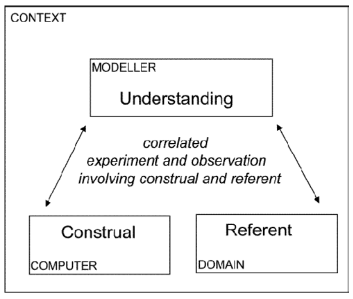
Empirical Modelling as *Construction*



Construal – in the sense of David Gooding

- Construing – "making sense of"/"explaining"
- Construals are a means of interpreting unfamiliar experience and communicating one's trial interpretations.
- Construals are practical, situational and often concrete. They belong to the pre-verbal context of ostensive practices.

The importance of construal …

In classical computer science, we aspire to understand comprehensively first, then build

Before we can prescribe programs, we first seek to prescribe the environment and the function of the program and roles of the users

"If I were you, I wouldn't start from here"

The importance of construal …

In practical computing , we often have to negotiate with systems that are far from ideal

In prescribing programs, we find that the function of the program and roles of the users emerge and the environment is evolving beyond our control

"We have to proceed from where we are"

The importance of construal …

Even in aspiring to high standards in specification and comprehension, we have to communicate about real-world confusion and how it might be resolved …

… Empirical Modelling is concerned with giving computer support to gaining such understanding

A case study in construal

In *"Constructivism in CSE"*, Ben-Ari speaks of the computer as "an accessible ontological reality" of which the student must develop a mental model.

In practice, even computing utilities – and even more systems! - are hard (?) for users to comprehend fully

Whether or not we can appreciate their full reality, need to expose mental models of them … for many reasons

My working understanding of the vi editor

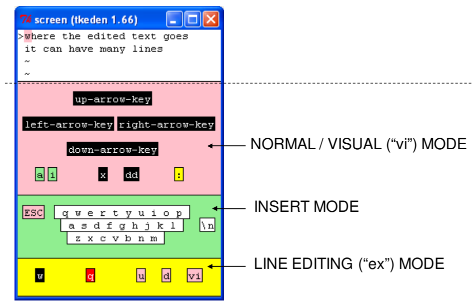
Key feature of vi: use of 'modes' (deprecated in HCI)

- When interacting with vi, we must know the current mode at any time: “need a good construal of modes”
- UNIX makes so much tacit and invisible – this suits the expert, but not the novice

Aim to gain / communicate understanding of vi modes by making a model ...

.. . such a model benefits both the builder and the observer, but mostly the builder

Construing modes in the `vi` editor



```
screen (tkeden 1.66)
>where the edited text goes
 it can have many lines
 ~
 ~
```

Basic observables …

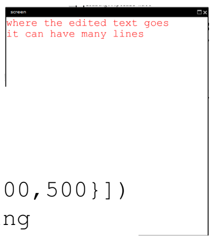
- **`textediting`** – the text that is being edited

```
%eden
textediting is "where the edited text
goes\nit can have many lines\n";
```

- **`docwin`** – the panel holding the text

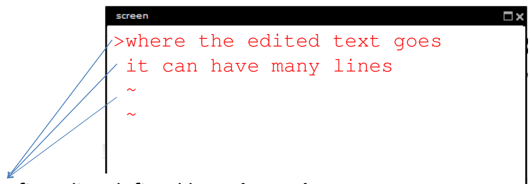
```
%scout
window docwin;
screen = <docwin>;
```

… basic observables



```
screen
where the edited text goes
it can have many lines
```

```
%scout
 docwin = {
     type: TEXT
     frame: ([{0,0},{500,500}])
     string: textediting
     bgcolor: "white“
     fgcolor: "red“
     border: 1
};
```

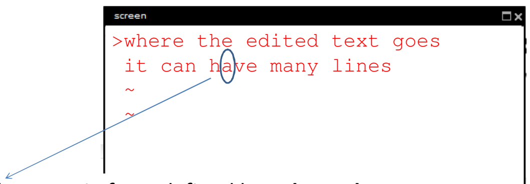


```
screen
>where the edited text goes
 it can have many lines
 ~
 ~
```

Prefix to line defined by a **dependency**

“>”	if it’s the current line
“ “	if it’s not current line and non-empty
“ ~”	if it’s an empty line

Basic dependencies …



```
screen
>where the edited text goes
 it can have many lines
 ~
 ~
```

Character in focus defined by a **dependency**

... it depends on

- the content of the text
- the current line number
- the column number in the current line

… basic dependencies

Formulating basic dependencies …

```
%eden
lines is [line1, line2, line3, line4];
line1 = "where the edited text goes";
line2 = "it can have many lines";
line3 = "";
line4 = "";

## textediting is line1 // "\n" // line2 // "\n" // line3
  // "\n" // line4 // "\n";
textediting is mark1 // line1 // "\n" // mark2 // line2 //
  "\n" // mark3 // line3 // "\n" // mark4 // line4 // "\n";

mark1 = mark2 = mark3 = mark4 = " ";

currline = 1;
```

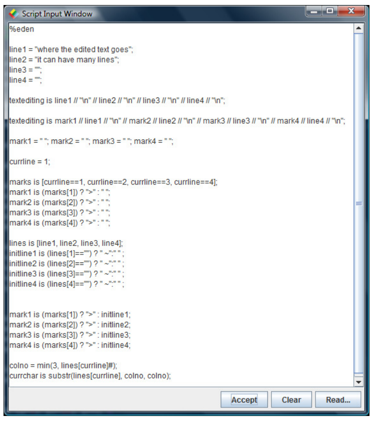
Formulating basic dependencies …

```
%eden

marks is [currline==1, currline==2, currline==3,
  currline==4];

mark1 is (marks[1]) ? ">" : " ";
mark2 is (marks[2]) ? ">" : " "; …

initline1 is (lines[1]=="") ? " ~":" " ;
initline2 is (lines[2]=="") ? " ~":" " ; …

mark1 is (marks[1]) ? ">" : initline1;
mark2 is (marks[2]) ? ">" : initline2; …

colno = min(3, lines[currline]#);
currchar is substr(lines[currline], colno, colno);
```

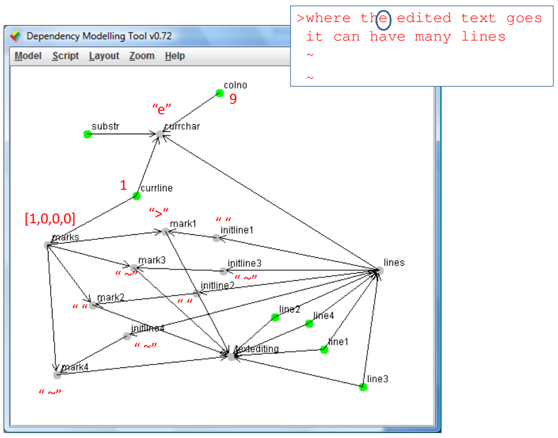
Visualising Dependencies

… using the DMT

"Dependency Modelling Tool"



```
%eden

line1 = "where the edited text goes";
line2 = "it can have many lines";
line3 = "";
line4 = "";

textediting is line1 // "\n" // line2 // "\n" // line3 // "\n" // line4 // "\n";

textediting is mark1 // line1 // "\n" // mark2 // line2 // "\n" // mark3 // line3 // "\n" // mark4 // line4 // "\n";

mark1 = " "; mark2 = " "; mark3 = " "; mark4 = " ";

currline = 1;

marks is [currline==1, currline==2, currline==3, currline==4];
mark1 is (marks[1]) ? ">" : " ";
mark2 is (marks[2]) ? ">" : " ";
mark3 is (marks[3]) ? ">" : " ";
mark4 is (marks[4]) ? ">" : " ";

lines is [line1, line2, line3, line4];
initline1 is (lines[1]=="") ? " ~":" " ;
initline2 is (lines[2]=="") ? " ~":" " ;
initline3 is (lines[3]=="") ? " ~":" " ;
initline4 is (lines[4]=="") ? " ~":" " ;


mark1 is (marks[1]) ? ">" : initline1;
mark2 is (marks[2]) ? ">" : initline2;
mark3 is (marks[3]) ? ">" : initline3;
mark4 is (marks[4]) ? ">" : initline4;

colno = min(3, lines[currline]#);
currchar is substr(lines[currline], colno, colno);
```



```
>where the edited text goes
 it can have many lines
 ~
 ~
```

Manual to automated

- Dependencies shape the environment for agent interaction
- What the modeller can do e.g. redefine `currline` and `colno` can be semi-automated
- Issues
  – making it *convenient* to make changes
  – *constraining* the scope of redefinition

  For this we have to provide an interface

Shaped meaning interactively

- Illustrating how `colno` can be redefined:

```
currline++;
for (i=1; i<=9; i++) {
  colno=i;
  write(currchar);
}
```

- Introducing `setcolno` and arrow buttons
- Affording, then restricting cf. suppressing automated interaction outside particular modes

Personal (and scruffy!) construal

- Reflects my limited understanding of **vi**
- It's not how **vi** works – e.g.
  - text in **vi** is not stored as a list of lines
  - the cursor control is not faithful to the editor
  - my names for the modes are not as documented

… but debugging the *users* is as important as debugging the *software*

Rethinking computing

- Fudging to cope with the empirical aspect in handling font size cf. "tuning an instrument"
- Ambitious proposals for a broader science of computing that embraces experiment fully
- Addressing the semantics of semantics

Acknowledgements

Allan Wong, who developed the DMT
Chris Roe, who investigated constructionism
Russell Boyatt, who put me right about vi
Ashley Ward, who helped debug the modellers