Notes on the use of the EDEN interpreter prepared by Yun Pui (Simon) Yung (1996)

EDEN

Basic features

- Disciplines in using EDEN
  illustrated by the DoNaLD translator
- More advanced features
- Evaluation strategy

Interfacing Scout, DoNaLD and EDEN

- Translation hierarchy
- Selecting entry point
- Relating Scout, DoNaLD and EDEN

Tk interface

- Features

Other practical hints

Attributes in DoNaLD

- Initialisation of EDEN strings and lists
- Writing a clocking mechanism in EDEN

1

Data Type

int	char	float
string	pointer	list
	@	

User-defined Function

```
func fac {
    return $1 > 1 ? fac($1 - 1) * $1 :1;
}

func fac { para N;
    return N > 1 ? fac(N - 1) * N : 1;
}
```

Variable

formula variable e.g. vat is price * 0.175
read/write var e.g. foreign = local * rate

User-defined procedure and action

```
proc monitor_v : v {
    writeln(v);
}
```

2

MSc Tutorial on Tkeden

Pre-defined EDEN Functions

I/O Functions

write(), writeln()

pe Conversion Functions

int(), char(), str(), float()
type()

String Functions

substr(), //, strcat(), nameof()

List Functions

sublist(), //, listcat(), array()

Time Functions

time(), ftime(), gettime()

Mathematical Functions

sin(), cos(), tan() ...

3

MSc Tutorial on Tkeden

Pre-defined EDEN Functions (cont.)

Unix Functions

getenv(), putenv(),
error(), error_no(),
backgnd(), pipe()
get_msgq(), remove_msgq(),
send_msg(), receive_msg(),
fopen(), fclose(), fprintf(), gets() ...

Script Functions

include(),
apply(),
execute(), todo(),
exit(),
eager(), forget(), touch(),
formula_list(), action_list(),
symboltable(), symbols(), symboldetail()

4

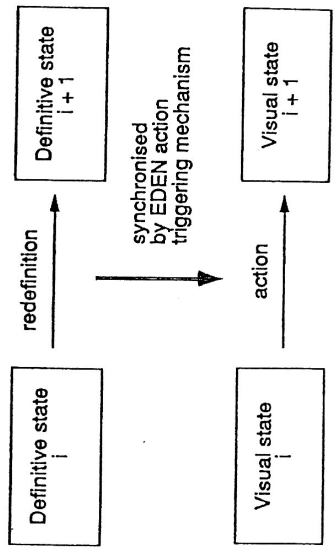
The existence of procedural elements in EDEN (assignments and procedures over RWV) means that we can, but by no means recommended to, use EDEN as a conventional programming language.

We need to know when to use the procedural elements and when to use the definitive elements.

- Exploit definitions as much as possible in representing abstract states
- Use DoNaLD and Scout, if possible, for matching between the visual state and the abstract state
- Try to avoid side-effects as much as possible in visualisation, e.g. avoid the use of loci in DoNaLD.
- Action may be used to synchronise between abstract and visual models, or to be used in simulating agent actions.
- Procedures are used to transit from one state to another.

5

Visualisation of Definitive State



Implementing Definitive Notations

1. Set Naming Scheme

DoNaLD Name	EDEN Name
table	_table
table/drawer	_table_drawer
table/drawer/width	_table_drawer_width

2. Implement Data Types & Operators

DoNaLD Type	EDEN Type
integer	integer
point	['C', integer, integer]
line	['L', point, point]

DoNaLD Operator	EDEN Operator/ Function
div	/
+ (vector sum)	see below

```
func vector_add {
   para p1, p2;
   return [
      'C',
      p1[1] + p2[1],
      p1[2] + p2[2]
   ];
}
```

3. Implement Implicit Actions

DoNaLD Code	EDEN Action Specification
integer i	N/A
point p	`proc P_p: _p { plot_point(&_p); }`
line L	`proc P_L: _L { plot_line(&_L); }`

More Advanced Features

Extracting state information

? v ;
symboltable()
symbols(type)
symboldetail(&var)

Generating / executing EDEN statements online

'var_name'
nameof(&var)
execute(string)
todo(string)

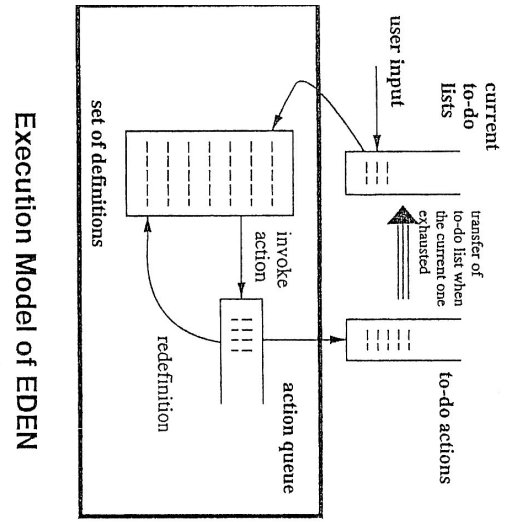
Controlling the execution

autocalc
eager()

6

```
proc dummy : A {  /* dummy() will be executed when A changes */
      auto i;     /* local variable */
      for (i = 1; i <= 10; i++) {
            V = i;
      }
}

proc Print_V : V {
      write(V, ' ');
}

A = 1;       /* change A to trigger dummy() */

/* result is :
1
*/
```

```
proc dummy : A {  /* dummy() will be executed when A changes */
      auto i;     /* local variable */
      for (i = 1; i <= 10; i++) {
            V = i;
      }     eager();
}

proc Print_V : V {
      write(V, ' ');
}

A = 1;       /* change A to trigger dummy() */

/* result is :
1 2 3 4 5 6 7 8 9 10
*/
```

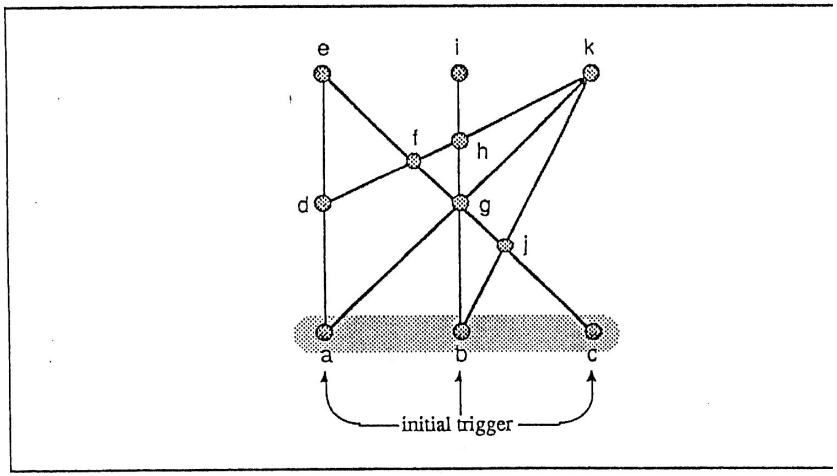
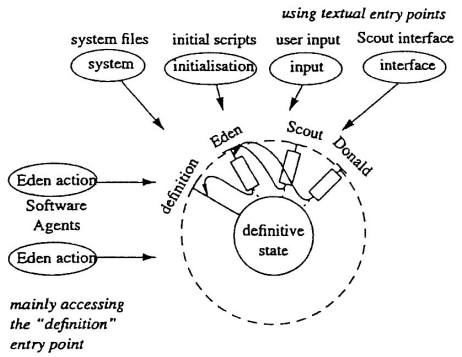


Figure 5-2: The dependency graph of a set of definitions



7

Scout, DoNaLD and EDEN

%scout

. *definitions in Scout*

%donald

. *DoNaLD definitions in the last DoNaLD context*
. *and viewport*

%eden

. *EDEN definitions and statements*

%error

. *unrecognised notation name will be treated*
. *as %eden*

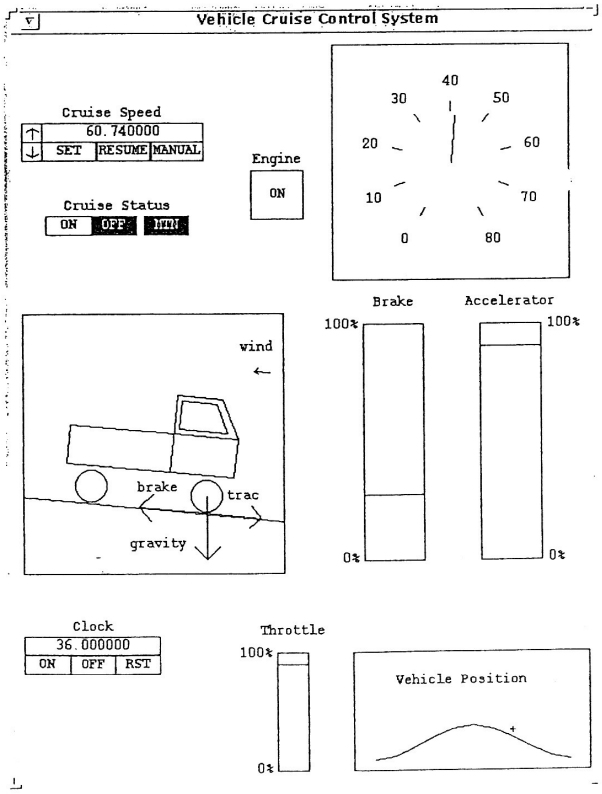
%donald

? *a line of EDEN, e.g.:*
?A_cable = "linestyle=dashed";

8

Extract From Vehicle Cruise Control Simulation

```
setBtn is (crSetBtn_mouse_1[2] == 4) ? pbDown : pbUp;
resBtn is (crResumeBtn_mouse_1[2] == 4) ? pbDown : pbUp;
manBtn is (crManualBtn_mouse_1[2] == 4) ? pbDown : pbUp;

%scout
window vehicle;
vehicle = {
        type: DONALD,
        box: [(10, 240), (230, 460)],
        pict: "VEHICLE",
        xmin: -700,  ymin: -300,
        xmax: 300,   ymax: 700,
        border: 1
};

%donald
viewport MODEL
openshape conceptCar
within conceptCar {
    circle frontWheel, backWheel
        frontWheel = circle((0,0), 60)
        backWheel = circle((-450, 0), 60)
```



• selecting DoNaLD viewport to view
  defining the mapping between the DoNaLD drawing space and Scout window

```
%scout
window vehicle = {
      type: DONALD
      box: [{10, 240}, {230, 460}]
      pict: "VEHICLE"
      xmin: -700,    ymin: -300
      xmax: 300,     ymax: 700
      border: 1
};

%donald
viewport MODEL              # not to be displayed

openshape conceptCar
within conceptCar {
      circle frontWheel, backwheel
      frontWheel = circle({0, 0}, 60)
      ...


viewport VEHICLE            # displayed in window vehicle

shape vehicle
vehicle = rot(conceptCar, {0, 0}, gradient)
```



```
%scout
window brakePedal = {
      type:       DONALD
      box:        [brakeOrg, brakeorg + {BAwidth, BAlength}]
      pict:       "BRAKE"
      xmax:       100
      ymax:       100
      border:     1
      sensitive:  ON
};

string startRelief;
window clkStartBtn = {
      string:     "ON"
      frame:      ([clkOrg + {0, 2.r + 6}, 1, 5])
      relief:     startRelief
      alignment:  CENTRE
      border:     1
      sensitive:  ON
};
%eden
startRelief is clkStartBtn_mouse_1[2] == 4 ? "sunken" :
      "raised";
```

```
brakePedal_mouse = [1, 4, 1, 37, 50];
clkStartBtn_mouse_1 = [1, 4, 0, 450, 600];
```


Features of Tk Interface

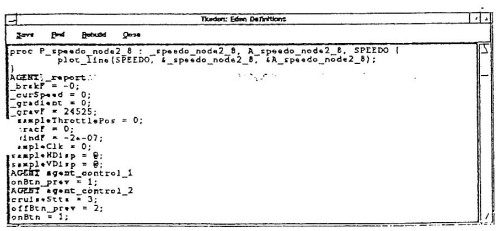
Supports different views of definitive store

Can examine definitions storing in the Scout and DoNaLD translators and the EDEN interpreter.

Since the translated Scout or DoNaLD definitions can be redefined in EDEN, there may be inconsistency between definitions as recorded in the translators and the EDEN interpreter.

In the EDEN definition view, a colour coding scheme is adopted to indicate the entry points of the definitions.

The sources of definitions (the agents that cause the defintions to be defined or modified) are also indicated.



Features of Tk Interface (cont.)

Recording history of interaction

Tkeden remembers:

- the scripts entered through the input window
- the definitions generated by the mouse or key actions in the Scout windows.
- error messages

A copy of the histroy is also saved automatically to the file named `.tkeden-history` in the user's home directory.

Allow save and load of current definitions

The main aim of the save and load facility is to save the current stage of script development so that it can be worked on in a separate session.

However, tkeden's 'save' facility will reorder the definitions, remove comments and has some problems in handling triggering of actions. Therefore, the usefulness of it is limited.

The history file may be another alternative.

Attribute	Applicable to	Value	Default
color	any shape	X colour name transparent	black
linewidth	line, arc, circle, ellipse, shape	integer	0 - min width
linestyle	line	dotted, dashed (poorly rendered) solid	solid
arrow	line	first, last, both, none	none
fill	any shape	solid, hollow	hollow
locus	any shape	true, false	false

13

Strings and Lists

• Although EDEN does not require declaration of variables, also it allow dynamic size changes of EDEN strings and lists, in some cases care should be taken to make sure the variables are in fact of certain types and contain large enough storage space for the intended operations.

• For example:

```
L[3] = 100;          /* L must have at least 3 elements */
scanf("%s", &s);     /* scanf() will not change the
                        type of s, s has to be initialised
                        as a beforehand */
```

• In many cases, assigning variables to [] or "" are already good enough.

• Array() and substring() functions are also very convenient.

```
L = array(4);        /* L = [@, @, @, @] */
L = array(3, 0);     /* L = [0, 0, 0] */
s = substr("", 1, 10); /* s contains 10 spaces */
```

14

A Clocking Mechanism

```
chime = 0;

proc clock_watcher : clock {
    if (clock - clock_init >= 5) {
        clock_init = clock;
        chime++;
    }
    todo("clock = "// str(time()) // ";");
    /* the redefinition is delayed to allow user interaction */
}

proc bell : chime {
    if (chime)
        writeln("Ding Dong!");
}

/* Initialise clock to the current time. This will trigger the
clocking mechanism as well */
clock = clock_init = time();
```

15

Note that recent versions of EDEN include the `edenclocks()` feature, so that the use of the `todo()` mechanism to support clocking is no longer necessary (cf. the two variants of the jugs model discussed in the module). The general principles governing the operation of the user input and action queue are still relevant however, as are the techniques used to interpret the `donald` and `scout` notations.