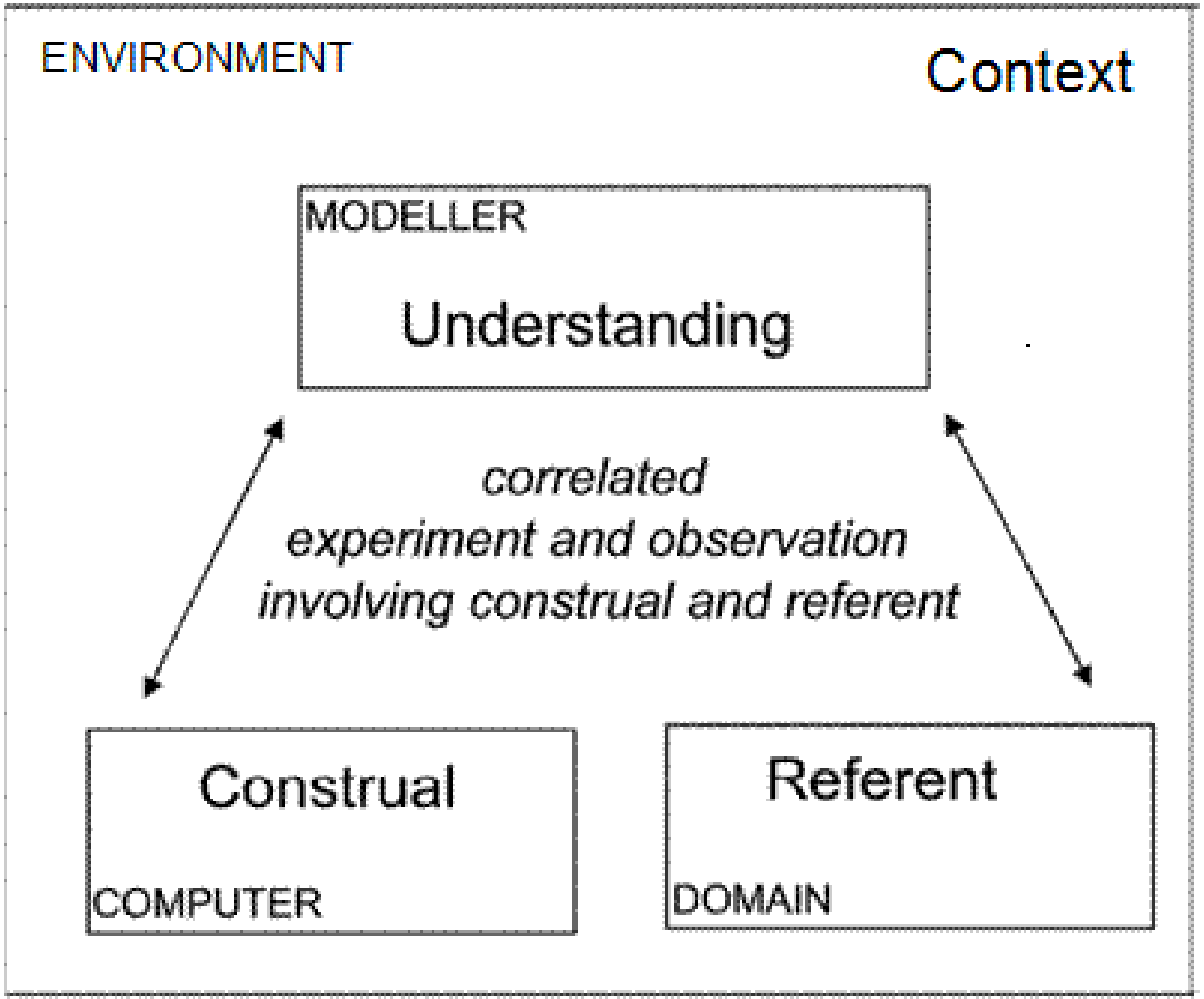
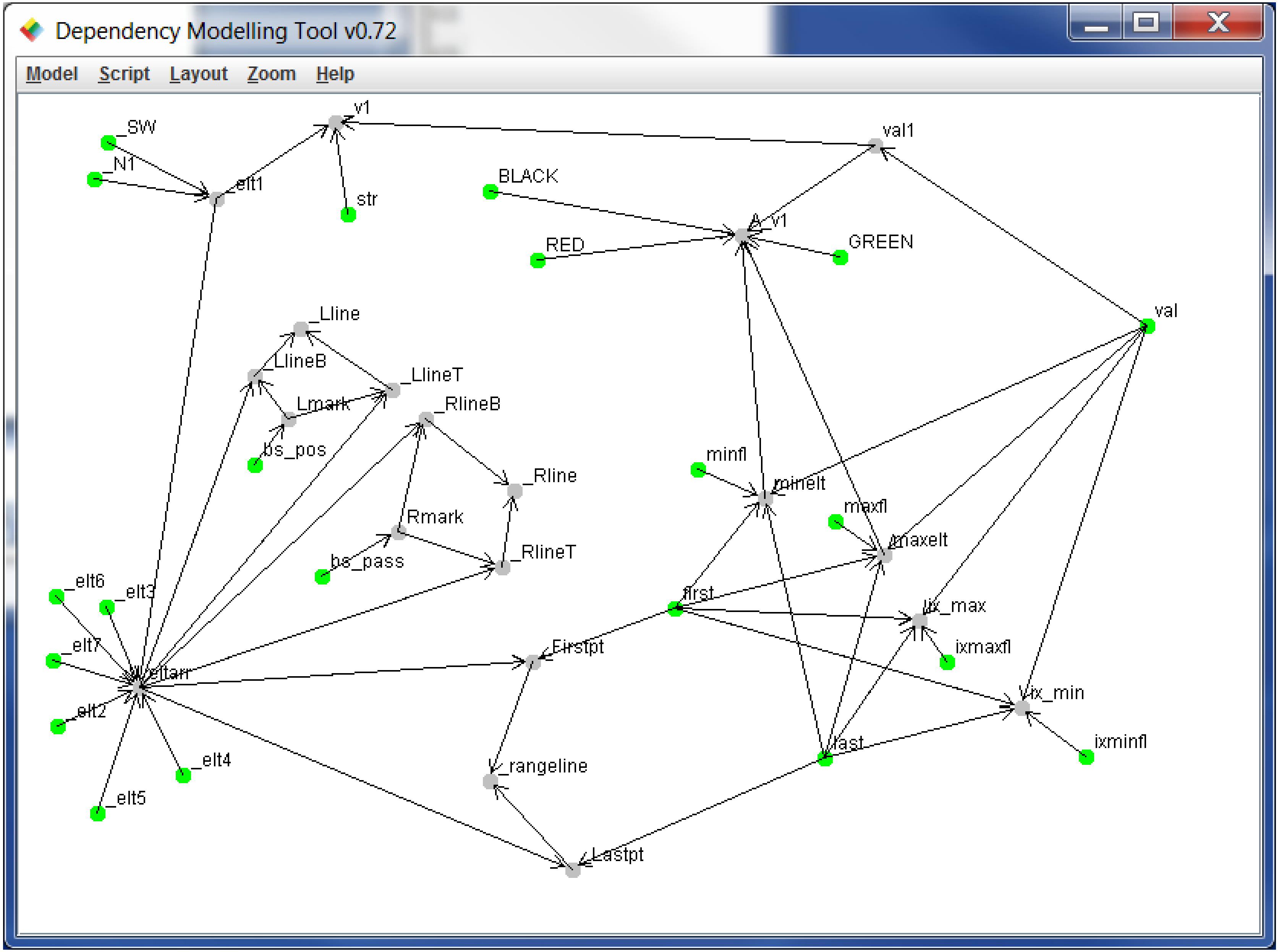
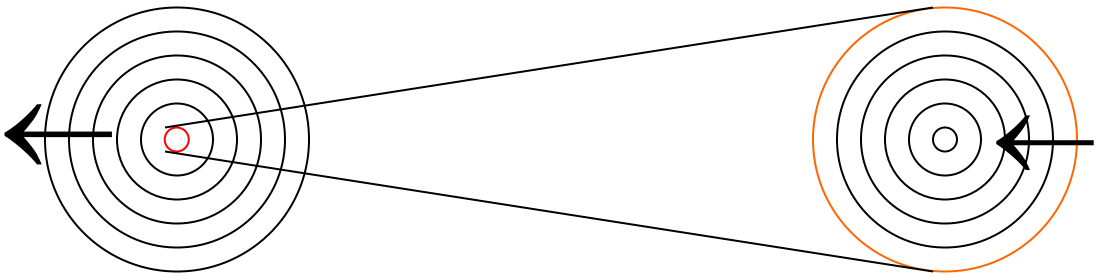
Empirical Modelling as *Construction*




Dependency Modelling Tool v0.72
Model
Script
Layout
Zoom
Help
v1
_SW
_N1
_elt1
str
val1
BLACK
_v1
RED
GREEN
_Lline
_LlineB
_LlineT
_Lmark
_RlineB
bs_pos
val
minfl
_Rline
_minelt
_Rmark
maxfl
_maxelt
_RlineT
bs_pass
_elt6
_elt3
first
_ix_max
_Firstpt
_elt7
ixmaxfl
_eltarr
_ix_min
_elt2
last
ixminfl
_elt4
_rangeline
_elt5
_Lastpt

The Onion Metaphor

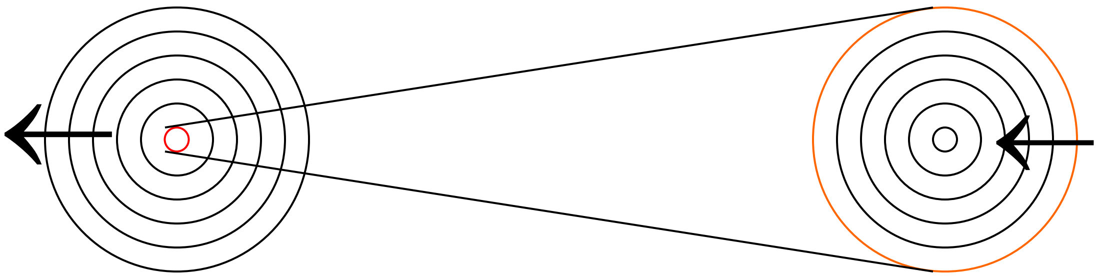


Theory building:
“Quality” of knowledge

Experimental understanding:
“Quantity” of interaction

core knowledge	*innermost*		least tested understanding
extending theory	↓	↑	*refining experiment*
	↓	↑	
speculative knowledge	*outermost*		most secure understanding

The Onion Metaphor



Theory building:
“Quantity” of knowledge

Experimental understanding:
“Quality” of interaction

least established theory	*innermost*	most refined interaction
extending theory	↓ ↑	*refining experiment*
	↓ ↑	
most stable theory	*outermost*	least restricted interaction

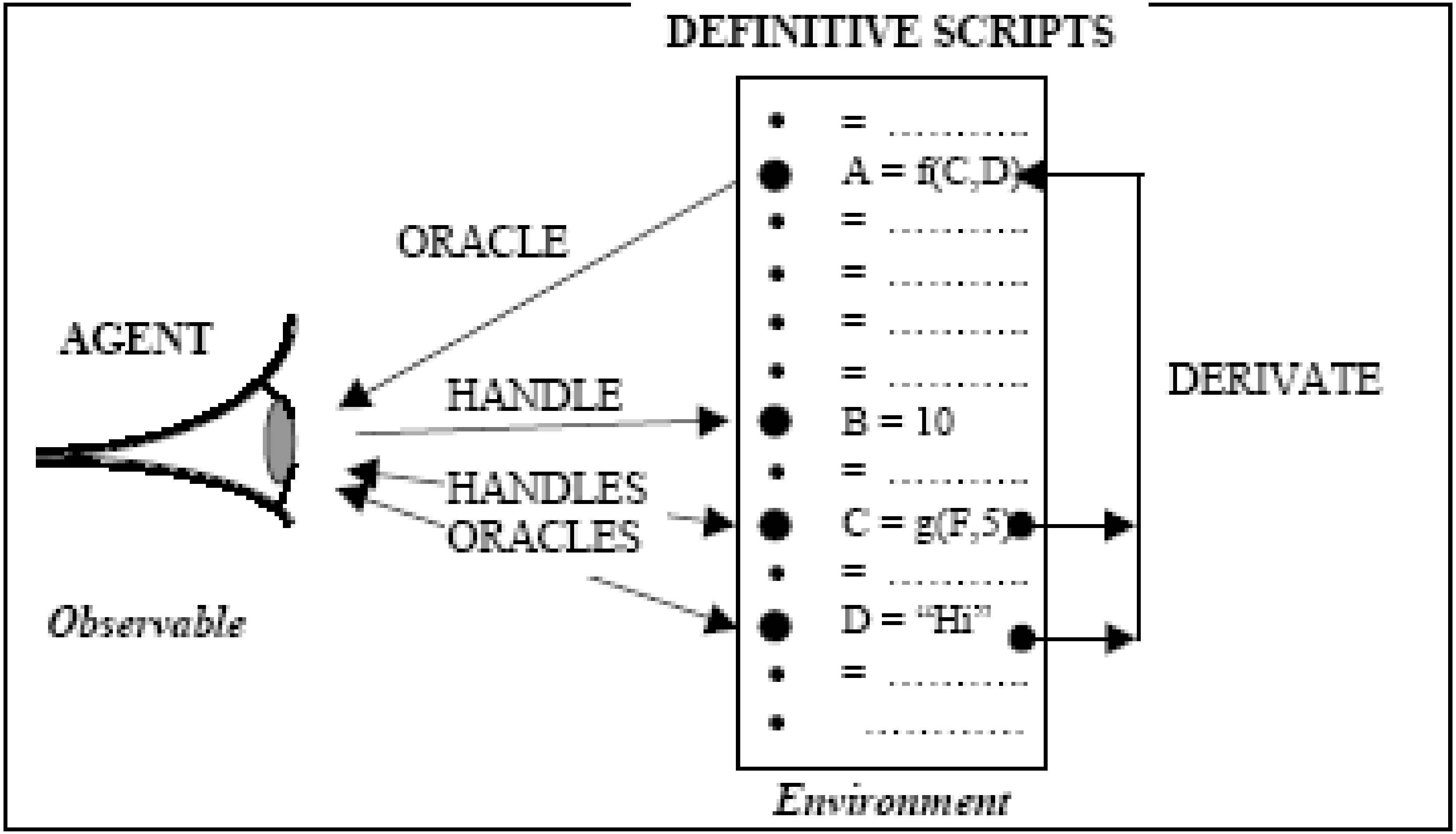


Figure 2-18: Definitive script as observer's model of state ('one-agent' modelling)

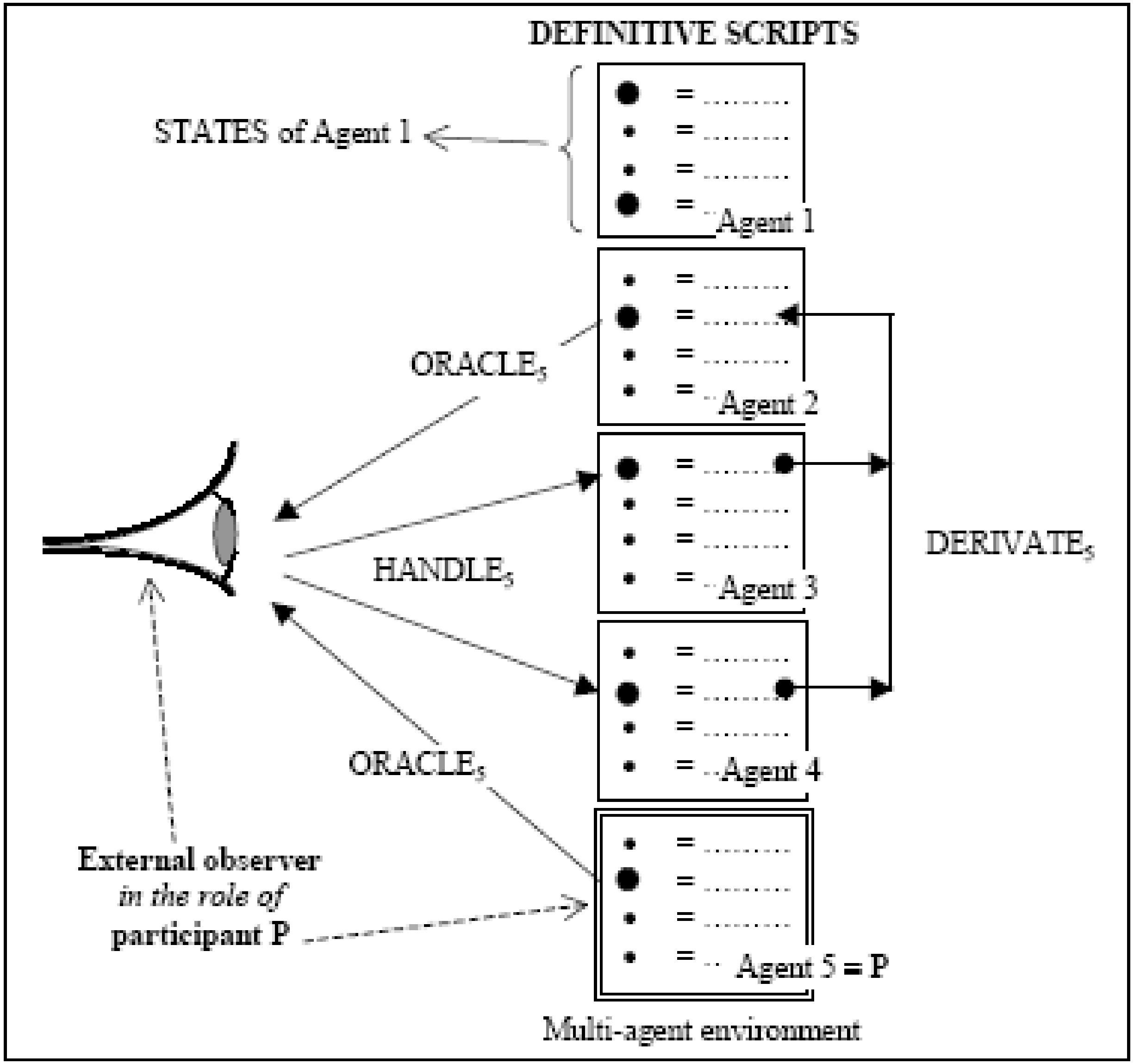
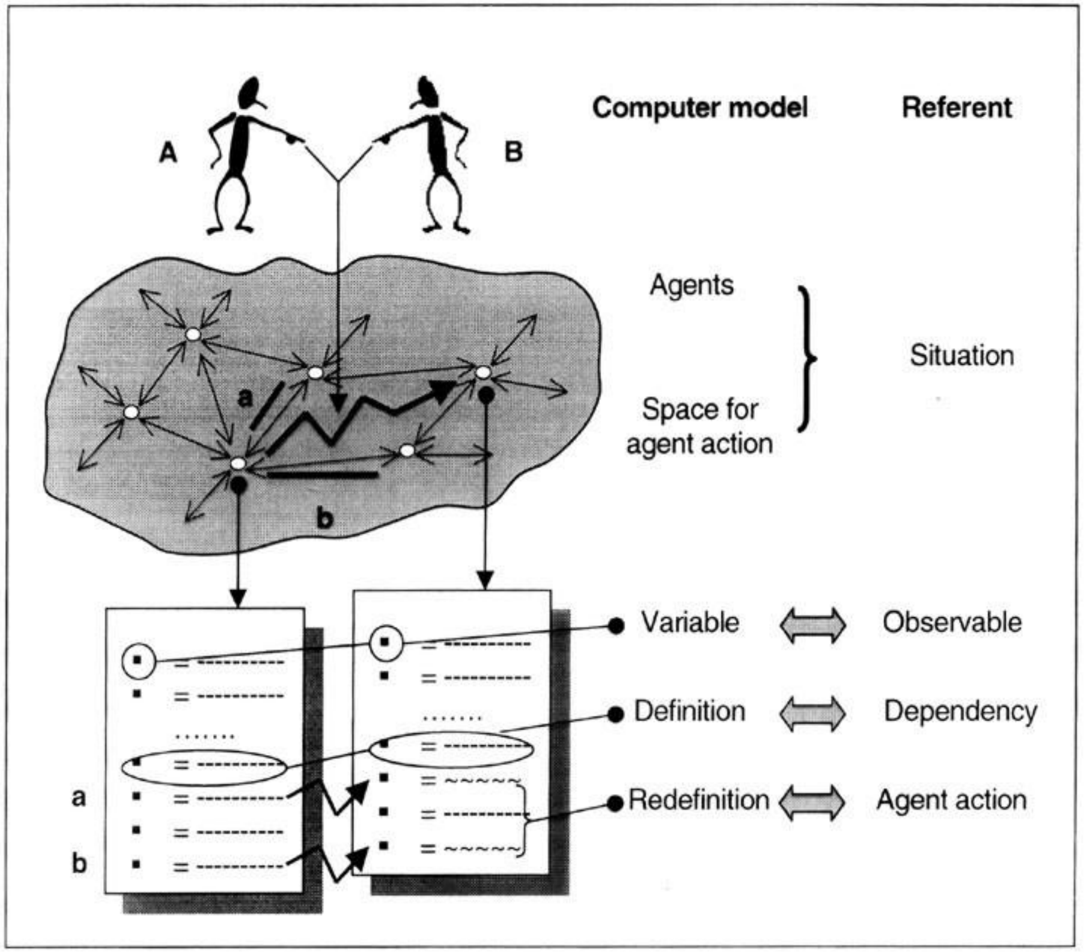


Figure 2-19: Definitive script as observer's model of state ('multi-agent' modelling)



Empirical Modelling for computer-based construals

private experience / empirical / concrete

interaction with artefacts: identification of persistent features and contexts

practical knowledge: correlations between artefacts, acquisition of skills

identification of dependencies and postulation of independent agency

identification of generic patterns of interaction and stimulus-response mechanisms

non-verbal communication through interaction in a common environment

directly situated uses of language

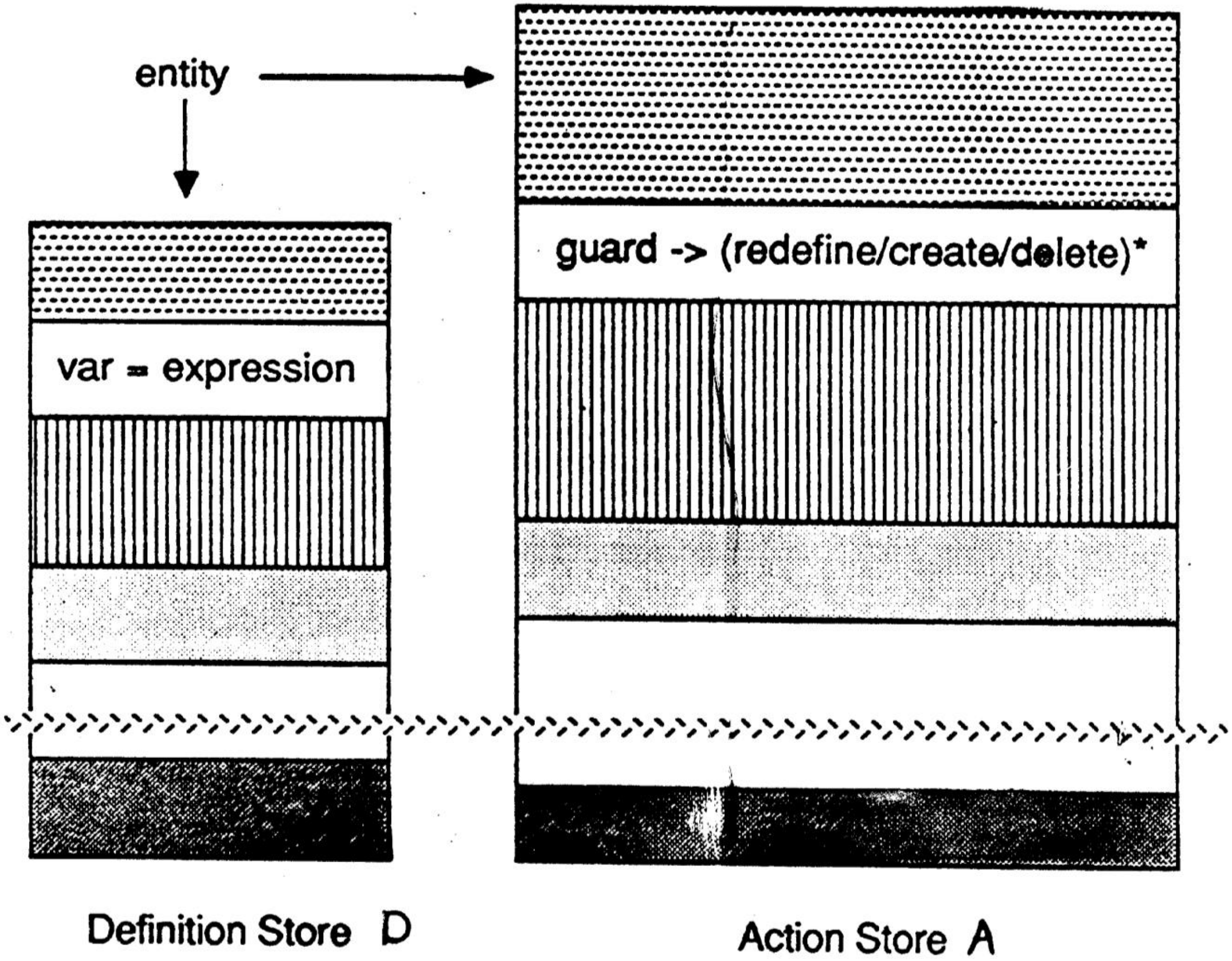
identification of common experience and objective knowledge

symbolic representations and formal languages: public conventions for interpretation

public knowledge / theoretical / formal

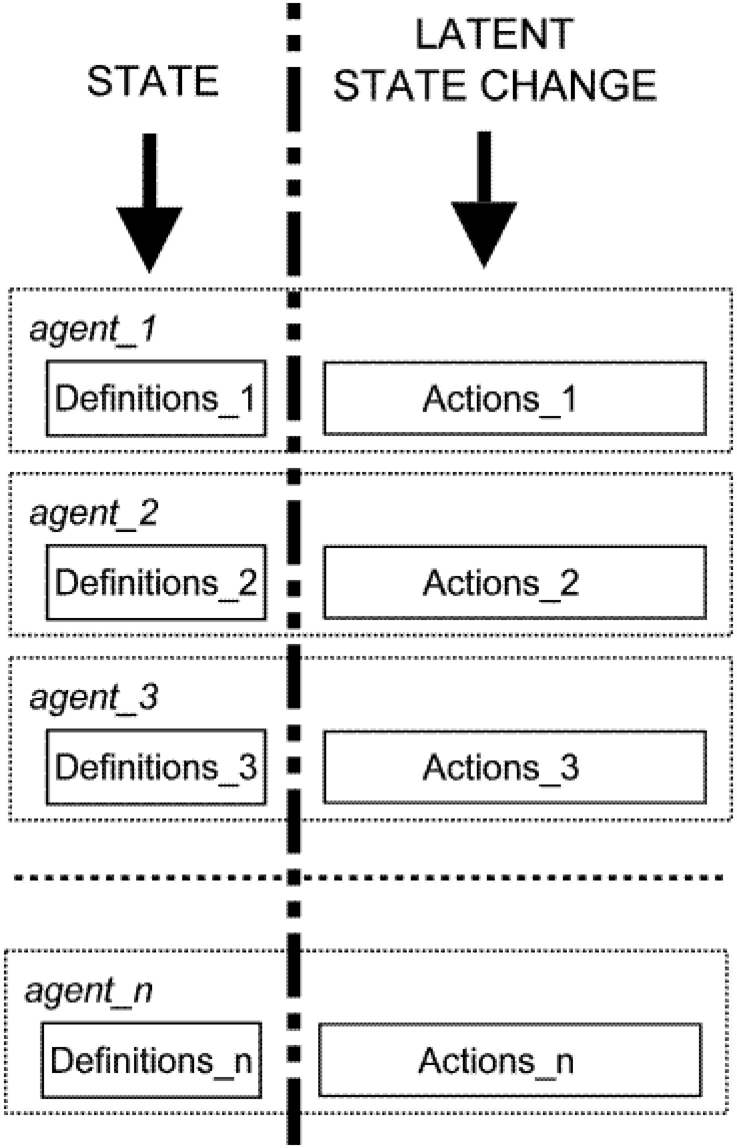
An Experiential Framework for Learning (EFL)

The Abstract Definitive Machine: entity = definitions + actions



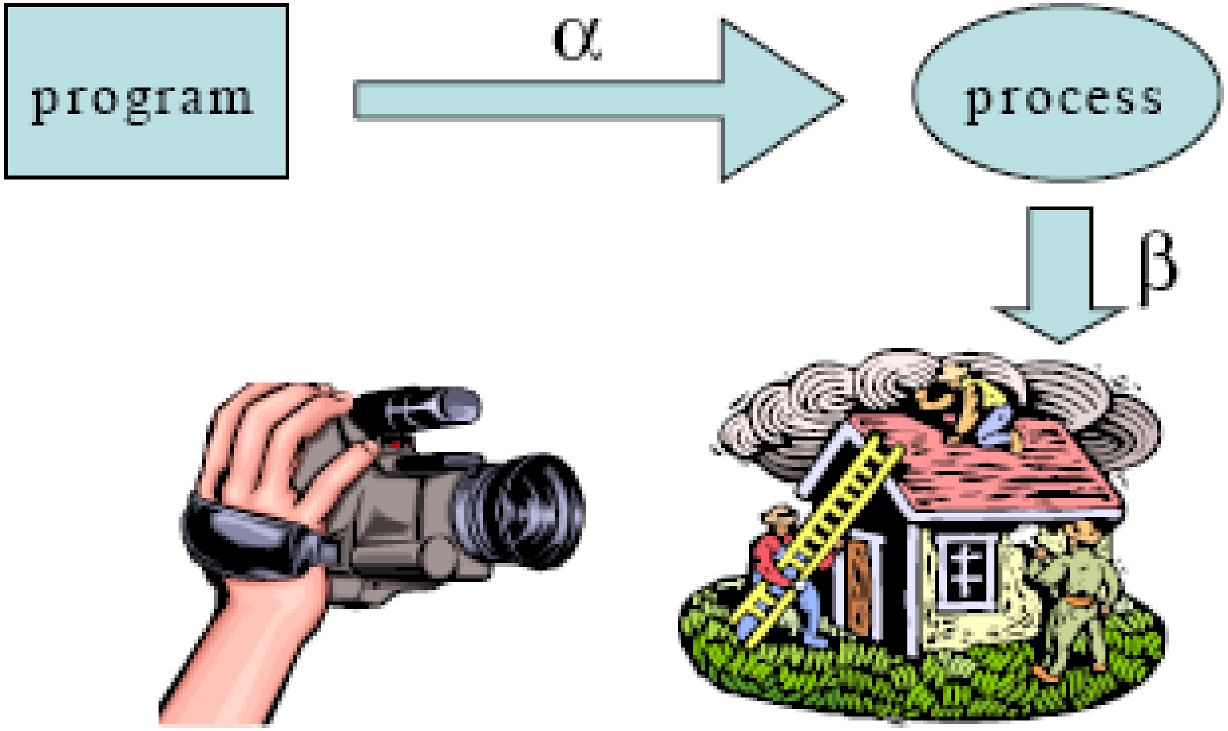
Linking LSD agents to ADM entities …

LSD agents'
state and derivate
observables



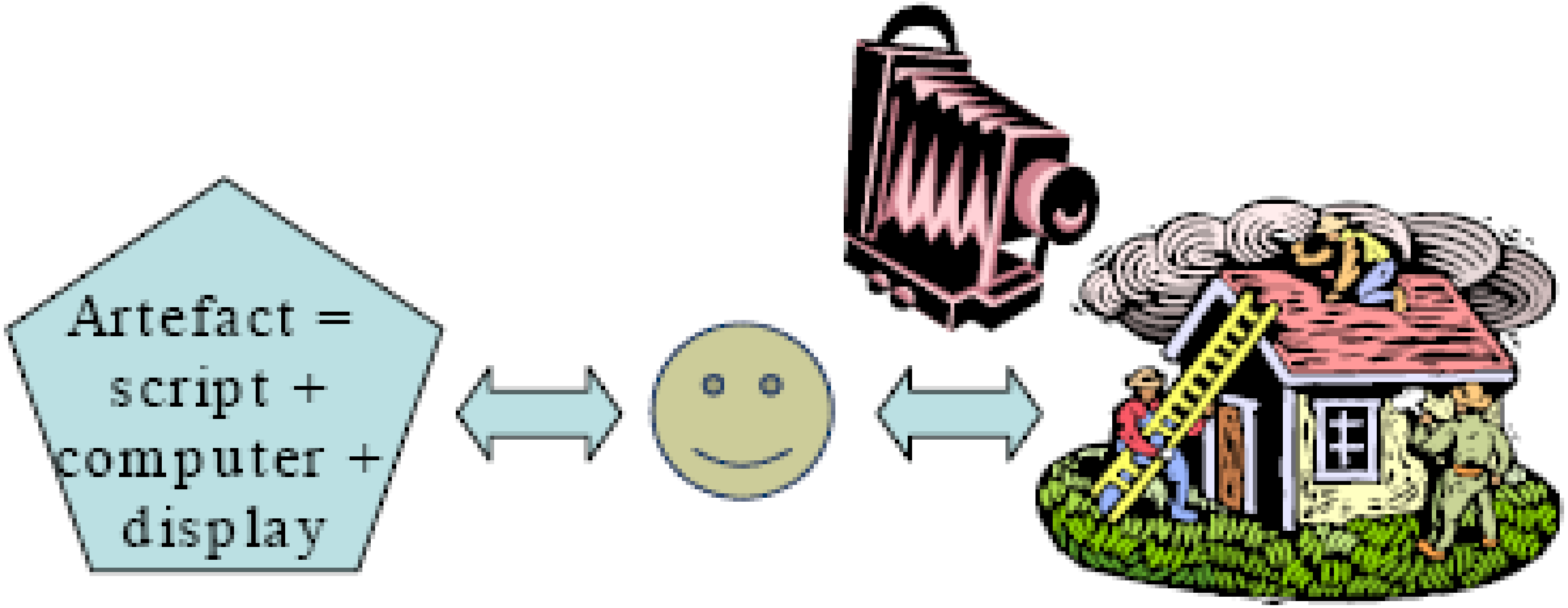
LSD agents'
protocols

Semantic Relations (I)



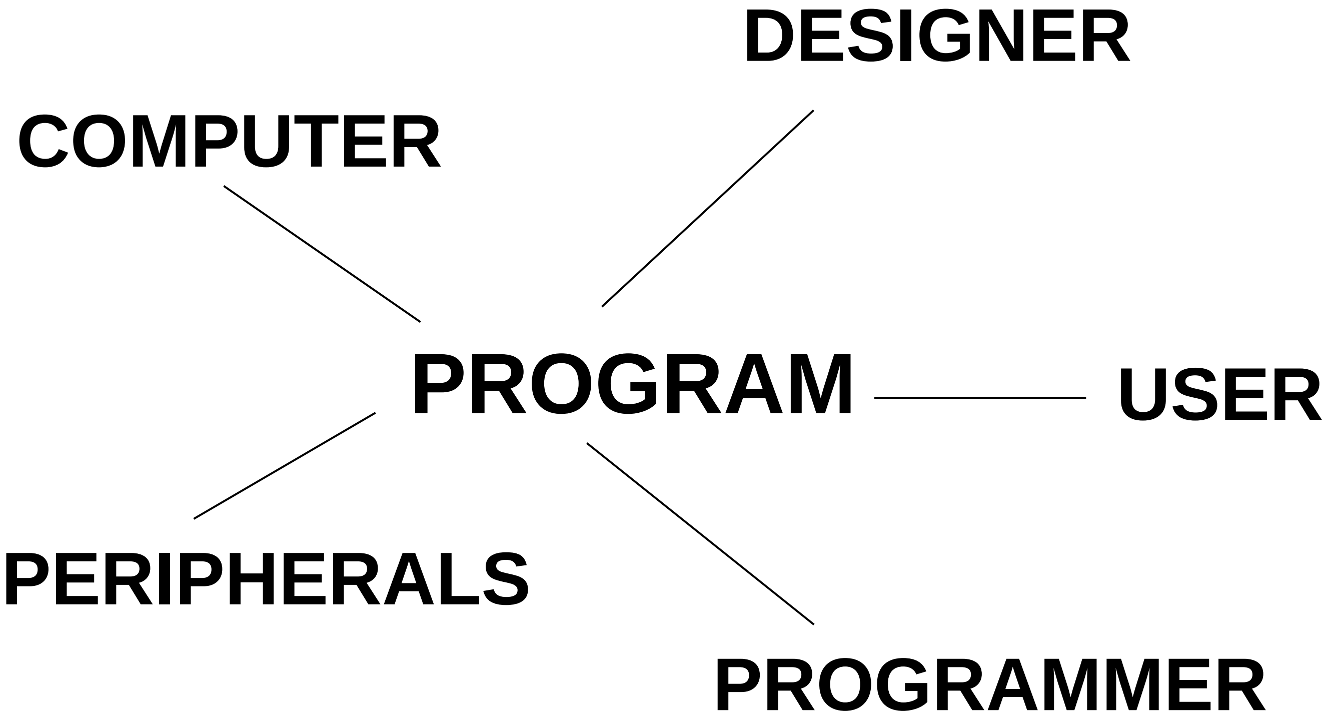
The semantics of a traditional program

Semantic Relations (II)



Artefact = script + computer + display

The semantics of a definitive program



DESIGNER

COMPUTER

PROGRAM

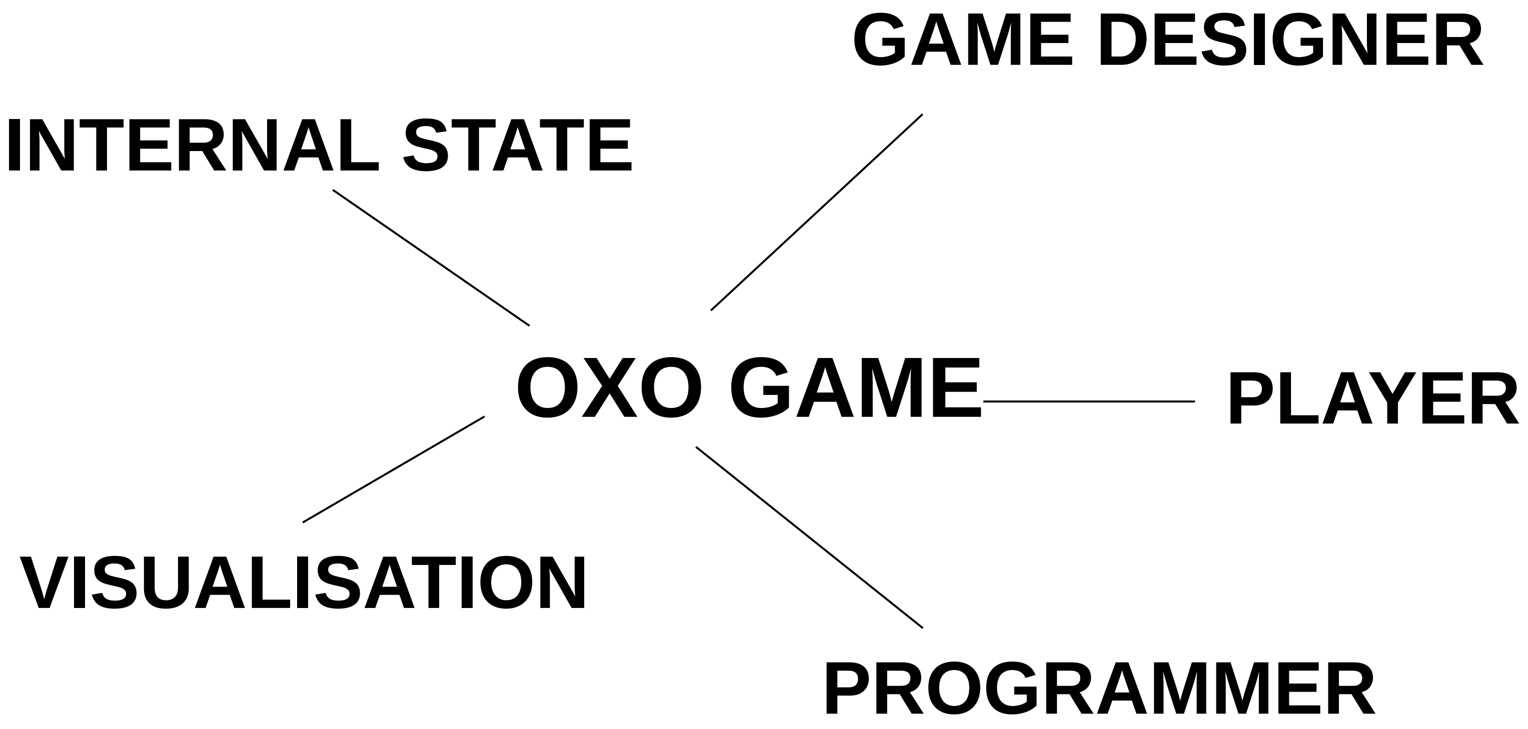
USER

PERIPHERALS

PROGRAMMER

Diverse relations / representations in a traditional program

… compare this with the OXO laboratory



GAME DESIGNER

INTERNAL STATE

OXO GAME

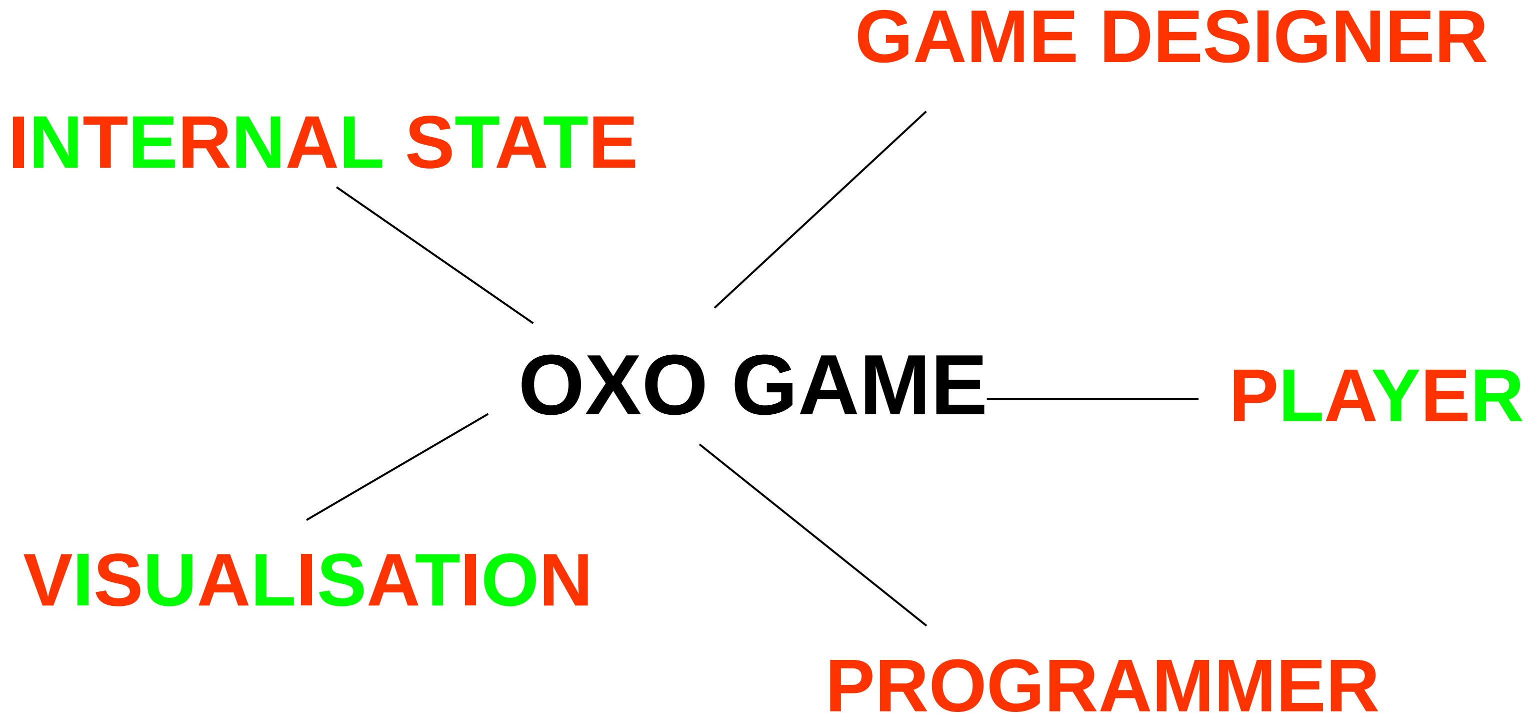
PLAYER

VISUALISATION

PROGRAMMER

… all relations mediated by definitions

… Behaviour as programmed state change



GAME DESIGNER

INTERNAL STATE

OXO GAME

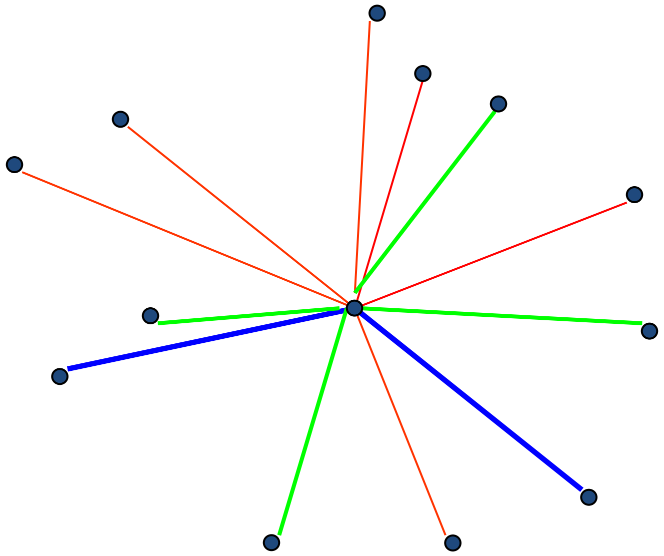
PLAYER

VISUALISATION

PROGRAMMER

Static and dynamic elements of state

Definitive scripts as “furry blobs”



● ≡ a definitive script

🟥 ≡ a nonsense redefinition

🟩 ≡ a plausible redefinition

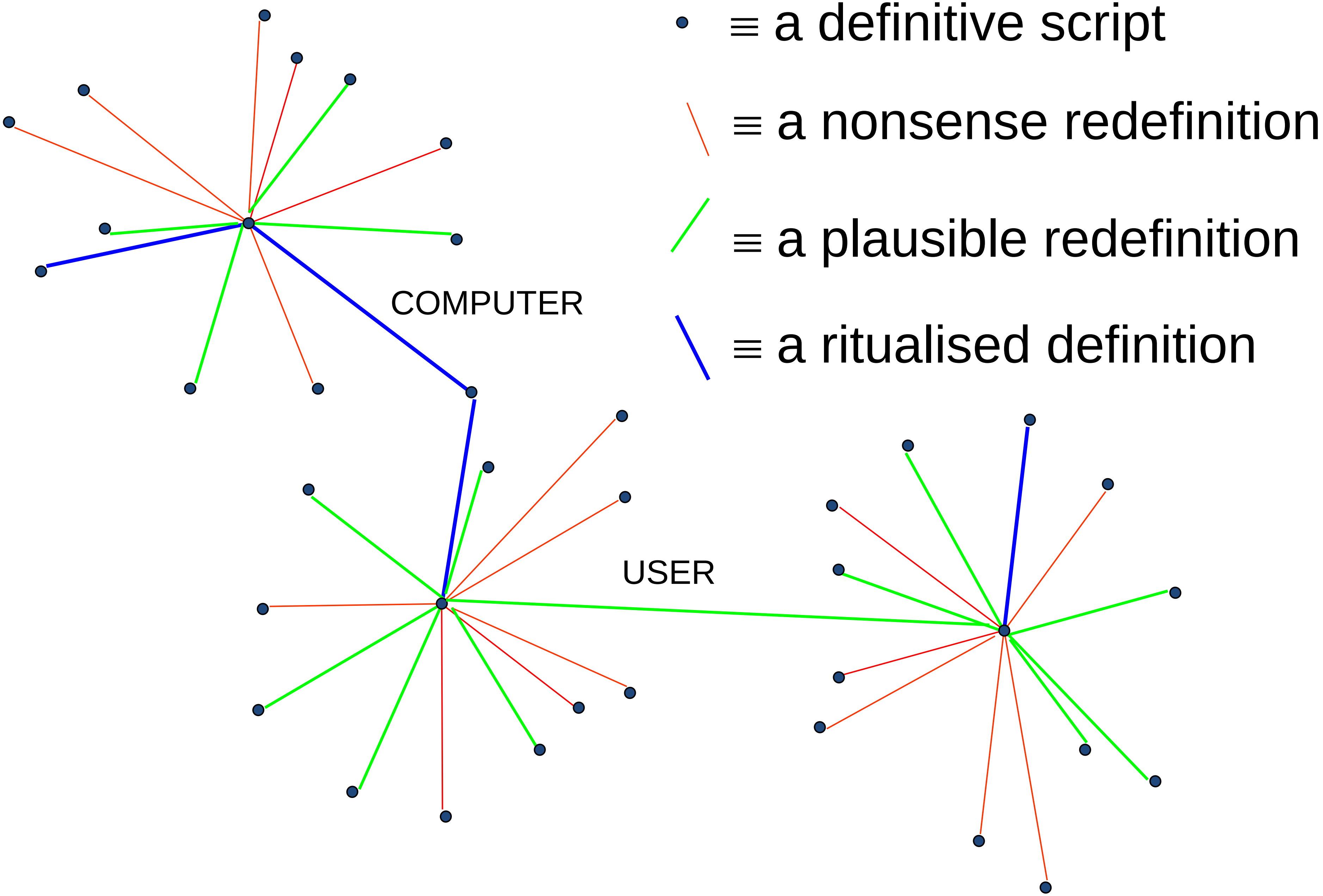
🟦 ≡ a ritualised definition

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



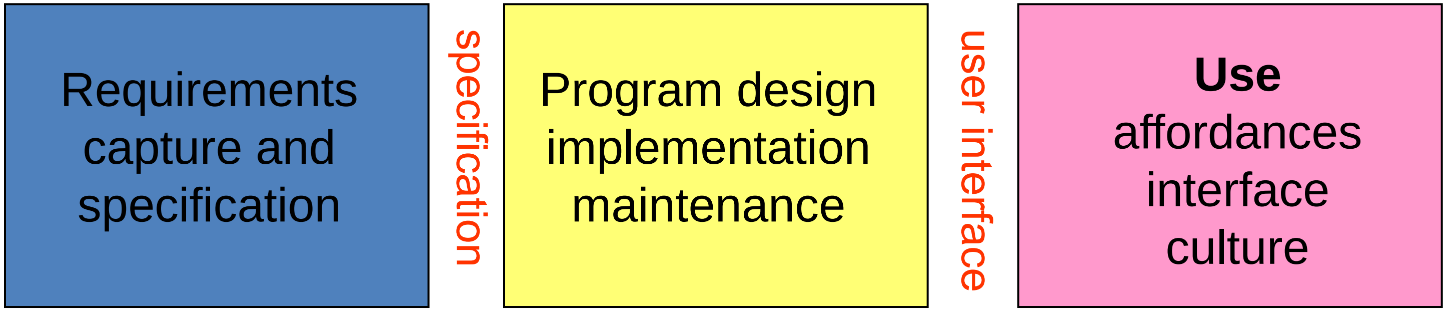
● ≡ a definitive script

🟥 ≡ a nonsense redefinition

🟩 ≡ a plausible redefinition

🟦 ≡ a ritualised definition

Traditional programming



Requirements capture and specification

specification

Program design implementation maintenance

user interface

Use affordances interface culture

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

constructing and programming the machine-like components

designing program by identifying objects and functions

technical interface development

e.g. writing Java code

human factors study

interface design

empirical studies of use

prototyping

e.g. goals, operators, methods (GOMS) evaluation

Empirical Modelling

Requirements
capture and
specification

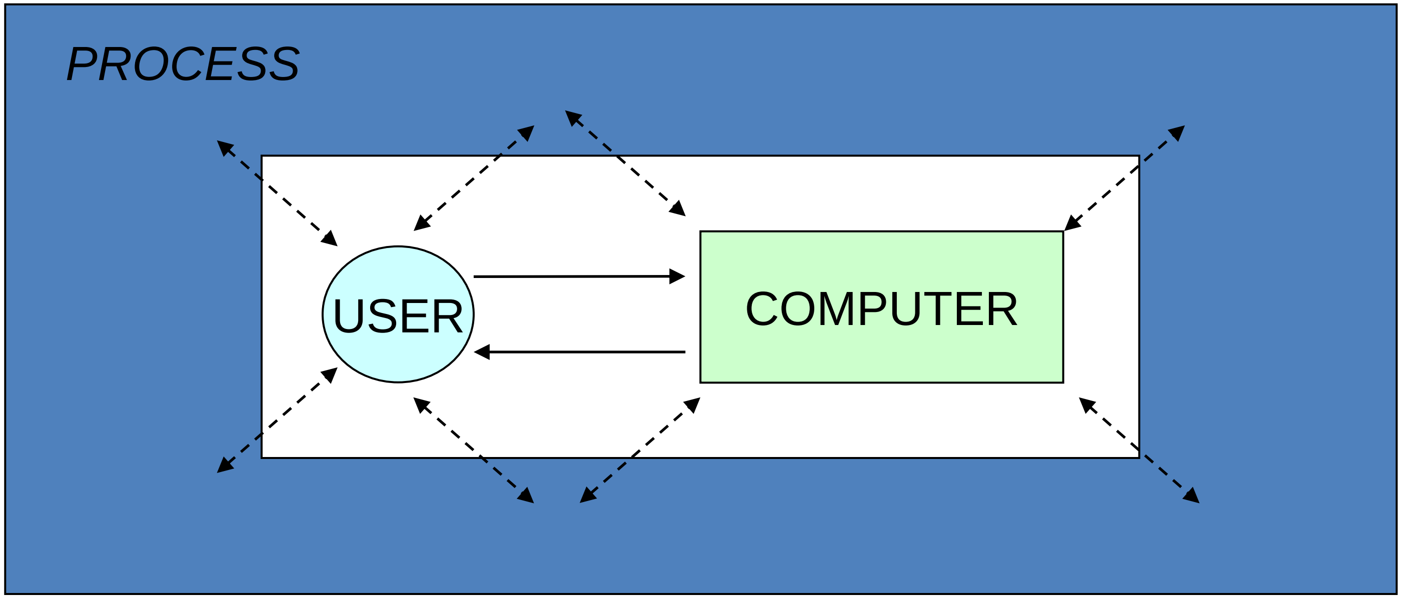
develop scripts
in isolation
as “furry blobs”
that represent
the observables
and dependencies
associated with
putative
machine-like
components
and
human interactions
and interpretations

Program design
implementation
maintenance

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

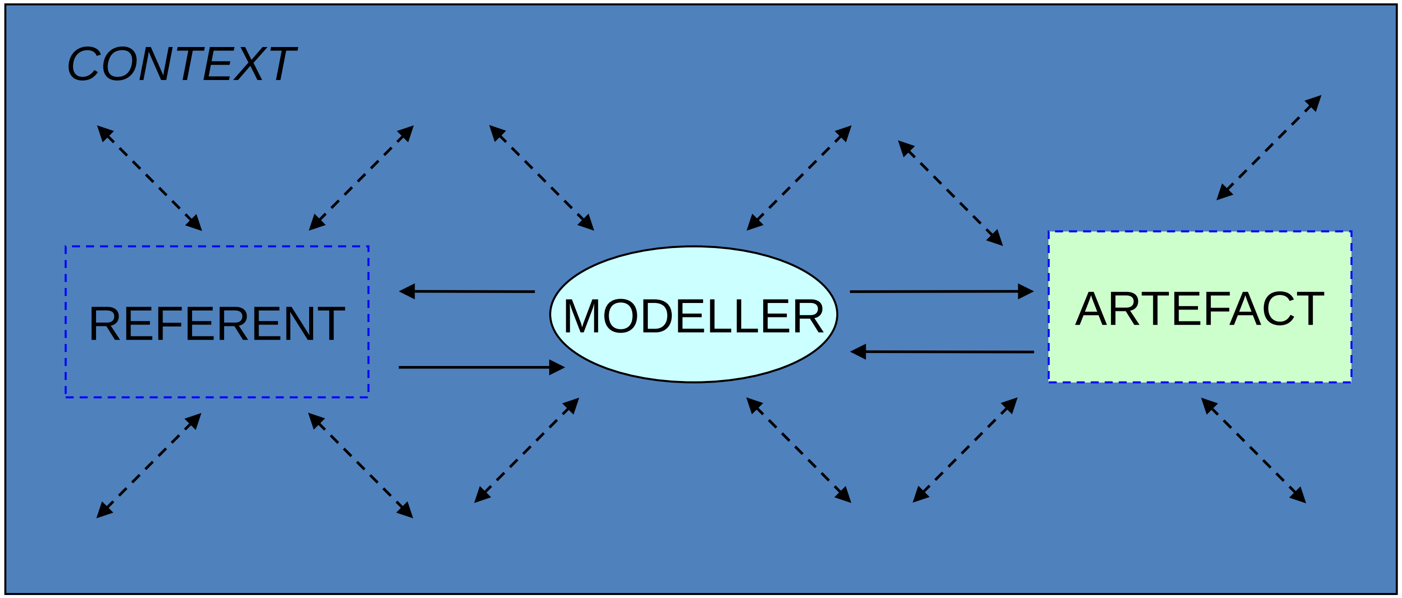
Use
affordances
interface
culture

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



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

Programs are understood in relation to processes in their surrounding environment

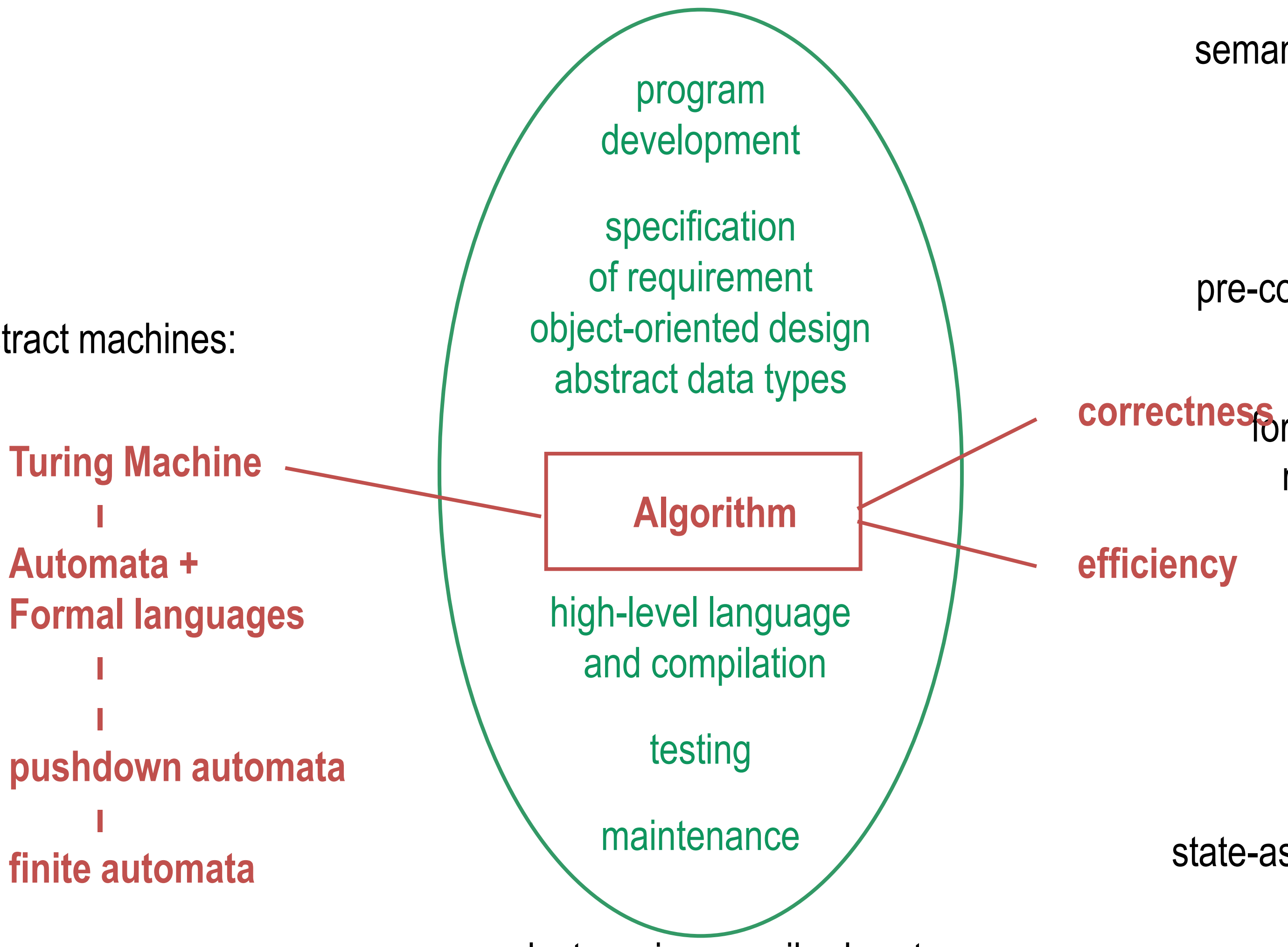


Artefacts and their referents as sculpted out of open interaction with the world

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

Focus of conventional Computer Science

computation = execution of algorithm (cf. mechanism + automation)



abstract machines:

Turing Machine

Automata +
Formal languages

pushdown automata

finite automata

program
development

specification
of requirement
object-oriented design
abstract data types

Algorithm

high-level language
and compilation

testing

maintenance

correctness

efficiency

product as circumscribed system

semantics and development
are machine-oriented

interaction shaped by
pre-conceived interpretation

formal and theory-driven
mathematics and logic

context is public,
committed and rigid

behaviour is primary
state-as-abstracted is derived

Empirical Modelling: a broader view of computing

computation = making sense of phenomena
and information processing (human computing)

observation and experiment

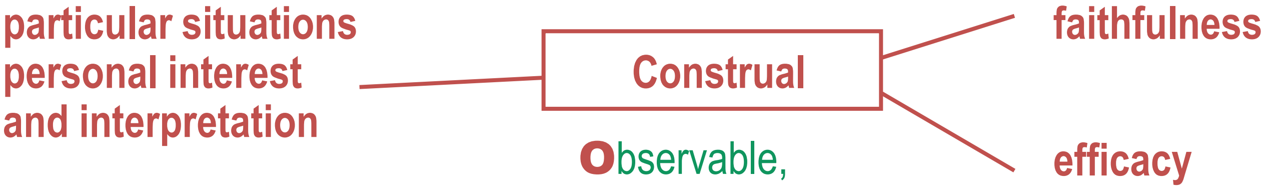
semantics and construction
are experience-oriented

Model or artefact
construction

personal engagement
with the world:

domain of interest
conflation of design,
development, use

interpretation shaped
by free interaction



particular situations
personal interest
and interpretation

Construal

faithfulness

efficacy

informal, intuitive, exploratory
imagination and memory

personal experience
and expression,
perception, observation,
dependency and agency,
sensory stimuli

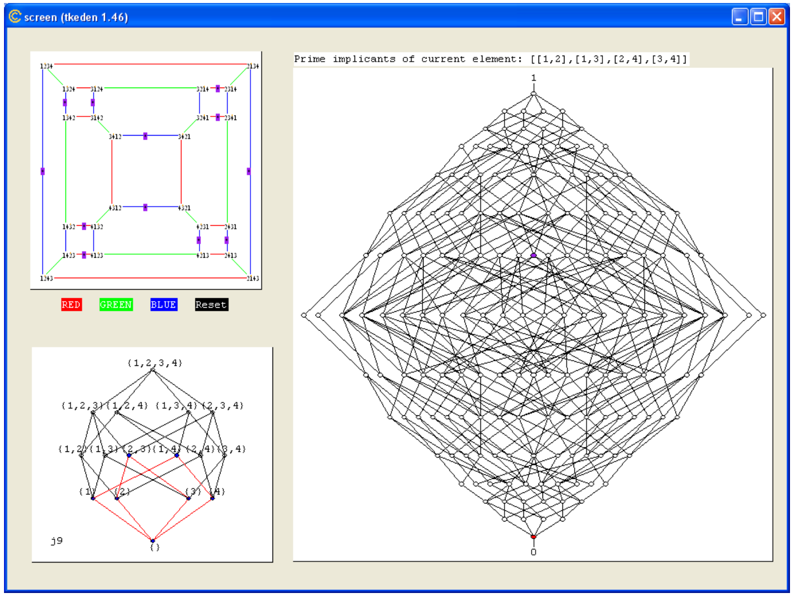
Observable,
Dependency,
Agency
in definitive scripts
in appropriate notations
driven by interaction

context is personal, provisional

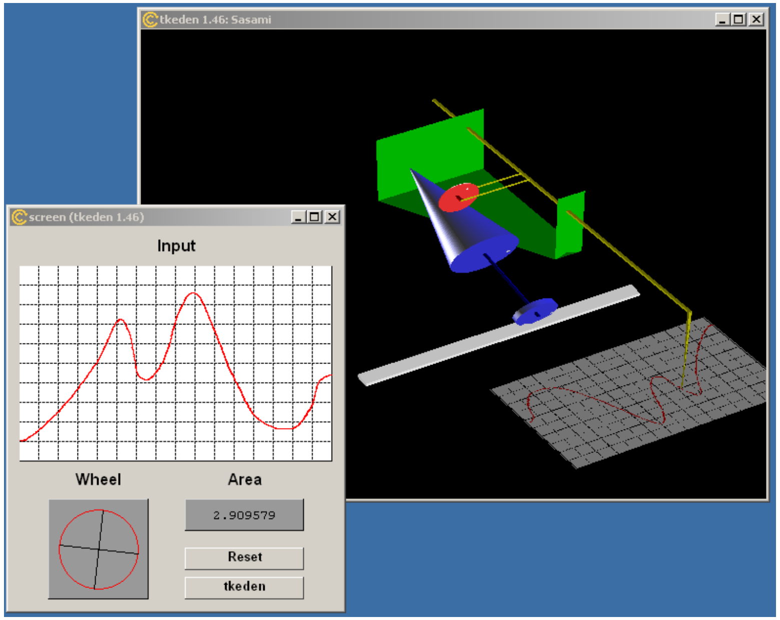
state-as-experienced is primary
behaviour is derived

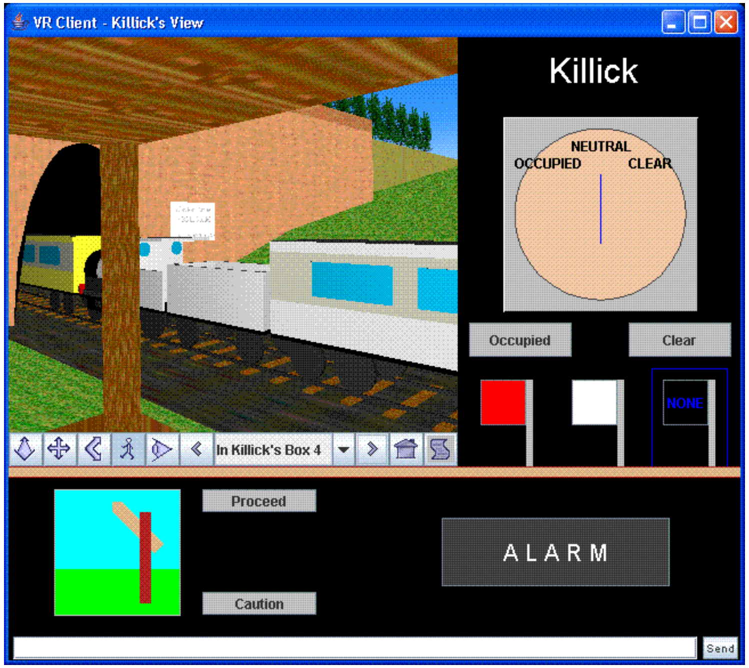
process in open environment

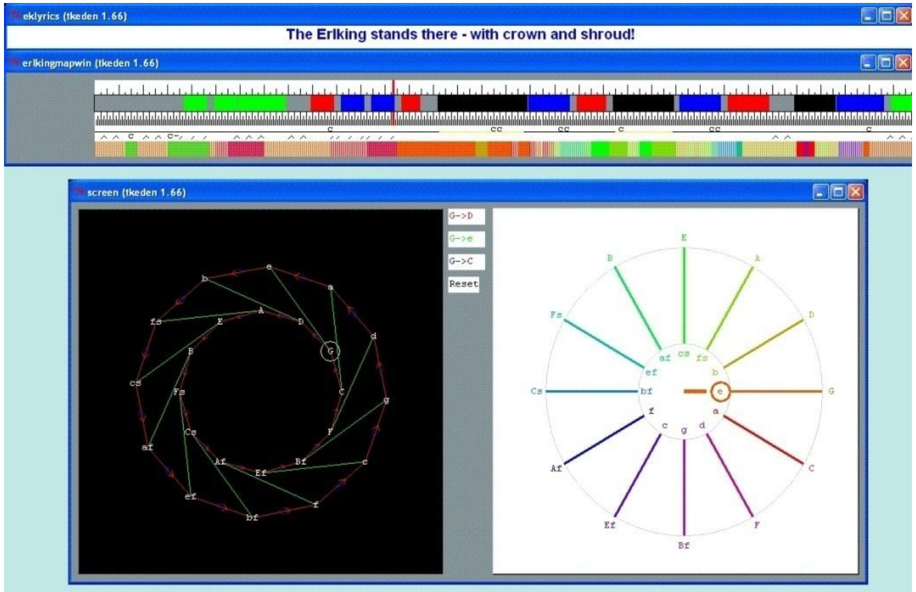
Sense-making in mathematics, in the physical world, social interactions and music ...



Prime implicants of current element: [[1,2],[1,3],[2,4],[3,4]]

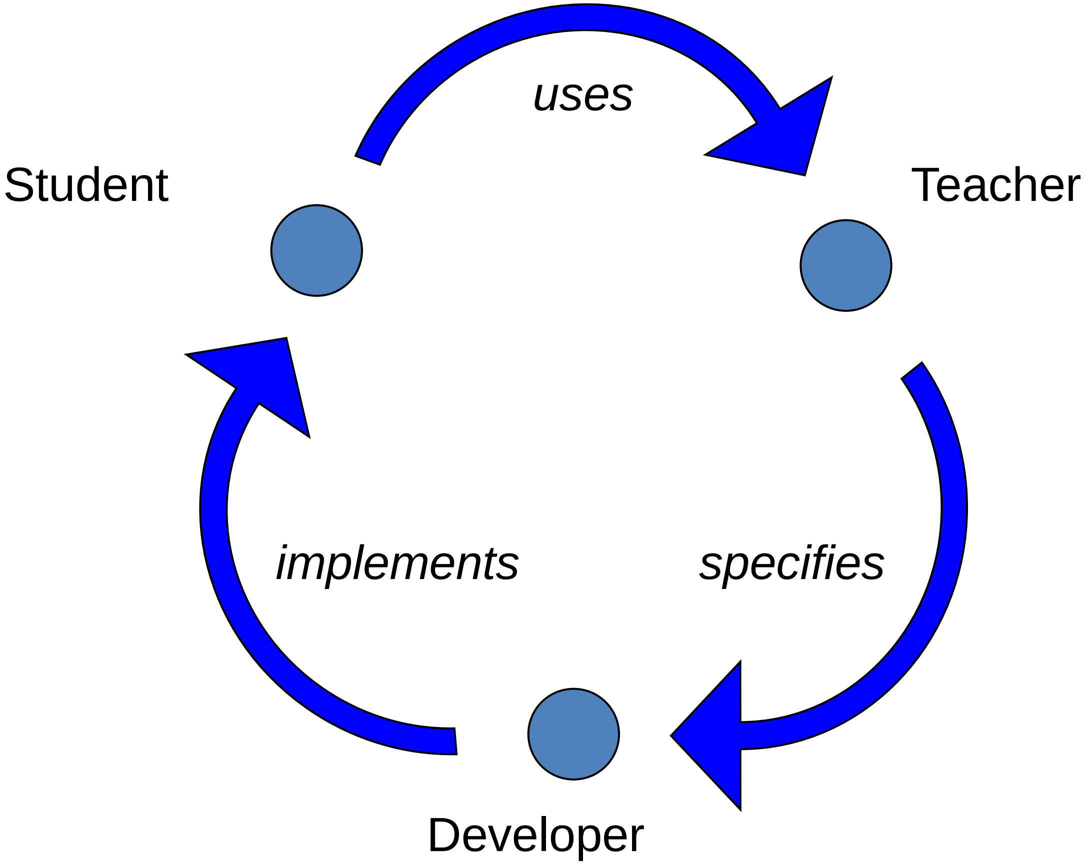




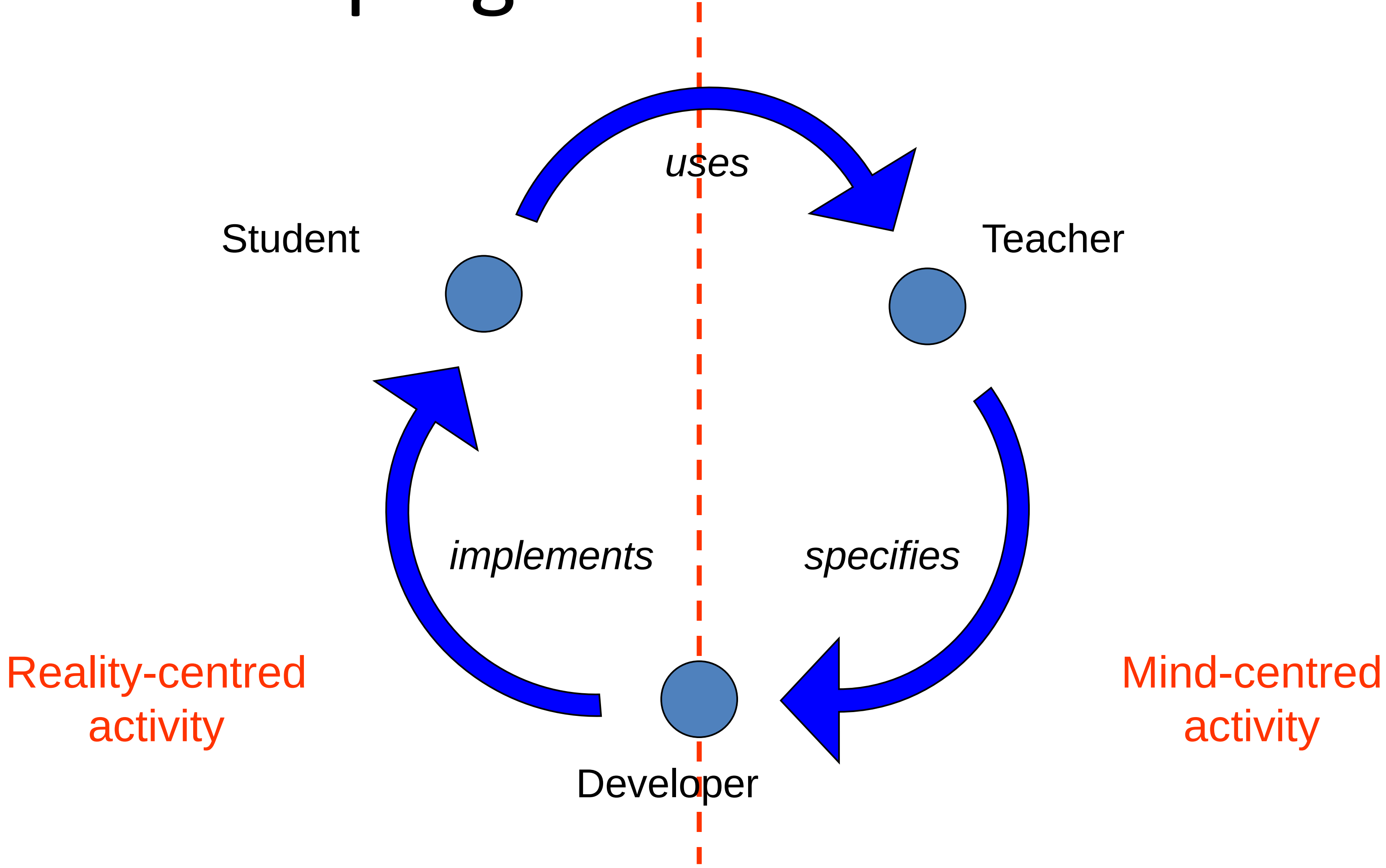


The Erlking stands there - with crown and shroud!

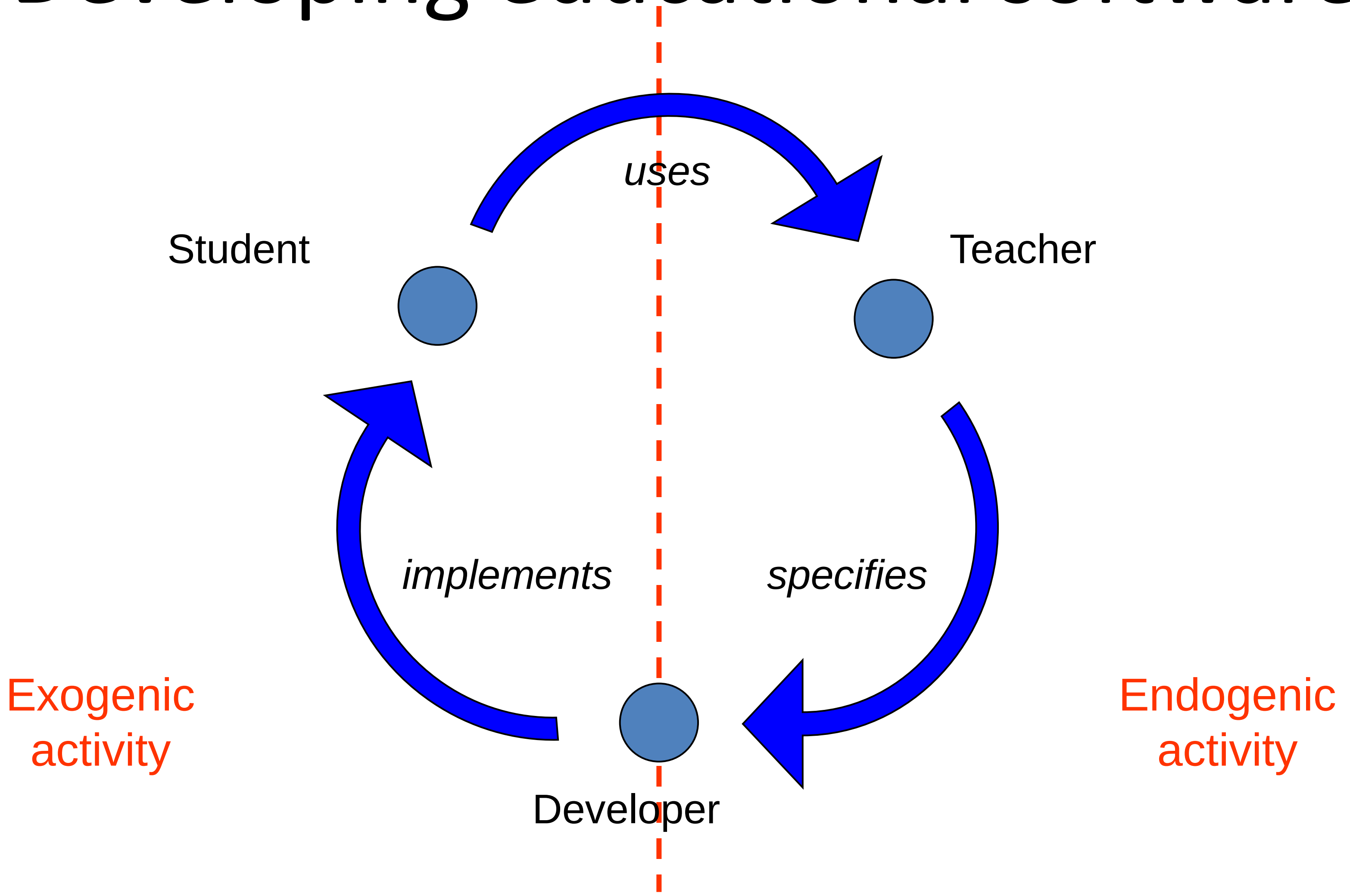
Developing educational software



Developing educational software



Developing educational software



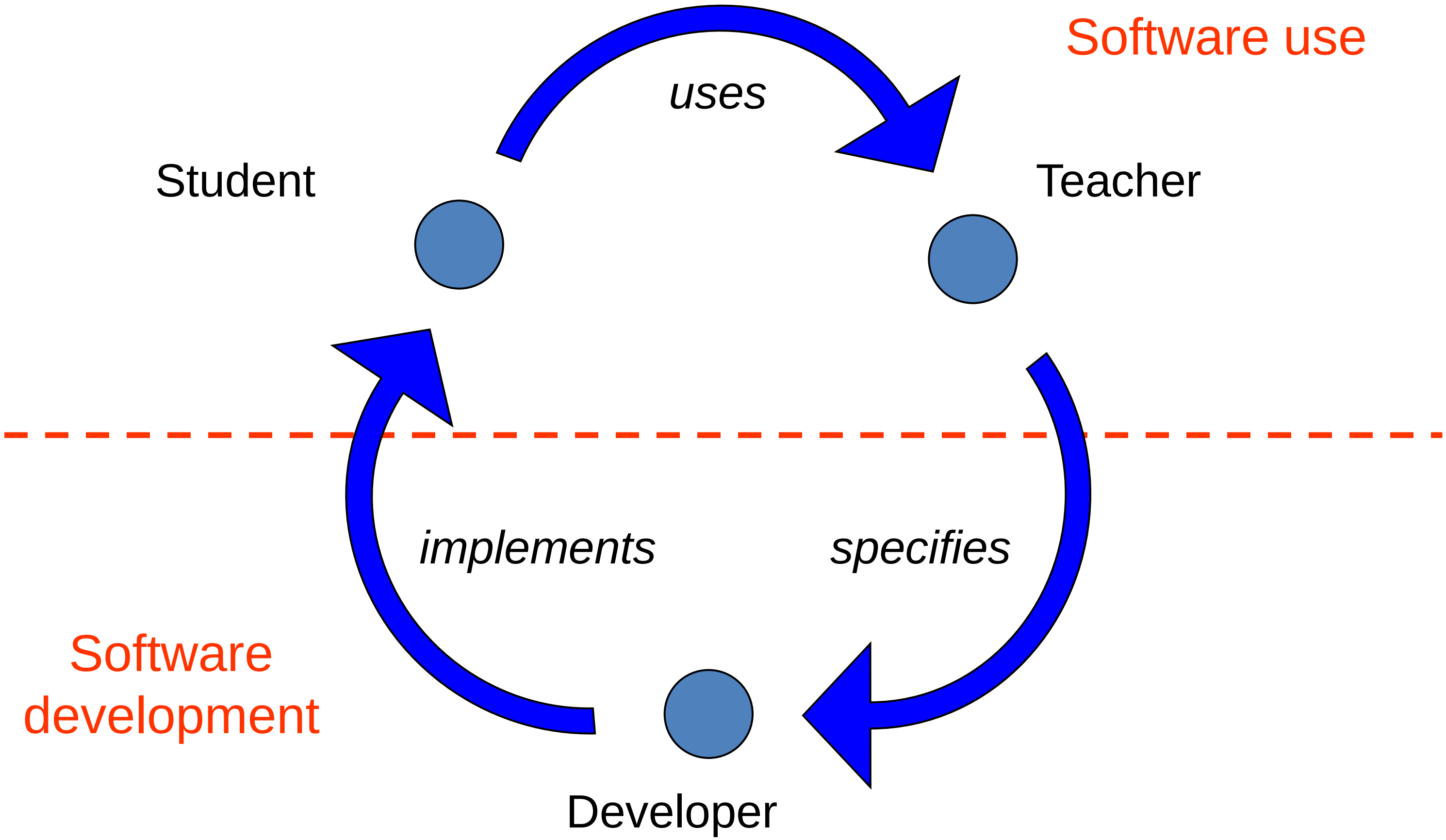
… subjectivity and objectivity are affairs not of what an experience is aboriginally made of, but of its classification.

William James: ERE p141

Exogenic activity

Endogenic activity

Developing educational software



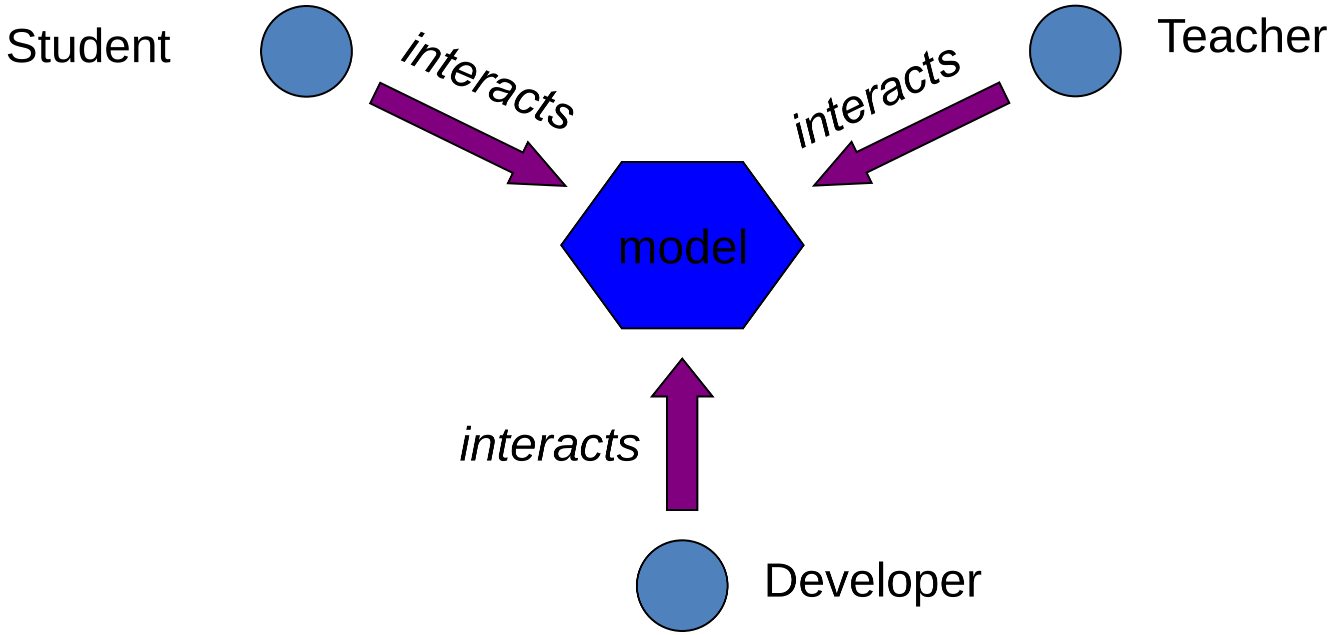
Perspectives of educational software

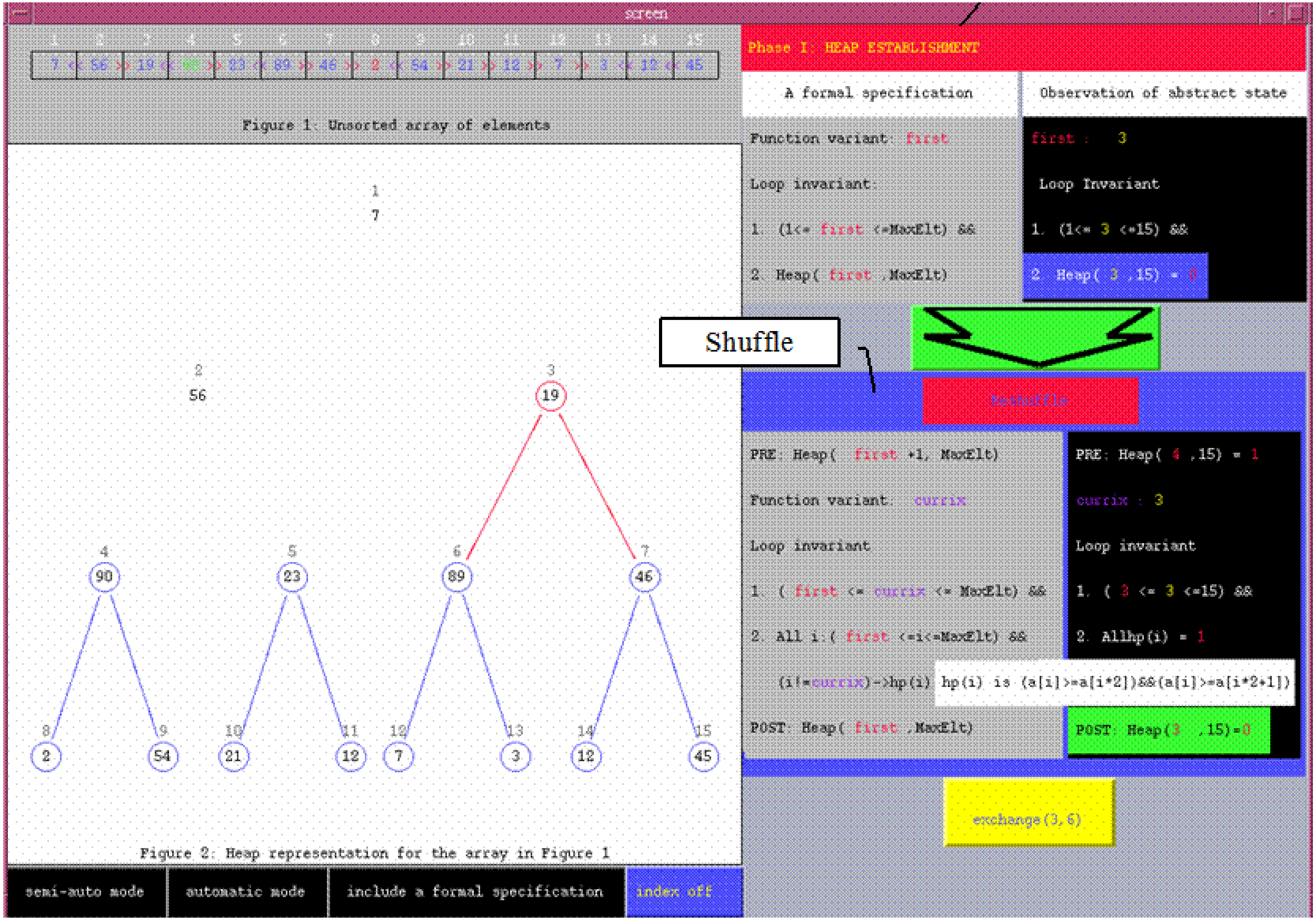
- Student vs teacher vs developer
- Mind-centred vs reality-centred
- Software development vs software use

➢ How can we bring together these different perspectives? Why?

Empirical Modelling (EM)

- Offers a set of principles for model building in any of the student, teacher and developer roles:





1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
7	56	19	90	23	89	46	2	54	21	12	7	3	12	45

Figure 1: Unsorted array of elements

Figure 2: Heap representation for the array in Figure 1

Phase I: HEAP ESTABLISHMENT

A formal specification	Observation of abstract state

```
Function variant: first

Loop invariant:

1. (1<= first <=MaxElt) &&

2. Heap( first ,MaxElt)
```

```
first :   3

Loop Invariant

1. (1<= 3 <=15) &&

2. Heap( 3 ,15) = 0
```

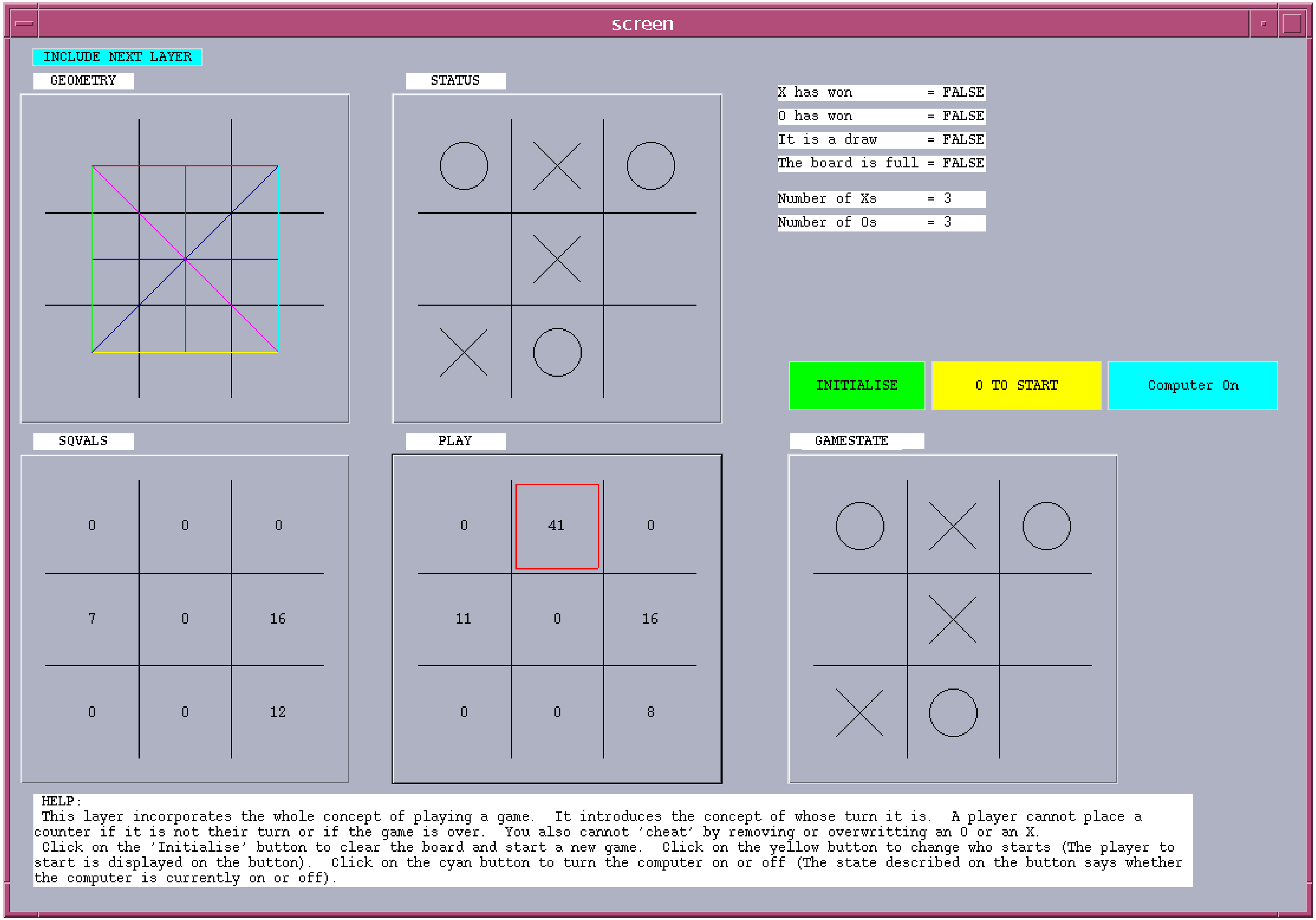
Shuffle

```
PRE: Heap(  first +1, MaxElt)

Function variant:  currix

Loop invariant

1. ( first <= currix <= MaxElt) &&

2. All i:( first <=i<=MaxElt) &&

   (i!=currix)->hp(i)

POST: Heap( first ,MaxElt)
```

```
PRE: Heap( 4 ,15) = 1

currix : 3

Loop invariant

1. ( 3 <= 3 <=15) &&

2. Allhp(i) = 1

POST: Heap(3  ,15)=0
```

hp(i) is (a[i]>=a[i*2])&&(a[i]>=a[i*2+1])

“Formal specification from an observation-oriented perspective”



```
X has won          = FALSE
O has won          = FALSE
It is a draw       = FALSE
The board is full  = FALSE

Number of Xs       = 3
Number of Os       = 3
```

STATUS

O	X	O
	X	
X	O	

SQVALS

0	0	0
7	0	16
0	0	12

PLAY

0	41	0
11	0	16
0	0	8

GAMESTATE

O	X	O
	X	
X	O	

HELP:
This layer incorporates the whole concept of playing a game. It introduces the concept of whose turn it is. A player cannot place a counter if it is not their turn or if the game is over. You also cannot 'cheat' by removing or overwritting an O or an X.
Click on the 'Initialise' button to clear the board and start a new game. Click on the yellow button to change who starts (The player to start is displayed on the button). Click on the cyan button to turn the computer on or off (The state described on the button says whether the computer is currently on or off).