T1: Visualisation and Concurrent Systems Modelling

T1.1. Role of Visualisation Tools

What connection has **visualisation**

with **concurrent systems modelling?**

ARGUE

To understand a complex system, must:

- analyse the system into agents
- model interfaces between agents in terms of observations

Visualisation is the *means*

to make activity at the agent interfaces "visible" to the designer

cf approach an engineer takes when building a complex device.

Analogy with conventional engineering

Engineer deals with components whose interaction is invisible

e.g. via electrical currents or hydraulic pressures

In constructing or diagnosing the faults in a complex system

engineer attaches equipment to the system

to make the interactions between components visible

measurements should reflect the activities they record

as accurately as possible

Accuracy

= precise correlation between stimulus and response

Ideally, a reliable and well-understood functional relationship

definitive scripts <--> interface mechanism to express correlation

visualisation aspect of the computer model of a complex system

<--> engineer's measuring instruments

"interface mechanism" because resembles mechanical linkage

that reliably and instantaneously transmits state change

Probing the system ....

The engineer's design tools:

- devices to provoke state-changes
- complemented by measuring instruments

The engineer probes the system by experiments

– initiating changes and studying the response

Many such experiments

<--> activities ultimately carried out by other agents in the system

Visualisation tools

= "measuring instruments" to aid concurrent systems design

Desirable to

- separate system model from the visualisation

cf attach instruments to a system to monitor behaviour

- perform visualisation in parallel with normal system operation

Definitive scripts can achieve this objective:

only need appropriate references for use in the RHS of formulae.

Visualisation unbounded:
why so many definitive notations?

Compare

what an engineer conceives as component interaction

e.g. in propagation of signals, or application of forces

with

what can observe directly, through visual or audible output

Typically

conceptual view of interactions within the system

uses notions **without** visual significance

=> developing visualisations for such concept

is a process that has no clearly prescribed limits

also other forms of output we experience directly, e.g. sonification

Each definitive notation for graphics is in its own way

"converting abstract concepts into something visual".

Each notation specifies a different relationship between

what the designer in mind <--> what is observed on the screen

Single-agent definitive programming

use of definitive notations for visualisation

= single agent definitive programming

where sole state changing agent is the system designer

Visualisation helps designer

- to grasp what is in fact invisible
- to model what a user does in practice see of the system

For instance, in the VCCS:

the speedometer

the cruise controller interface

the brake and pedal

represent the interface to the driver of the vehicle

the animation of the forces acting on the vehicle

the diagrammatic position of the vehicle

the clock

the throttle

assist the designer in understanding how the controller performs.

Definitive scripts for visualisation

– the representation of agent views

single agent = designer

designer has perspective on the views of many different agents

Constructing definitiive script for visualisation is

an exercise in knowledge representation, recording

- what agents can respond to
- what they can change

[cf fundamental notions of LSD specification, to follow ...]

In the VCCS:

the definitive programmer can make free with the script

the designer can modify it in any way that is

plausibly consistent with the real-world semantics

the mouse user can interact via the driver's interface

Analysis and representation of agent perceptions and privileges

is a first step towards

modelling and animation a concurrent system

T1.2. Some Illustrative Examples

T.1.2.1. The SCOUT Notation: Basic Principles

Assume designer-computer interaction

via the screen, the keyboard and the mouse

Consider visualisation in terns of

"state-changes to the screen"

different symbolic representations

model conceptually different kinds of observation

Three simple ways to describe patterns on the screen:

window layout	Scout
textual display	
2d-line-drawing	DoNaLD

Scout allows the designer to describe the whole screen layout

by devising windows in which to display

- graphical ouput in DoNaLD
- associated textual annotations

[Design and implementation of Scout is work of Simon Yung]

T1.2. Some Illustrative Examples (cont)

T1.2.2. The lines.sdae script

lines.sdae script is an exercise in visualisation for mathematics

animates combinatorial structures
associated with planar configurations of 4 lines

lines.sdae illustrates:

- combining conceptually different types of visualisation

  two DoNaLD + two ARCA windows

  combined and annotated within the SCOUT environment.

  Lines in ARCA and DoNaLD have a very different significance

  ARCA diagrams are abstract combinatorial graphs:

  can trace path of coloured directed edges in an ARCA diagram

  cf studying transitions in a finite state machine

- dependency relationships that cross conceptual boundaries

  configuration of lines *Donald*

  ~> ranked poset as functionally dependent diagram

  ~> ranked poset as combinatorial diagram *ARCA*

  ~> path through a Cayley diagram

  e.g. map oriented line into coloured combinatorial edge

  map sequence of ranks into combinatorial path

  handled by "bridging definitions" + an action!

T1.2.2. The lines.sdae script

Idealisation ....

- mathematical idealisation associated with the definitive script

  variables in the definitive script <---> Platonic points & lines

  visualisation is only as accurate

  as the computer arithmetic and screen hardware allows

  "can observe what we cannot simulate exactly on a computer"

  cf numerical solution of differential equations

T1.2.3. Putting rooms into windows: roomviewer.s script

roomviewer.s illustrates use of Scout

- to say where and how to display (say) a DoNaLD picture

  DoNaLD room appears in 2 windows, differently displayed

  In Scout independently specify
  - the location and shape of the graphics window
  - the portion of a DoNaLD picture to be displayed within it

  Can exploit to adapt a window to the size of its contents
  - as in the roomviewer.s menu buttons

  or to ensure that a picture always stays within a fixed frame
  - as in the DoNaLD poset in lines.sdae

- to connect textual annotations with

  picture content and other aspects of system state:

  monitoring the length of the cable *roomviewer.s*

  whether the table obstructs the door

  labelling the menu options to reflect status

  displaying invariants of line configurations *lines.sdae*

  These uses of text reflect different quite modes of interpretation

  e.g. warning to designer

  message to the user

  information to the mathematician

The roomviewer.s script (cont)

- to represent and agent privileges

  designer experiments in extending and elaborating scripts by adding new definitions or redefining existing variables

  can simulate agents with more restricted privileges

From Visualisation to Animation

definitive script can be used to model agent privileges

to generalise single-agent definitive programming:

shift emphasis from

identifying agent privileges

– *what can agent do in principle,*

to *enabling conditions for exercising them*

– *when and how it acts*

In animating, complement Scout with actions in Eden:

- can introduce mouse sensitive regions <-> redefinitions

  e.g. menu buttons in roomviewer.s

- can impose constraints on redefinitions e.g.

  so that cable can't be overextended in roomviewer.s

NB in generalising single-agent programming

traditional sequential programming not a natural special case

No reason why not parallel redefinition

as in "designing VCCS interface whilst simulation is running"

conventions to restrict agents to exercise privileges in set pattern

- don't help us at the initial stages of specification
- obscure the experimental foundation of behaviour

The JUGS program as an illustrative example .....

Issues discussed by Simon Yung in case study of JUGS

SCOUT: a definitive notation for SCreen layOUT
designed and implemented by Simon Yung

Case study: use of SCOUT for JUGS program prototyping

See tank.s, tank.e and tank.disp.e in ~wmb/public/demo/SCOUT

1) context for the JUGS problem

   about the problem

   - the key parameters
     capacity and content, emptiness
     menu status, target

   - the agents
     the interface designer
     the programmer
     the pupil
     the teacher

   - privileges of agents
     redesign the display
     design algorithms
     push buttons
     set problem parameters

2) modelling vs exploratory design

   modelling: following an existing screen layout?
   exploratory design: experimenting with new layouts?

3) the Screen Layout Modelling Process

   Three stages:

   a) Consider form of the screen display in program execution
      - conceive screen in particular state

   b) Identify the geometric entities and invariant relationships
      - identify what is generic about the display
        e.g. windows always in the same place
             fixed size
      - identify what dimensions / attributes represent
        e.g. reverse video => unavailable option
             number of tildes = quantity of water

   c) Define locations and contents of windows in SCOUT

4) Special-purpose nature of the SCOUT notation

   The assumptions that underlie SCOUT
   The advantages in use of application-oriented notations

5) The flexibility offered to the designer by a definitive script

   Representing the designer's privileges
   Comparison with other paradigms

6) Separation of Control and Presentation

control of parameters for JUGS screen not specified in SCOUT
=> faster program development / parallel development

7) Exploratory Screen Layout Development

Screen layout design by prototyping a first approximation
then iterating by experimentation

Activities performed in exploratory design in JUGS

- adding and deleting items
- relocating display items
- modifying relationships between variables
- testing the design

Easy undo actions

8) Flexible arrangement of definitions and views of the design

Different arrangements of definitions conceptually different
OOP perspective
adapt to strategies for design
e.g. conceive regions of display: global layout
vs window by window development

Possibility of automatic support for this