EM and the Development Process

To what extent can techniques such as agile development and scripting resolve the problems of meeting users' evolving requirements?

Can the demands of users of the web, mobile devices and embedded systems for ways to customise applications flexibly in the stream-of-thought be addressed within the conceptual framework of software development as currently understood?

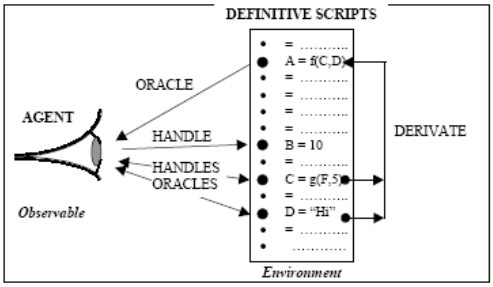


Figure 2-18: Definitive script as observer's model of state ('one-agent' modelling)

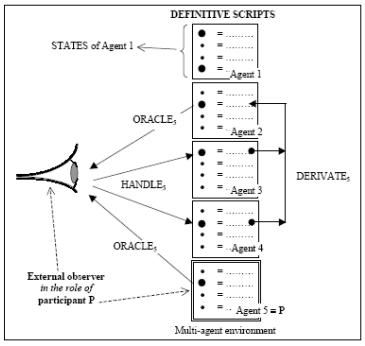


Figure 2-19: Definitive script as observer's model of state ('multi-agent' modelling)

Observables

Observables are entities

- whose identity is established through experience
- whose current status can be reliably captured by experiment

Can be physical, scientific, private, abstract, socially arbitrated, procedurally defined etc.

Dependency and Agency

- An *agent* is an observable (typically composed of a family of co-existing observables) that is construed to be responsible for changes to the current status of observables
- A *dependency* is a relationship between observables that - in the view of a state-changing agent - expresses how changes to observables are indivisibly linked in change

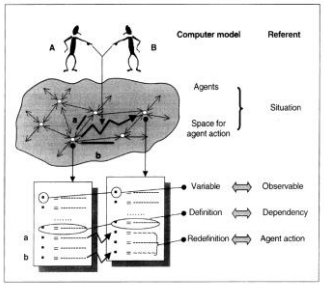
Agents

Agents are responsible for state-changes:
- meta-agents: e.g. the model builder
- agents determining model behaviour

Observables mediate agent actions/interactions

Use 'LSD notation' to specify perceptions and protocol (= *privileges*) of agents

Examples
- meta-agent: software developer; architect
- agent: users, devices; room user, door



Empirical Modelling for computer-based construals

Virtues of a definitive script

- represents view (cf spreadsheet)
- variables correspond to observables
- hides invisible activity
- can represent indivisibility in action

... when interpreted with agent protocol

- allows experimental basis of knowledge
- reflects different status of parameters

... supports open-ended incremental and distributed development

Roles for modelling with definitive scripts

Definitive scripts support artefacts that help developers

- to identify reliable interactions with their environment

- to recognise when there is a working understanding

- enable complex co-operative behaviour

- to construe complex system behaviour as agent interaction

… a formal stance neglects the empirical basis for knowledge of reliable systems that embraces all such activities

EM vs traditional modelling

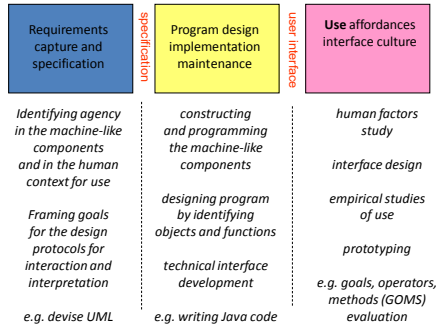
- conflate concerns
- represent via metaphor
- support ambiguity
- encourage customisation
- expose empirical roots
- are shaped by construal

- separate concerns
- represent symbolically
- expect/impose precision
- promote standardisation
- hide empirical foundation
- discard explanation

Two emphases

- Empirical Modelling encourages us to consider programming in a holistic way, using similar principles to deal with the entire process of development from conception to customisation and use
- It also has a means to represent the specific activity that is captured by a traditional program (a “pseudo-program”)

Traditional programming



Empirical Modelling

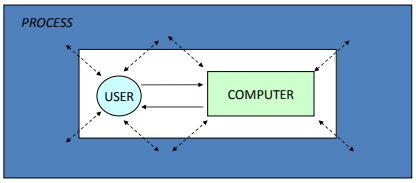
Requirements capture and specification	Program design implementation maintenance	**Use** affordances interface culture
develop scripts in isolation as "furry blobs" that represent the observables and dependencies associated with putative machine-like components and human interactions and interpretations	*identify and document reliably reproducible sequences of redefinition / chains of "furry blobs" that correspond to programmable automatable machine behaviours and ritualisable human behaviours and interfaces*	*exercise, explore, customise, revise and adapt sequences of redefinition and interpretation to reflect emerging and evolving patterns of interaction and interpretation; extend and augment observables to support additional functionalities combining scripts*

Rethinking programming …

… formal specification from an observation-oriented perspective

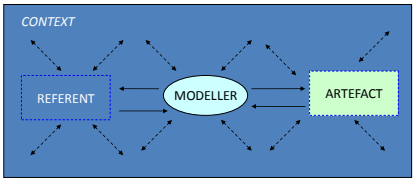
Programming from two perspectives

- a program is conceived with reference to how its behaviour participates in a wider process with functional objectives: states emerge as the side-effects of behaviours
- a computer artefact is developed so as to reflect the agency within an environment: the artefact and environment evolve until (possibly) program-like processes emerge



Conventional programs as embedded in *processes* of interaction with the world

Programs are understood in relation to processes in their surrounding environment

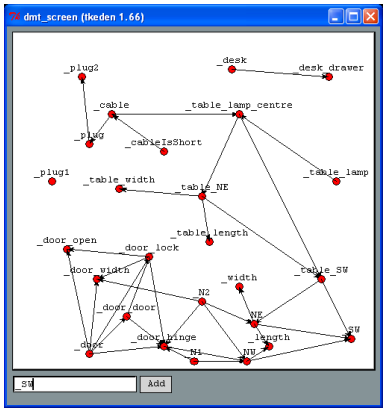


Artefacts and their referents as sculpted out of open interaction with the world

States of the referent and the artefact are connected through experience of interacting with the referent and the artefact

Objects and dependencies

- An **object** corresponds to a particular way of associating observables: grouping together observables according to whether they exist concurrently
- A **dependency** links observables according to how they are linked in change: whether making a change to the value of one observable necessarily entails changing others



Object model vs. account of observation

An account of observation is in some respects a more primitive concept than an object model: it entails fewer preconceptions about what might be observed …

"**Definitive scripts are neutral wrt agent's views & privileges"**

Object model vs. account of observation 2

Definitive script expresses different agent views and privileges to transform

(cf. subject-oriented programming)

"What architect can do vs what user can do"

… highlights how the script affords *views of* and *access to* possible transformations

Objects vs observations 1

A definitive script

represents the atomic transformations of a geometric symbol

DoNaLD room can be transformed through redefinition in ways that correspond 'exactly' to the observed patterns of change associated with opening a door, or moving a table

Objects vs observations 2

Thesis:

- set of atomic transformations of a symbol captures its semantics [cf. Klein's view of a geometry as "the study of properties invariant under a family of transformations"]
- Illustration via a geometric pun (demo)

Is the DoNaLD room an object in the class-based OOP sense? 1

Can view each room transformation as a method for the object

BUT

definitive script is an object specification

only if

set_of_transformations_performed_on_room is **circumscribed**

Is the DoNaLD room an object in the class-based OOP sense? 2

Circumscription creates objects

BUT

a definitive script merely reflects observed latent transformations

Comprehending / designing an object = knowing / determining everything we can do with it

BUT

definitive script doesn't circumscribe the family of transformations that we can apply

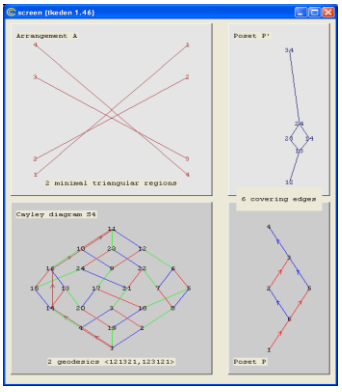
From logic to experience

- the computer enables us to use logical constructs to specify relationships that admit reliable interpretations and support robust physical realisations
- human skill and discretion plays a crucial role in crafting ritualisable experiences
- NB classical computer science doesn't take explicit account of robust physical realisations or ritualisable experience

From experience to logic?

- open-ended interaction with what is experienced is a means to representing with a high degree of realism and subtlety (cf. the strained representation of observables in the Miranda 3D OXO)
- mathematical concepts such as abstract lines as "realised" in this fashion

linesBeynon1991



The linesBeynon1991 script …

```
real sc,size
point O,Ox,Oy
O_Ox_Oy=[-500,500],O-[size,size],O+[size,size]
size,sc = 380.0,1/size
real a12,a23,a34,b12,b23,b34
point A1,A2,A3,A4,B1,B2,B3,B4
line l1,l2,l3,l4
#a12 and b12 determine the distances between the L and R ends of lines 1 and 2
a12=1.0
a23=1.0
a34=2.0
b12=2.0
b23=5.0
b34=1.0
#A1 and B1 are the left and right endpoints of line 1
A1=Ox
A2=Ox+[0, a12 div (a12+a23+a34)]*sc
A3=Ox+[0,(a12+a23) div (a12+a23+a34)]*sc
A4=Ox+[0,(a12+a23+a34) div (a12+a23+a34)]*sc
B1=Ob
B2=Ob-[0,(b12) div (b12+b23+b34)]*sc
B3=Ob-[0,(b12+b23) div (b12+b23+b34)]*sc
B4=Ob-[0,(b12+b23+b34) div (b12+b23+b34)]*sc
l1=[A1,B1]
l2=[A2,B2]
l3=[A3,B3]
l4=[A4,B4]

label j,j2,j3,j4,k1,k2,k3,k4
j1 = label("j",A1)
j2 = label("j",A2)
j3 = label("j",A3)
j4 = label("k",A4)
k1 = label("j",B1)
k2 = label("j",B2)
k3 = label("j",B3)
k4 = label("k",B4)
```

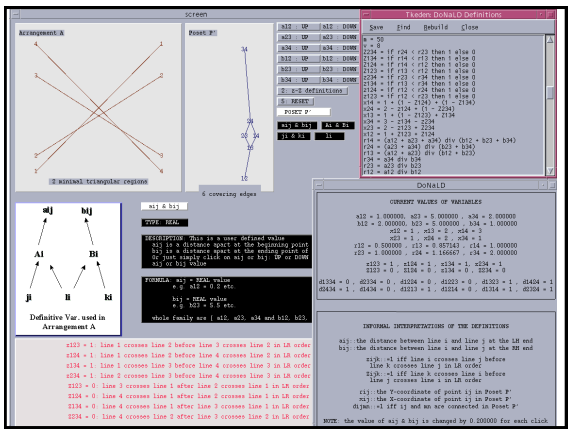
```
#this is the DoNaLD script that defines Figure 1(b) in the viewport "POSET"
viewport POSET

#r12 determines the LR position of the line 1_2 intersection
#s12 is to be the crossing index of the line 1_2 intersection
#   eg: this is 1 if lines 1 and 2 are the top pair at their pt of IN
real r12,r23,r34,r13,r24,r14
int s12,s23,s34,s13,s24,s14
r12,r23,r34 = a12 div b12 , a23 div b23, a34 div b34
r13 = (a12+a23) div (b12+b23)
r24 = (a23+a34) div (b23+b34)
r14 = (a12+a23+a34) div (b12+b23+b34)

#s123 = 1 if line 1 crosses line 2 before line 3 crosses line 2 in LR order
int z123,z124,z134,z234
int ZZ3,Z124,Z134,Z234
z123 = if r12<r23 then 1 else 0
z124 = if r12<r24 then 1 else 0
z134 = if r13<r34 then 1 else 0
z234 = if r23<r34 then 1 else 0
Z123 = if r13<r12 then 1 else 0
Z124 = if r14<r12 then 1 else 0
Z134 = if r14<r13 then 1 else 0
Z234 = if r24<r23 then 1 else 0

#s12 calcs the crossing index for lines 1 and 2
# "by default" this is 1 but is incremented if line 3 or 4 crosses line 1
#before line 2 crosses line 1 in LR order
s12 = 1-Z123+Z124
s13 = 1-(1-Z123)+Z134
s14 = 1-(1-Z124)-(1-Z134)
s23 = z123+Z234
s24 = 2-s12+(1-Z234)
s34 = 3-s134-z234
```

```
int U1213,U1214,U1314,U2324,U1223,U1224,U1234,U1334,U2334,U1323,U1424,U2434,U1434
U1213 = g1213 * r1213 * d1213
U1214 = g1214 * r1214 * d1214
U1314 = g1314 * r1314 * d1314
U2324 = g2324 * r2324 * d2324
U1223 = g1223 * r1223 * d1223
U1224 = g1224 * r1224 * d1224
U1334 = g1334 * r1334 * d1334
U2334 = g2334 * r2334 * d2334
U1323 = g1323 * r1323 * d1323
U1424 = g1424 * r1424 * d1424
U1434 = g1434 * r1434 * d1434
U2434 = g2434 * r2434 * d2434

int s1213,s1214,u1314,u2324,u1223,u1224,u1334,u2334,u1323,u1424,u2434,u1434
u1213 = g1213 * (1-r1213) * d1213
u1214 = g1214 * (1-r1214) * d1214
u1314 = g1314 * (1-r1314) * d1314
u2324 = g2324 * (1-r2324) * d2324
u1223 = g1223 * (1-r1223) * d1223
u1224 = g1224 * (1-r1224) * d1224
u1334 = g1334 * (1-r1334) * d1334
u2334 = g2334 * (1-r2334) * d2334
u1323 = g1323 * (1-r1323) * d1323
u1424 = g1424 * (1-r1424) * d1424
u1434 = g1434 * (1-r1434) * d1434
u2434 = g2434 * (1-r2434) * d2434

int v1213,v1214,v1314,v2324,v1223,v1224,u1334,u2334,u1323,u1424,u2434,u1434
v1213 = (1-g1213) * (1-r1213) * d1213
v1214 = (1-g1214) * (1-r1214) * d1214
v1314 = (1-g1314) * (1-r1314) * d1314
v2324 = (1-g2324) * (1-r2324) * d2324
v1223 = (1-g1223) * (1-r1223) * d1223
v1224 = (1-g1224) * (1-r1224) * d1224
v1334 = (1-g1334) * (1-r1334) * d1334
v2334 = (1-g2334) * (1-r2334) * d2334
v1323 = (1-g1323) * (1-r1323) * d1323
v1424 = (1-g1424) * (1-r1424) * d1424
v1434 = (1-g1434) * (1-r1434) * d1434
v2434 = (1-g2434) * (1-r2434) * d2434
```

```
int V1213,V1214,V1314,V2324,V1223,V1224,V1334,V2334,V1323,V1424,V2434,V1434
V1213 = (1-g1213) * r1213 * d1213
V1214 = (1-g1214) * r1214 * d1214
V1314 = (1-g1314) * r1314 * d1314
V2324 = (1-g2324) * r2324 * d2324
V1223 = (1-g1223) * r1223 * d1223
V1224 = (1-g1224) * r1224 * d1224
V1334 = (1-g1334) * r1334 * d1334
V2334 = (1-g2334) * r2334 * d2334
V1323 = (1-g1323) * r1323 * d1323
V1424 = (1-g1424) * r1424 * d1424
V1434 = (1-g1434) * r1434 * d1434
V2434 = (1-g2434) * r2434 * d2434

%donald
int n12, n23, n24, n34, n14, n13
n13 = U1314+U1334+U1323+u1213+v1213+V1334+V1323+V1314
n23 =U1324+U2334+u1223+u1323+v1223+V1324+V2334
n24 =U2434+u2324+u1224+u1424+V2434+v2324+v1224+V1424
n34 =u2334+u2134+u1434+u2434+v1334+v2334+v1434+V2434
n14 =U1424+U1434+u1214+u1314+V1424+V1434+v1214+v1314
n12 =U1213+U1214+U1223+U1224+V1224+V1223+V1213+V1214

int V1213,V1214,V1314,V2324,V1223,V1224,V1334,V2334,V1323,V1424,V2434,V1434
V1213 = (1-g1213) * r1213 * d1213
V1214 = (1-g1214) * r1214 * d1214
V1314 = (1-g1314) * r1314 * d1314
V2324 = (1-g2324) * r2324 * d2324
V1223 = (1-g1223) * r1223 * d1223
V1224 = (1-g1224) * r1224 * d1224
V1334 = (1-g1334) * r1334 * d1334
V2334 = (1-g2334) * r2334 * d2334
V1323 = (1-g1323) * r1323 * d1323
V1424 = (1-g1424) * r1424 * d1424
V1434 = (1-g1434) * r1434 * d1434
V2434 = (1-g2434) * r2434 * d2434

%donald
int n12, n23, n24, n34, n14, n13
n13 = U1314+U1334+U1323+u1213+v1213+V1334+V1323+V1314
n23 =U1324+U2334+u1223+u1323+v1223+V1324+V2334
n24 =U2434+u2324+u1224+u1424+V2434+v2324+v1224+V1424
n34 =u2334+u2134+u1434+u2434+v1334+v2334+v1434+V2434
n14 =U1424+U1434+u1214+u1314+V1424+V1434+v1214+v1314
n12 =U1213+U1214+U1223+U1224+V1224+V1223+V1213+V1214
```

Interesting comparisons …

- the lines script as not object-oriented – most of its core observables are associated with relationships that cannot be identified with any single object
- the lines script as resembling a functional programming script in its homogeneity (“all definitions”), but associated with directly accessible external observables …



Features of the lines model …

- directly accessible external observables: z123 = 1 means that line 1 crosses line 2 before line 3 crosses line 2 in L-to-R order
- the ideal geometry as associated with a mode of interaction with the model (subject to being able to enhance the accuracy of arithmetic indefinitely on-the-fly)