Introduction to EDEN

From a practical perspective …

Background

EDEN interpreter due to Y W (Edward) Yung (1987)

Designed for UNIX/C environment
EDEN = evaluator for definitive notations

"hybrid" tool = definitive + procedural paradigms
… essential to drive UNIX utilities and hw devices

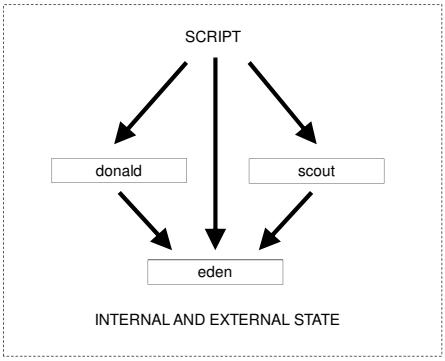
Extensions by Y P (Simon) Yung, Pi-Hwa Sun, Ashley Ward, Eric Chan and Ant Harfield

The use of the word "definitive"

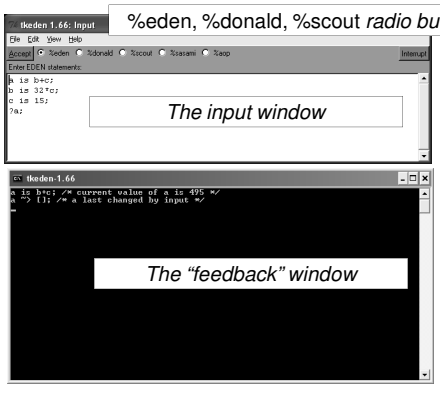
definitive = definition-based

a *definitive notation* = a notation within which definitions of variables can be made

a *definitive script* = a set of definitions expressed in one or more definitive notations



The basic architecture of the EDEN interpreter



Interpreting the eden notation

Basic EDEN interaction

- Use the File option to include scripts and to save the history of interaction
- Use the View option to inspect the current contents of the script and the command history
- Use the Help option to get quick reference information for eden, donald and scout
- Use the Accept button (or alt-A) to process script in the input window
- Use keyboard shortcuts to recall previous input

Basic characteristics of EDEN 1

The eden notation uses C-like

- syntactic conventions and data types
- basic programming constructs:

  `for`, `while`, `if` and `switch`

Types: `float`, `integer`, `string`, `list`.

Lists can be recursive and need not be homogeneous in type. Comments are prefaced by ## or enclosed in /* .... */.

Basic characteristics of EDEN 2

Two sorts of variables in eden:

formula and *value* variables.

Formula variables are definitive variables.

Value variables are procedural variables.

The type of an eden variable is determined dynamically and can be changed by assignment or redefinition.

Programming / modelling in EDEN

The three primary concepts in EDEN are:

- definition
- function
- action

Informally

definition ~ spreadsheet definition

function ~ operator on values

action ~ triggered procedure

Definitions in eden

A formula variable v can be defined via

v is f(a,b,c);

EDEN maintains the values of definitive variables automatically and records all the dependency information in a definitive script.

Yellow text indicates eden keywords

Functions in eden

Functions can be defined via

```
func F
/* function to compute result = F(a,b,...,c) */
{
        para a, b, ..., c        /* pars for the function */
        auto result, x, y, …, z      /* local variables */
        <sequence of assignments and constructs>
        return result
}
```

Actions in eden

Actions can be defined via

```
proc P : r, s, …, t
/* proc triggered by variables r, s, …, t */
{
        auto x, y, …, z     /* local variables */
        <sequence of assignments and definitions>
}
```

Action P is triggered whenever one of its triggering variables r, s, … , t is updated / touched

Basic concepts of EDEN 1

Definitions are used to develop a definitive script to describe the current state: change of state is by adding a definition or redefining.

Functions are introduced to extend the range of operators used in definitions.

Actions are introduced to automate patterns of redefinition where this is appropriate.

Basic concepts of EDEN 2

In model-building using EDEN, the key idea is to first build up definitive scripts to represent the current 'state-as-experienced'.

You then refine the script through observation and experiment, and rehearse meaningful patterns of redefinition you can perform.

Automating patterns of redefinition creates 'programs' within the modelling environment

Standard techniques in EDEN

Interrogating values and current definitions of variables in eden. To display:

- the current value of an eden variable v, invoke the procedure call

```
writeln(v)
```

- the defining formulae & dependency status of v, invoke the query

```
?v;
```

Typical EDEN model development

Edit a model in one window (e.g. using Textpad) and simultaneously execute EDEN in another

Cut-and-paste from editor window into interpreter window.

In development process, useful to be able to undo design actions: restore scripts of definitions by re-entering the original definitions.

To record the development history comment out old fragments of scripts in the edited file.

Managing EDEN files

Useful to build up a model in stages using different files.

Can include files using

```
include("filename.e");
```

or via the menu options in the input window.

Can consult / save entire history of interaction.

System also saves recent interaction histories.