18,lin19,lin20,lin21,lin22,lin23,lin2
4,lin25,lin26,lin27,lin28,lin29,lin30,lin31,lin32,lin33,lin34,lin35
,lin36,lin37,lin38,lin39,lin40,lin41,lin42,lin43,lin44,lin45,lin46,
lin47,lin48,lin49,lin50,lin51,lin52,lin53,lin54,lin55,lin56,lin57,l
in58,lin59,lin60,lin61,lin62,lin63,lin64,lin65,lin66,lin67,lin68,li
n69,lin70,lin71,lin72,lin73,lin74,lin75,lin76];
noflines is alllines#;
linesthru1 is [lin1,lin2,lin3,lin11,lin12,lin17,lin18];
linesthru2 is [lin1,lin4,lin13,lin19];
linesthru3 is [lin1,lin5,lin14,lin20];
linesthru4 is [lin1,lin6,lin7,lin15,lin16,lin21,lin22];
linesthru5 is [lin2,lin8,lin23,lin24];
linesthru6 is [lin3,lin4,lin8,lin25];
linesthru7 is [lin5,lin6,lin8,lin26];
linesthru8 is [lin7,lin8,lin27,lin28];
linesthru9 is [lin2,lin9,lin29,lin30];
linesthru10 is [lin4,lin6,lin9,lin31];
linesthru11 is [lin3,lin5,lin9,lin32];
linesthru12 is [lin7,lin9,lin33,lin34];
linesthru13 is [lin2,lin6,lin10,lin35,lin36,lin41,lin42];
linesthru14 is [lin4,lin10,lin37,lin43];
linesthru15 is [lin5,lin10,lin38,lin44];
linesthru16 is [lin3,lin7,lin10,lin39,lin40,lin45,lin46];
linesthru17 is [lin11,lin47,lin48,lin49];
linesthru18 is [lin12,lin13,lin47,lin50];
linesthru19 is [lin14,lin15,lin47,lin51];
linesthru20 is [lin16,lin47,lin52,lin53];
linesthru21 is [lin17,lin23,lin48,lin54];
linesthru22 is [lin18,lin19,lin24,lin25,lin49,lin50,lin54];
linesthru23 is [lin20,lin21,lin26,lin27,lin51,lin52,lin54];
linesthru24 is [lin22,lin28,lin53,lin54];
linesthru25 is [lin29,lin35,lin48,lin55];
linesthru26 is [lin30,lin31,lin36,lin37,lin50,lin52,lin55];
linesthru27 is [lin32,lin33,lin38,lin39,lin49,lin51,lin55];
linesthru28 is [lin34,lin40,lin53,lin55];
linesthru29 is [lin41,lin48,lin52,lin56];
linesthru30 is [lin42,lin43,lin50,lin56];
linesthru31 is [lin44,lin45,lin51,lin56];
linesthru32 is [lin46,lin49,lin53,lin56];
linesthru33 is [lin11,lin57,lin58,lin59];
linesthru34 is [lin13,lin15,lin57,lin60];
linesthru35 is [lin12,lin14,lin57,lin61];
linesthru36 is [lin16,lin57,lin62,lin63];
linesthru37 is [lin23,lin35,lin58,lin64];
linesthru38 is [lin25,lin27,lin37,lin39,lin59,lin60,lin64];
linesthru39 is [lin24,lin26,lin36,lin38,lin61,lin62,lin64];
linesthru40 is [lin28,lin40,lin63,lin64];
linesthru41 is [lin17,lin29,lin58,lin65];
linesthru42 is [lin19,lin21,lin31,lin33,lin60,lin62,lin65];
linesthru43 is [lin18,lin20,lin30,lin32,lin59,lin61,lin65];
linesthru44 is [lin22,lin34,lin63,lin65];
linesthru45 is [lin41,lin58,lin62,lin66];
linesthru46 is [lin43,lin45,lin60,lin66];
linesthru47 is [lin42,lin44,lin61,lin66];
linesthru48 is [lin46,lin59,lin63,lin66];
linesthru49 is [lin11,lin15,lin35,lin39,lin67,lin68,lin69];
linesthru50 is [lin13,lin37,lin67,lin70];
```

```
linesthru51 is [lin14,lin38,lin67,lin71];
linesthru52 is [lin12,lin16,lin36,lin40,lin67,lin72,lin73];
linesthru53 is [lin23,lin27,lin68,lin74];
linesthru54 is [lin25,lin69,lin70,lin74];
linesthru55 is [lin26,lin71,lin72,lin74];
linesthru56 is [lin24,lin28,lin73,lin74];
linesthru57 is [lin29,lin33,lin68,lin75];
linesthru58 is [lin31,lin70,lin72,lin75];
linesthru59 is [lin32,lin69,lin71,lin75];
linesthru60 is [lin30,lin34,lin73,lin75];
linesthru61 is [lin17,lin21,lin41,lin45,lin68,lin72,lin76];
linesthru62 is [lin19,lin43,lin70,lin76];
linesthru63 is [lin20,lin44,lin71,lin76];
linesthru64 is [lin18,lin22,lin42,lin46,lin69,lin73,lin76];
linesthru is
[linesthru1,linesthru2,linesthru3,linesthru4,linesthru5,linesthru6,
linesthru7,linesthru8,linesthru9,linesthru10,linesthru11,linesthru1
2,linesthru13,linesthru14,linesthru15,linesthru16,linesthru17,lines
thru18,linesthru19,linesthru20,linesthru21,linesthru22,linesthru23,
linesthru24,linesthru25,linesthru26,linesthru27,linesthru28,linesth
ru29,linesthru30,linesthru31,linesthru32,linesthru33,linesthru34,li
nesthru35,linesthru36,linesthru37,linesthru38,linesthru39,linesthru
40,linesthru41,linesthru42,linesthru43,linesthru44,linesthru45,line
sthru46,linesthru47,linesthru48,linesthru49,linesthru50,linesthru51
,linesthru52,linesthru53,linesthru54,linesthru55,linesthru56,linest
hru57,linesthru58,linesthru59,linesthru60,linesthru61,linesthru62,l
inesthru63,linesthru64];
```

Appendix J: An LSD account for a train arrival and departure protocols

This is an LSD account for a train arrival and departure protocols developed by Y. P. Yung in the EM research group. The involved agents are a stationmaster, a guard, a driver, a train, passengers and doors.

```
agent sm() {

oracle    (time) Limit, Time,      // knowledge of time to elapse before departure due
     (bool) guard_raised_flag,     // knowledge of whether the guard has raised his flag
     (bool) driver_ready,          // knowledge of whether the driver is ready
     (bool) around[d],             // knowledge of whether there's anybody around doorway
     (bool) door_open[d];          // the open/close status of door d (for d = 1 ..
number_of_doors)

state     (time) tarrive = |time|, // the S-M registers time of arrival
     (bool) can_move,              // the signal observed by driver for starting engine
     (bool) whistle = false,       // the whistle is not blowing
     (bool) whistled = false,      // the whistle has not blown
     (bool) sm_flag = false,       // S-M lowers flag
     (bool) sm_raised_flag = false;// S-M has not raised flag

handle    (bool) can_move,
     (bool) whistle,
     (bool) whistled,
     (bool) sm_flag,
     (bool) sm_raised_flag;
     (bool) door_open[d];          // the open/close status of door d (for d = 1 ..
number_of_doors)

derivate
               number_of_doors
     (bool) ready =     /\    (!door_open[d]);    // are all doors shut?
                 d = 1
     (bool) timeout = (Time - tarrive) > Limit;   // departure due

protocol
     door_open[d] ^ !around[d] -> door_open[d] = false; (d = 1 .. number_of_doors)
     ready ^ timeout ^ !whistled -> whistle = true; whistled = true; guard(); whistle =
false;
     ready ^ whistled ^ !sm_raised_flag -> sm_flag = true; sm_raised_flag = true;
     sm_flag ^ guard_raised_flag -> sm_flag = false;
     ready ^ guard_raised_flag ^ driver_ready ^ engaged ^ !can_move -> can_move = true;

}



agent guard() {

oracle    (bool) whistled, sm_raised_flag, brake;

state     (bool) guard_raised_flag = false,
     (bool) guard_flag = false,
     (bool) brake;

handle    (bool) guard_raised_flag, guard_flag;

derivate  LIVE = engaging || whistled;

protocol
     engaging -> brake = true; running = false;
```

```
     sm_raised_flag ^ brake -> brake = false; guard_flag = true; guard_raised_flag = true;
     guard_flag ^ !sm_flag -> guard_flag = false;

}




agent driver() {

oracle    (bool) can_move, engaged, whistled;
     (position) at, from;

handle    (position) to, from,
     (bool) running,
     (bool) driver_ready = false;

state     (bool) driver_ready = false,
     (position) from;

protocol
     whistled ^ !driver_ready -> driver_ready = true;
     engaged ^ from <> at -> from = at; to = next_station_after(at);
     can_move ^ engaged -> driver_ready = false; running = true;

}




agent train() {

state     (bool) running = true,
     (bool) brake = false,
     (bool) door_open[d] = false, (d = 1 .. number_of_doors)
     (position) from = station1,
     (position) to = station2,
     (position) at = some_position,
     (bool) engaging, engaged, leaving, alert;

handle    (bool) alert;

derivate
     (bool) engaging = running ^ to == at,
     (bool) leaving = running ^ from == at,
     (bool) engaged = !running;

protocol
     engaging ^ !alert -> alert = true; guard(); sm();
     leaving ^ alert -> alert = false; delete guard(), sm();

}




agent passenger((int) p, (int) d, (position) _from, (position) _to) {
// passenger p intending to travel from station _from to station _to
// and he will access through door d of the train
oracle    (position) at,
     (bool) door_open[d];

state     (bool) pos[p] = OUT_DOOR, alighting[p], boarding[p], join_queue[p,d];

handle    (position) from[p] = _from;
     (position) to[p] = _to;
     (int) door[p] = d;
     (bool) pos[p],
     (bool) door_open[d];

derivate
     alighting[p] = at == to[p] ^ pos[p] != OUT_DOOR && engaged;
     boarding[p] = at == from[p] ^ pos[p] != IN_DOOR && engaged;
     join_queue[p,d] = (alighting[p] ^ door_open[d] ^ pos[p] == IN_DOOR) ||
          (boarding[p] ^ door_open[d] ^ pos[p] == OUT_DOOR);
     LIVE = !(at == to[p] ^ pos[p] == ON_PLATFORM);
```

```
protocol
     at == to[p] ^ pos[p] == AT_SEAT -> pos[p] = IN_DOOR;
     alighting[p] ^ !door_open[d] -> door_open[d] = true;
     alighting[p] ^ pos[p] == AT_DOOR ^ door_open[d] ^ !queuing[d]
          -> pos[p] == OUT_DOOR; door_open[d] = false; pos[p] = ON_PLATFORM;
     alighting[p] ^ pos[p] == AT_DOOR ^ door_open[d] ^ queuing[d]
          -> pos[p] == OUT_DOOR; pos[p] = ON_PLATFORM;
     boarding[p] ^ !door_open[d] -> door_open[d] = true;
     boarding[p] ^ pos[p] == AT_DOOR ^ door_open[d] ^ !queuing[d]
          -> pos[p] = IN_DOOR; door_open[d] = false; pos[p] = AT_SEAT;
     boarding[p] ^ pos[p] == AT_DOOR ^ door_open[d] ^ queuing[d]
          -> pos[p] = IN_DOOR; pos[p] = AT_SEAT;

}



agent door((int) d) {

oracle    (int) pos[p], door[p]; (p = 1 .. number_of_passengers)

state     (bool) queuing[d], occupied[d], around[d];

derivate
     queuing[d] = there exists p such that join_queue[p,d] == true;
     occupied[d] = there exists p such that (pos[p] == AT_DOOR ^ door[p] == d)
     around[d] = there exists p such that (door[p] == d ^
          (pos[p] == IN_DOOR || pos[p] == AT_DOOR || pos[p] == OUT_DOOR))

protocol  queuing[d] ^ !occupied[d] ^ join_queue[p,d] == true
          -> pos[p] = AT_DOOR; (p = 1 .. number_of_passengers)
}
```