Timetabling from an Empirical Modelling perspective

Meurig Beynon
Soha Maad, Allan Wong, Ashley Ward

Department of Computer Science
University of Warwick
http://www.dcs.warwick.ac.uk/modelling

Figure 2: The case-study context

Content of talk

Introduction and Orientation

1. A simple timetabling case study
2. Timetabling application vs. timetabling instrument
3. Perspectives on Empirical Modelling (EM)

Principles of EM

4. Modelling with definitive scripts
5. Semantics of definitive scripts
6. Agent-oriented modelling over definitive state representations

Application of EM to timetabling

7. Observation, dependency and agency in timetabling
8. The timetabling instrument in practice
9. Features and matters arising

Summary and Conclusion

Figure 4: Screenshot of the Temposcope

4

Timetabling *application*

Characteristic features:

- interface is designed for a specific functionality
  e.g. enter staff availability, create schedule

- the requirement for the application has been devised in conjunction with a preconceived administrative process
  e.g. timetable automatically generated after data input

- separation of roles between user and the developer
  e.g. user control limited to parametrisation of environment

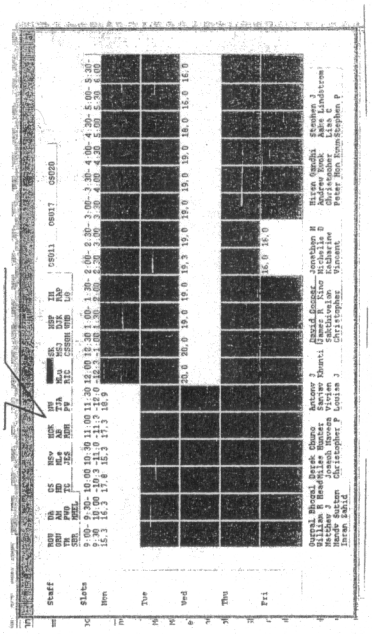
- procedures are designed and optimised to suit their function
  e.g. for large timetables, need sophisticated algorithms

Application supports

- efficient execution of preconceived functions

- optimised to suit the capabilities of the user

- and the characteristics of the computer



3

6

Timetabling *instrument*

Characteristic features:

- interfaces are developed *ad hoc* to suit opportunistic purpose
  > no preconceived restriction on what display means
  e.g. use grid cells to display a staff member's initials

- there is no predetermined administrative process
  > pattern of interaction is open-ended and uncommitted
  e.g. availability data is not required in advance

- conflation of roles of user and the developer
  > user conceives new modes of use and adapts system
  e.g. can introduce colour coding of themes on-the-fly

- computer model reflects current states of mind
  > automated activity is focussed on maintaining state (cf spreadsheet), not on effecting transitions
  e.g. maintains number of staff available in a given slot

Instrument supports

- continuous engagement of the user

- involving experimental interaction

- and a negotiation of meaning

4

7

Orientation for talk

Customarily argue *all computational paradigms equivalent*

? can one program be an application / another an instrument

... not concerned with *abstract programs* but with *physical states*

Shift in emphasis:
not *program as algorithm* but *computer as artefact*

Flexibility in use doesn't stem from adapting abstract programs but from reinterpreting physical phenomena

No comprehensive method

- to interpret an abstract program as a physical phenomenon
  > how program manifests is beyond program specification

- to realise a physical phenomenon via an abstract program
  > need to shape aspects of computer impact empirically

5

Adopt approach to construct instruments ... **Empirical Modelling**

To extent that current paradigms can deliver necessary features, Empirical Modelling is proposed as a conceptual framework: (e.g. may feel that relations and views, triggering and rule-based systems etc already supply a basis for powerful instruments)

In so far as current programming paradigms fail to deliver, Empirical Modelling is proposed as a radical alternative

Important feature:

presume a link between how a computer model is constructed and how it can be interpreted and adapted (cf classical theory of computation: deemed unimportant how a behaviour is realised)

cf motivating concept behind OOA and OOD

Thesis of Empirical Modelling (EM)

- flexibility in interpreting and shaping interaction stems from understanding of phenomena
- understanding of phenomena is fundamentally about relating them to observables, agents and dependencies
- there are many different ways to construe a phenomenon in these terms, and some are more useful than others
- it is perverse to try to construct a computer model to represent a phenomenon without reference to a construal i.e. without invoking observation, agency & dependency
- the quality of a computer model as a representation of a phenomena is determined by the quality of the construal on which it is based, and the extent to which its construction is faithful to this construal
- *experiment and observation* is the mediator of quality

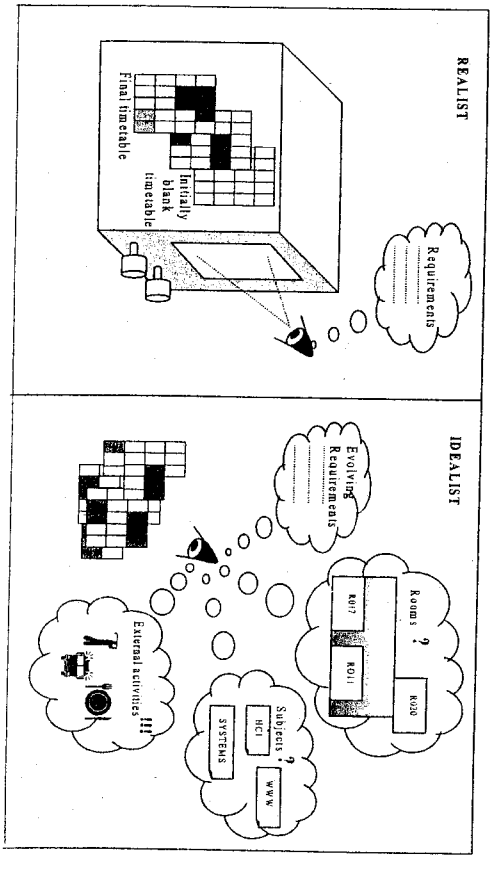


Figure 1: Realist and Idealist perspectives on timetabling

Why is the *instrument* perspective important in timetabling?

Human and computer co-operation is essential
Experiential aspects of timetabling activity are significant
- effective interfaces condition quality of the timetable
- user / automated actions ideally need to be integrated

Flexibility in the administrative process is essential
Functionality has to be adaptable

- unexpected changes and anomalies in the world
- not only designing for function, but adaptation of function
- impossible to preconceive the adaptations that needed

The Realist vs. the Idealist Timetabler

Realist perspective favours ...

use of the computer to spare the user effort
to diminish the need for intellectual engagement of user

Idealist perspective favours ...

use of the computer to enhance the impact of user effort
to enable more subtle and powerful intellectual engagement

EM is aimed at the idealist, and at job satisfaction not efficiency
Promotes computers as life enhancing rather than labour saving

Presumes problems on the human scale, whence case study

Dependency maintenance in computing ....

- the spreadsheet - accountancy and finance

  what if?, macro generation, culture of use

- Brian and Geoff Wyvill - geometric modelling (early 1970s)

  visualisation, integration of design and simulation

- Todd - prototype relational database systems (late 1970s)

  data modelling, functional dependencies, views

- modelling with scripts of definitions ("**definitive scripts**")

  combines qualities of all 3 dependency maintenance systems

  aspires to use dependency maintenance in a principled manner

Empirical Modelling (EM), University of Warwick since 1982

http://www.dcs.warwick.ac.uk/modelling


```
%eddi
ALLFRUITS (NAME char key, BEGIN int, END int);
ALLFRUITS << ["granny",8,10],["lemon",5,12],["kiwi",5,6],["passion",5,7];
ALLFRUITS << ["orange",4,11],["grape",3,6],["lime",4,7],["pear",4,8];
ALLFRUITS << ["cox",1,12],["red",4,8];

APPLE (NAME char key, PRICE real, QNT int);
APPLE << ["cox",0.20,8],["red",0.35,4],["granny",0.25,10];

CITRUS (NAME char key, PRICE real, QNT int);
CITRUS << ["lime",0.30,3],["orange",0.55,8],["kiwi",0.75,5],["lemon",0.50,2];

SOLDFRUIT (NAME char key, UNITSOLD int);
SOLDFRUIT << ["cox",100],["granny",153],["red",70];
SOLDFRUIT << ["kiwi",23],["lime",15],["lemon",55],["orange",78];

FRUITS is ALLFRUITS & NAME;
POPCITRUS is (FRUITS.CITRUS & NAME).(SOLDFRUIT : UNITSOLD > 50 & NAME);
```

```
?ALLFRUITS;
```

NAME	BEGIN	END
granny	8	10
lemon	5	12
kiwi	5	6
passion	5	7
orange	4	11
grape	3	6
lime	4	7
pear	4	8
cox	1	12
red	4	8
pineapple	2	6

```
?CITRUS;
```

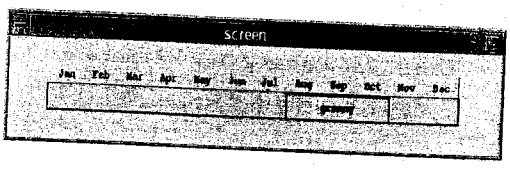
NAME	PRICE	QNT
orange	0.55	8
kiwi	0.75	5
lemon	0.5	2

```
EARLYCITRUS is (CITRUS*ALLFRUITS:BEGIN<=4)%NAME;

?EARLYCITRUS;
```

NAME
orange

```
CITRUS << ["lime",0.3,3];

?EARLYCITRUS;
```

NAME
orange
lime




```
%scout
point rlpos = {100, 100};
box br1 = {rlpos, rlpos + {500, 25}};
window record1 = {
    type: DONALD
    box: br1
    pict: "view1"
    bgcolor: "lightgrey"
    xmin: 0
    xmax: 120
    ymin:0
    ymax:10
    border: 1
    sensitive: ON
};
screen = <record1/record0>;

%donald
viewport view1
openshape bar1
within bar1 {
  label L
  int begin, end, height, length
  line W, E, S, N
  point SE, SW, NE, NW
  L = label{fruitname!, SW+{length div 2, 5}}
  W = [NW, SW]
  E = [NE, SE]
  S = [SW, SE]
  N = [NW, NE]
  SE = SW + {length, 0}
  SW = {(begin-1)*10, 0}
  NE = SW + {length, height}
  NW = SW + {0, height}
  height = 10
  length = (end-begin+1)*10
}

%eden
_bar1_begin is (indexfruit==0)? 0: ALLFRUITS[indexfruit][2];
_bar1_end is (indexfruit==0)? 0: ALLFRUITS[indexfruit][3];
fruitname is "granny";
indexfruit is name2index(fruitname);

func name2index{
para s;
auto i,result;
        result = 0;
        for(i=2;i<=ALLFRUITS#;i++){
                if(ALLFRUITS[i][1]==s)
                        result=i;
        }
        return result;
};
```

What is a definitive script?

16

Script is made up of a family of definitions

script	::=	*family_of_definitions*
definition	::=	*variable = formula*
formula	::=	*explicit_value*
	|	*function (list_of_variables)*

In natural use, *script* has static & dynamic interpretations:
- cf *examination script* and *actor's script*

Definitive script principally seen as *static* - standard interpretation

- definitive script defines a **state**
- new definition or re-definition defines a **transition**

Notes

- there is scope for *parallel* redefinition (true concurrency)
- script can also be interpreted as a sequence of definitions:
  dynamic intepretation as a recipe for state-change
- both static and dynamic interpretations typically ambiguous:
  cf candidate vs examiner's view of exam script
  cf playwright vs actor's interaction with play script

Computer support for definitive scripts

17

a *definitive notation* = simple notation for formulating definitions

underlying algebra
= family of data values + operators on these data values

emphasis is on the *experiential significance* of data values

e.g. data values can be points and lines + geometric operators

Implement a definitive notation via an evaluator that

- permits specification of definitions, functions and actions
  function = user-defined operator for use in definition
  action = procedure triggered by variable (active values)
  e.g. to invoke redefinition, drive display etc
- keeps all dependent values up-to-date and schedules action
  along with interactive user input in a 'concurrent' fashion

nterpreter for this purpose at Warwick: EDEN

DEN combines definitive notations to handle
lists, scalars, strings
points, lines, shapes, labels
displays, windows, text

The Semantics of Definitive Scripts

18

Orientation

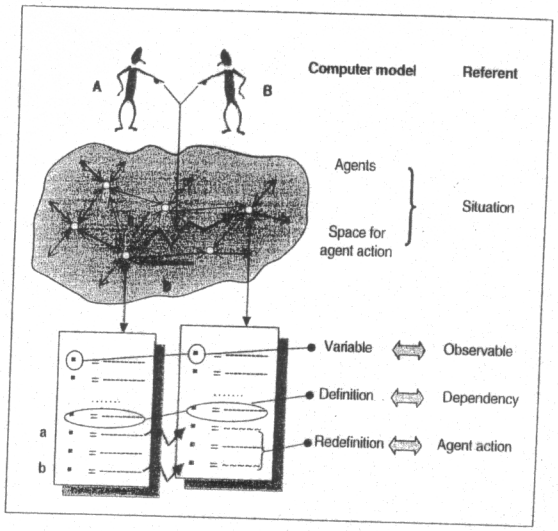
Characteristic of dependency maintenance applications

referent is open-ended in the following sense:

- exploratory experimental character of interaction
  what if? in spreadsheet
  'geometric instrument' in Wyvill
- acceptability is determined by the situation now in mind
  content of database determined by real-world state
  interaction with spreadsheet may be speculative
- subjective judgements may be involved
  geometric models shaped by *what looks good*
  price at which product is sold determined
- possible future states not circumscribed, not preconceived
  can't predict database content or status of balance sheet
  don't know all useful geometric designs and dependencies

referent can evolve
extend / refine database as a non-terminating application
can re-interpret a geometric model and re-orient design

19



Empirical Modelling for computer-based construals