# Programming as Human-Computer Interaction

**Meurig Beynon**

*Dept of Computer Science, University of Warwick, Coventry CV4 7AL*

## 1. *Motivating issues for programming research*

These notes concern a programme of research into a new programming paradigm that has been developed at Warwick over recent years. As explained in [1], our approach has been informed by fundamental concerns arising in a variety of different areas of CS:

programming paradigms — Backus

Is there a good general-purpose programming paradigm?
Can we resolve the tension between theoretical respectability and practical power and usefulness?
How should we approach parallel programming challenges?
What is a good programming paradigm for interaction?

software engineering — Harel, Brooks, Balzer

How do we move from requirements to module decomposition?
How do we deal with reactive systems development?
What is the role for formal specification in software design?
Can we generalise sequential specification techniques to concurrent systems?

AI and applications — McDermott, Cantwell Smith, Kent

How do we develop intelligible programs?
How do we relate programs to their real-world interpretation?
Can we develop an adequate theory of reference?
Can we overcome the problems of a logicist position?
Are there modes of programming suitable for the naive user?

Objects and the problems of generalisation e.g. penguins can't fly
Closed world assumption
Non-monotonic logic

foundations — Turing, Tarski, Scott-Strachey

Is Turing's framework for computation adequate for modern CS?
Are the concepts used to provide foundations for classical mathematics appropriate for CS?

## 2. *Perspective of the paper*

Take the view that current wisdom in programming practice and software engineering doesn't address the *essence* of the problem of developing software for modern applications (cf Brooks' No Silver Bullet).

Our central thesis is that the essence of programming is representing transformations of state in a way that assists human interpretation. The methods we exploit involve a far-reaching generalisation of a principle for human-computer interaction to be found in spreadsheets.

We do not argue that our approach will displace current methods and concepts, but believe it has the potential to integrate them in a way that is hard to envisage without a radical change of perspective.

In particular, can see the roles for techniques from e.g.
- functional programming
- object-oriented design
- formal specification

## 3. *Content of paper*

Personal view of how these issues are being and will be successfully addressed in our program of research into a new programming paradigm.

Little technical detail: sketchy overview of the main concepts

Consideration of some of the key distinctive features of our approach + identification of the future framework for programming they suggest

Disclaimer:

- spectrum of issues is wide
- practical realisation doesn't yet do justice to the concepts

Previous paper [1] introduced our research program with reference to:
- what are the problems we're trying to solve?
- where are the most useful and relevant insights to be found?

In this paper, complementary (more speculative) concern with
- what we've achieved and can achieve
- how far we can clarify a vision and a goal for the research
- conclusions we can draw about the future of programming

## 4. *Two interpretations of programming*

"programming in the narrow sense"

Computer programming
= prescribing a sequence of computer instructions to solve a computational problem

"programming in the broad sense"

Programming as process
= all that is involved in developing a complex software system to perform an appropriate function

Share Brooks' view that the essence of the programming problem is bound up with the process aspect.

## 5. *The essence of programming*

What have the two definitions in common?

programming = transformations of state + human interpretation

In the narrow sense

states are machine states
transformations are machine state changes
human interpretation is encoding input and decoding the output.

In the broad sense

states are concerned with all state-changing agents
transformations with states of these agents and their interactions

also concerned with
the state of the software design + design transformations

### 6. *Classical computer programming*

programming = transformations of machine state + human interpretation

transformations of state are **automatic** and **reliable**
execution of a Pascal program on a traditional computer

human interpretation is **off-line**
establish conventions for giving input and inferring output

Historically, programming culture has been dominated by mathematical theory of algorithms and "batch mode computation". In this framework, we are concerned with how to automate a transformation that can be interpreted as computing a particular input-output relationship.

Simple example: Turing machine computation to add two numbers.

### 7. *Computer Programming: States and tranformations*

In computer programming, conceive state in terms of the values recorded in the registers of the machine. For this reason, see state as an undesirable focus of attention.

traditional programming paradigms focus on describing **transformations** of state.

e.g. functional programming circumscribes the whole computation as a single transformation of state, and discourages any consideration of "intermediate computational state"

In practice, need to worry about intermediate execution e.g for debugging and interaction with the computer

In a procedural framework, there is another perspective on state in the programmer's mind: an abstract algorithm that involves state concepts closer to the application. In this view, can interpret the intermediate execution to some degree.

Note:

Use of assertions, invariants, multiple assignment, predicate transformation methods etc reinforce the model in the programmer's mind by stressing the need to describe interpretable states formally.

Even abstract algorithmic languages reflect the influence of traditional machine code (e.g. unnecessary sequentiality, superfluous encoding).

## 8. *Programming – state in design and execution*

Classical Computer Programming operates in a tightly constrained world

can rely upon the computer to carry out operations

we have well-established conventions for communicating data to and from the machine

because we're only concerned with sequential interactions, we don't have to worry about complex interference and synchronisation

... even in this context, led to consider states in more than one role

machine state, abstract state of algorithm execution *procedural*

state of the functional program definition *functional*

In practice (e.g. because we need to design a complicated procedural program, or invoke lazy evaluation in the specification of a functional program, have to consider all three). Observation also suggests - possibly misleading - that we can commute the state problem from execution state to design state. [Misleading because don't get a satisfactory view of the execution from a specification of a functional program whose design is transparent, and vice versa.] Even so, there's confirmation here that execution and design states do both matter in good programming practice. For instance, care about *efficiency* and *maintainability*.

## 9. *The role of state in software development*

In programming in the broad sense, the need for satisfactory state representations is even greater

state-changing agents are less reliable
modes of interaction more diverse, more problem specific
concurrency leads to problems of synchronisation and interference

["programming a reactive system entails machine design"]

The design / maintenance process is far more complex

problems include
- decomposing the program into modules
- propagating design changes between modules
- coping with incompleteness and inconsistency

taking account of the views of more participants in design process:
e.g. different kinds of user, conflicting design requirements

*10. The definitive script state model of spreadsheet*

state = agent view

represented by definitive script + protocol for redefinition

variables in the script correspond to observations (in the scientific sense) of the real-world system being represented

definitions express the way in which changes in these observations are synchronised in transition between states

Central abstraction: indivisible propagation of state-change

*11. Applications in the software development process*

1. representing the assumptions about interaction between state-changing agents in the application

   identify the observations needed to express how each agent

   can register changes of state in its environment through perceived (synchronised) changes in values

   is privileged to affect the state of other agents through changing parameters observed by the other agents

   cf the agent-oriented model of the VCCS: speed-transducer, vehicle dynamics, speedometer

2. representing the ways in which different design agents can interact with the current design

   e.g. privileges of the dashboard designer vs the engine designer
   refining the design (e.g. enhancing the speedometer)
   debugging the specification (e.g. relocating the backwheel)

3. representing aspects of the design process

   e.g. deconstruction of design
   design of speedometer in isolation
   modes of inspection of design
   design of screen layout / design of vehicle profile
   vs experimentation with simulation over time

   recovering / recording history / versions

## 12. *Observations and definitive variables*

Observation = scientific observation

Underlying the modelling strategy is the identification of a comprehensive set of observations. For the VCCS, this set is "what the designer(s) of the VCCS need(s) to be able to observe to investigate the model fully". This includes the observations associated with

appearance of the dashboard
profile of the vehicle
components required for visualisation of the vehicle performance
relevant attributes of the vehicle in motion (e.g. speed, forces acting)
values computed / communicated between internal components

This in particular accounts for how the engineer interprets the model

what are the forces acting?
how are the components behaving?
how is the cruise controller responding?

Also accounts for how we assess the role to be played by the driver

where are the control buttons?
how intelligible are the input and status conventions?

Intended to accord with the behaviour of an actual vehicle cruise controller such as might build. Privileges of the designer include refining the integrator that models analogue changes.

13. *Global state*

Comprehensive set of observations <--> global system state
The omniscient omnipotent perspective: can see all and change anything

State-based model to support programming BUT
doesn't represent state by accumulation of side-effects
have persistence of state

cf calculator vs spreadsheet, function key vs graph of function

Definitive script and agent emerge as fundamental concepts

Analysis of data dependencies in transformation determine the scripts

Independence of changes identifies agents
Moving the candle and its reflection is one and the same action: it pertains to a single agent
Moving the candle and turning the mirror are independent actions whose concurrent execution presumes two agents

"Analysis of observations leads us to the identification of agents"

14. *Significance as a Method of Requirements Analysis*

Requirements analysis can be oriented to solution of a specific task

Consequence may be that the specification is hard to change

This is the argument for *modelling the application* rather than addressing the specific task cf JSD and OOD.

Our approach is in the spirit of JSD BUT exploits the fact that the same set of observations can be the basis of an enormous variety of models

Reinforcing this idea:

same script + different agents gives a completely different model

fundamental mechanisms that relate observations much more easily re-used and have far greater generality in application than objects

cf lever *mechanism*
**used in** car, nutcracker, crane, signal *object*
**i.e.** devices for transport, demolition, lifting, communicating *function*

Practical evidence from rapid prototyping with extensive re-use
e.g. VCCS -> sailboat, billiards simulations

*15. Global state vs views*

Data dependencies are much richer than we can satisfactorily model with current tools cf *if ... then ... else* ... definitions. There are interactions where data dependency changes dynamically ...

... this + applications above indicate that we need more dynamic view-oriented methods of organising scripts

e.g. select a subset of definitions
compose two sets of definitions
substitute one set of definitions for another
introduce a specialisation of a script
replicate a piece of script
partially evaluate
generate script dynamically (e.g. change TopSpeed of vehicle)

Our present implementation methods are inadequate for such use.

*16. The act-of-faith*

Central to the use of agent-oriented modelling over definitive representations of state is a qualitative distinction between two kinds of interpretation of state. This is associated with the "Experiment Paradox".

The experiment paradox involves two perspectives on state changes

investigating what might happen *"requirements analysis"*
confirming what does happen *"preliminary design specification"*

Traditional activities associated with computer programming
e.g. logical specification and object-identication
are post-confirmation

Requirements analysis investigation is prior to the act-of-faith

Relevant issues:

modelling in a database not amenable to total automation
what we enter into the spreadsheet is not predictable

make presumptions about experimental context in science:
there aren't any other agents around / can't intervene

act-of-faith associated with clarification of the underlying assumptions
re reliability of state changes

17. *Concept of Interpretation differs according to side of the act of faith*

**interpretation** has two ingredients:
that which is referred to: "identifying the referents"
presumptions about the relation between referents

Variables in a mathematical theory
give no guidance about what are the referents *"abstraction"*
precisely prescribe how the referents relate

Variables in a definitive script
can be readily associated with their referents
"state-based correspondence with observation"
can stand in relationships that are not preconceived

Can confirm the interpretations of variables in a spreadsheet by interacting with them: does the relationship between them in change of state conform to what is observed in experiment?

cf isn't appropriate to say of the VCCS as *absolute logical assertions*: "the length of the vehicle **is** 30" or even "the length of the vehicle **is** positive"

There is a role for such interpretation in "AI" programming:

e.g. In modelling legal transactions can't abstract the machine operations -
the significance of an action is entirely different according to context

cf computer breaks down at a particular point in a transaction, what is my balance? perhaps (probably a hairy point of law) even the activities within the machine have external significance

lawyer has fatal heart attack just as enters last detail into computer, computer crashes before transaction is processed ...

## 18. *Passing from Experiment to Theory*

requirements -> specification
experiment -> theory
agent-oriented observation-based framework
"this is what I observe to be the case"
-> programming framework
"this is what I confidently believe is reliably true"

Unlike representing requirements in conventional idiom because

incomplete observation and partial objects, particular constraints

are qualitatively different. In particular, identifying an object or a constraint is more than incomplete observation can justify ... perhaps you haven't observed that this object has other methods, or its actions entail other consequences, or that the constraint can be violated.

Difference borne out by the relative difficulty of *changing the specification* in the two approaches

cf incrementally refining observations, introducing new agents vs revising theories, modifying predicates, hacking objects

## 19. *Fundamental Principles underlying Model-Building*

A most important aspect of experiment is what agents we presume

e.g. do we believe in miracles?

**What could have switched on the light?**
A person, an animal, a bird: not a woodlouse, not the wind

| |
|---|
| Presume that nothing changes except through action of an agent |
| Actions of agents only interact through common perception |

[cf whenever I brush my teeth my uncle in Australia blows his nose]

## 20. *Advantages over logical specification framework*

Important foci for (unsuccessful) research in logical setting are

**Problem of negation (closed world assumption)**

*"What isn't provably true is deemed false"*

In our framework, we don't presume observation complete at any point
In specific problem-solving, we can proceed as an experimental scientist might: "hang on, while I perform another experiment"

**Problem of Exceptional Objects (commitment in OOP)**

*"Penguins are birds but penguins don't fly."*

The semantics of our model can be defined by use. Our use of a penguin model can reflect the more scientifically accurate proposition that "a penguin is a bird that is never observed to fly". We can take up an agnostic position, and need not commit ourselves concerning the truth of the predicate "penguins can fly".

**Problem of Monotonicity of Logic**

*"a consequences of a set of statements is also a consequence of a superset"*

Treating observations as predicates leads to problems of non-monotonicity e.g. when we tie a brick onto a bird that can otherwise fly. A definitive script is not a set of assertions of incontrovertible truths. What's more, the introduction of a new agent has quite the opposite effect to the introduction of a new predicate: it liberates the model rather than constrains it: "There are more things in Heaven and Earth than are dreamt of in your philosophy".

*21. Role for Commitment*

Commitment can enter the picture in variety of ways

Can adjoin assumptions and compile to a conventional program
e.g. implementing the button user's VCCS

Can add monitors and constraints to help us to maintain invariants

The operators underlying definitions are presumed to have reliable interpretations

Expect that commitment increases as the model is developed
e.g. functionality of VCCS is strongly prejudiced towards particular modes of redefinition by this stage: *as if* we'd written a VCCS program

*22. Passing from Experiment to Theory is a universal process*

Have discussed applications to

Design Representation

Program Debugging

Program Construction

Other examples of usefulness

Proof presentation
leading the reader through a sequnce of states

Complex Task Management
assembling large volumes of inexact and incomplete data
translating between data formats for incompatible tools

## 23. *A New Programming Paradigm*

states in programming

states in the construction of a complex function

states associated with assertions (WP):
procedural element magnified by consideration of invariants

states as manifestations of agent views
data structure: different concurrent agent views
parsing as involving hierarchical agent activity

... parallels with debugging mode of development ....

modes of composition of state

Harel's statecharts: orthogonality and depth

... parallels with programming constructs ...
loci, iterations, parallel composition, arrays

## 24. *Reference*

.. wherein many other references may be found

1. W. M. Beynon *New Paths for Programming in Theory and Practice* University of Warwick, September 1992