Programming from an Empirical Modelling perspective

From modelling with definitive scripts to programming:
- representing state in programming
- behaviour of programs
- the semantics of programs

States relevant to programming ...

- state within the executing program
- external state: what is visible?
- state in respect of interaction
- state in program development
- state significant in the external world

diverse representations are required:
- state within the executing program
  - Program variables, machine locations
- external state: what is visible?
  - Graphics / display techniques
- state in respect of interaction
  - Statechart, message sequence diagram

Diverse representations required ...
- state in program development
  UML diagrams, prototypes
- state significant in the external world
  apprehended by the human interpreter

cf. Brian Cantwell-Smith on semantics …
States within oxoJoy1994

Definitive scripts express …
- internal state – contents of squares
- visible state – appearance of the board
- interaction state: whose turn is it?
- state of development
- state of mind of the player: which square?

Dependencies in the oxoJoy1994 model

– note continuous spectrum of redefinitions

closes after being opened

OXO GAME

a nonsense redefinition

Definitive scripts as “furry blobs”

≡

- interaction state: whose turn is it?

PROGRAM

USER

INTERNAL STATE

GAME DESIGNER

PLAYER

OXO GAME

VISUALISATION

PROGRAMMER

≡

… compare this with the OXO laboratory

Diverse relations / representations in a traditional program

… all relations mediated by definitions

… Behaviour as programmed state change

INTERNAL STATE

GAME DESIGNER

OXO GAME

VISUALISATION

PLAYER

PROGRAMMER

Static and dynamic elements of state

Plausible: could open the desk drawer
– note continuous spectrum of redefinitions
Ritualised: door automatically closes after being opened
Nonsense: opening the drawer makes the room smaller
Traditional programming

- Requirements capture and specification
  - Identifying agency in the machine-like components and in the human context for use
  - Framing goals for the design protocols for interaction and interpretation
    - e.g. devise UML

- Program design implementation maintenance
  - Designing machine-like components
  - Developing by identifying objects and functions
  - Technical interface development
    - e.g. writing Java code

- Use affordances
  - Human factors study
  - Interface design
  - Empirical studies of use
  - Prototyping
    - e.g. goals, operators, methods (GOMS)
    - Evaluation

Modelling with definitive scripts:
... a holistic view of state that integrates and conflates all the different perspectives

in contrast to

Programming-in-the-wild:
... an eclectic model of state in which many different strategies for representation and interpretation are jumbled up together

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”)

Empirical Modelling

<table>
<thead>
<tr>
<th>Requirements capture and specification</th>
<th>Program design implementation maintenance</th>
<th>Use affordances interface culture</th>
</tr>
</thead>
<tbody>
<tr>
<td>develop scripts in isolation as “furry blobs” that represent the observables and dependencies associated with putative machine-like components</td>
<td>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</td>
<td>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</td>
</tr>
</tbody>
</table>

- Requirements capture and specification
  - Identifying agency in the machine-like components and in the human context for use
  - Framing goals for the design protocols for interaction and interpretation
    - e.g. devise UML

- Program design implementation maintenance
  - Designing machine-like components
  - Developing by identifying objects and functions
  - Technical interface development
    - e.g. writing Java code

- Use affordances
  - Human factors study
  - Interface design
  - Empirical studies of use
  - Prototyping
    - e.g. goals, operators, methods (GOMS)
    - Evaluation
**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”*

**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

The linesBeynon1991 script ...

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: \( z_{123} = 1 \) 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)

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
An EM perspective on programming …
… some problematic issues

In focusing on current state-as-experienced, we have some problems to resolve:

• Behaviour raises questions about agency: what is the status of a “computer” action?
• How do we deal with state-as-experienced in semantic terms?
• How do we make science of activities in which human interpretation is so critical?