David Mellor

Agent-oriented vs OOA for Concurrent Systems Simulation 30%
Buoys at Sea 70%

Insight into principles

in modelling there is no design in the normal sense

modelling can take little advantage of a method presenting a coherent view of the global functionality of the system since we simply cannot implement it, so the apparent lack of control structure causes us no problems ... distributing control down to individual objects is in fact a definite advantage.

appreciation of role of definitive principles for state representation:
abstracts synchronous atomic update from higher level procedural actions that fundamentally alter the state
run-time support

fan-out and fan-in

actions / time-based mechanisms / delays

balance of which tasks sit at the synchronous and which at the asynchronous levels is a powerful programming tool.

cope with continuous change and expansion .... asymmetric coupling of sections of code together ... ideal for the user-interface

Contribution to critique

- Problems for Structural Integrity with agent-oriented approach

agent-orientation gives up much of the advantage of encapsulated data, as it operates on global state, so we have neither convenient control nor data structuring

global state change – generally regarded as terrible
can't see the flow of data, can't track dependencies that we need to control changes to code

no protection for changing the wrong agent

too much scope for different kinds of communication

OO amenable to re-use: agent orientation stronger coupling
=> re-use requires alterations to the code

- Specification Structure and Implementation Relationships

2 level LSD + EDEN representation
source linked weakly to the specification, change less controlled than in OOD – uncontrolled change when used lazily

Problems of Definitive Representation

[current] environment tends to rely far too much on run time evaluation to determine errors in the model ... bugs able to slip through in the form of code that was never executed.

One pass disallows forward state reference => agent code split up

Uni-directional state references make the coupling of agents very strong ... difficult to detect the readers and writers of an agent that would have to be changed if any fundamental changes were made to it.

Use of software tools (Scout / DoNaLD / EDEN)

Very ambitious implementation in eden and donald. Illustrates range of communication mechanisms. Also includes eden code for scenarios. Heavy use of donald rather than working at eden level.

Use of specification techniques

Very full and explicit LSD specification. Some interesting and novel aspects to the specification: extension of protocol specification to include sequences reflecting a scenario.

Comments

Essay

This is an outstanding essay that has taught me more about agent-oriented modelling over definitive representations of state than all the referee's reports I've read in 10 years. I hope that you can prepare a publication from it, perhaps in cooperation with me, and look forward to discussing your insights and observations further. I would also like to keep a copy of the essay for reference.

Buoys at Sea

A really interesting example that is well worth exploring. You've done a great job in constructing this piece of practical work, and it all adds up to a most valuable contribution to understanding better what has to be done within the agent-oriented development framework.

Nicholas Pownall

Naive Physics & its Application to Cricket Simulation 30%
Cricket Simulation using ADM 70%

Insight into principles

Not too sensitive to the CS perspective: very much a physicist's outlook

Don't know that the perception / action model is appreciated properly
e.g. batsmen always attempt one and only one run is global control / behaviour aspect, fielder protocol is throw as soon as own the ball. But doesn't fielder perform other actions, such as choosing to run, in response to other stimuli? who determines where throw goes etc?

"possible actions of players are determined by the actions of others and players must interact to keep the game going"

Contribution to critique

Provokes consideration of how far physical laws explain anything (cf Pylyshyn's discussion of person making an emergency call). Obliquely brings out the limited physical content in the simulation problem.

Use of software tools (Scout / DoNaLD / EDEN / ADM)

Brave to take on the ADM, but some convoluted code?
Surely termination of throw() is inappropriately modelled as action of the apparently already defunct agent!
Don't like global variable fudge requiring two entities

Credit for breaking new ground here ...

Use of specification techniques

Use of LSD not well-developed. Has included intergration-over-time mechanisms in the perception / action framework. Looks very much as if development has been programming exercise in ADM. Semantics of pure function not suggested by find_position ... names.

Actual specifications of batsman and umpire suggest more interest.

Notes for examiner

Naive Physics => confluences / qualitative differential equations
envisionment
confluences not sufficiently quantitative for a useful approximate physical model to be developed

So favours envisionment over confluences ((cf urchincricket.e?)

very little physics contained within the simulation

cf Pylyshyn's Computation and Cognition

Comments

I much appreciate your taking up the cricket simulation challenge, and look forward to developing it further. I feel that you're at a bit of a disadvantage at present in not having had much experience of programming projects / issues in a CS culture and we shall need to work on that to bring out the best of the specification – e.g. to improve the link to the LSD

specification and iron out bugs in execution. We should aim for a departmental report on this, and hope to develop this to a full publication later.

The strength of this work is in the practical program development, and there may be some issues we should raise with Simon re ADM / eden on his return. The best LSD specifications seem to be buried in the ADM text, and the actual LSD extract you give isn't the best you could have chosen.

Issues: to what extent did you do *bona fide* LSD specification prior to implementation?!

I would like to have seen more discussion of what's going on at a higher-level than use of ADM, and more consideration of where – if anywhere – advantage can be gained from using a proper physical model to develop the simulation.

You've left comparison with naive physics entirely implicit e.g. in passing you indicate the importance of a precise state model, but don't consider for example how urchincricket.e relates to envisionment approach.

Paul Ness

Definitive Methods vs OOP for Software Development **40%**
Sail Boat Simulation **60%**

Insight into principles

observations vs objects

object is observed "in one way"
only one context for observation: static determined when the object interface is developed

design of an object relies on the designer knowing what an object does ... with definitive design no such prior knowledge is assumed.

scenario-oriented approach is full-life cycle development method

[need for] close correspondence between software architecture and the usage scenario (cf IB's "requirements and testing" theme)

Obscure / interesting references to: [declarative nature of the definitive variable] ... data dependency is only between a variable and its *uses* (?)

Contribution to critique

Essay is well-structured as a comparison OO vs definitive method

Use of software tools (Scout / DoNaLD / EDEN)

This is an excellent piece of software development, similar in quality to the VCCS

Use of specification techniques

Shows very good appreciation of how to use LSD and how to generate a dynamic model from this ...

Comments

A really fine piece of written and practical work. Your sensitivity to a major theme in the course: observations vs objects, is strongly evident throughout. You /we should work on a departmental report out of this I feel.

Particularly intrigued by your choice of practical example – look forward to looking into this in more detail.

Spelling slips: JSD = Jackson **System** Development **Beynon!**

Ian Tan

Definitive Methods vs State Charts for Requirements Spec 50%
Binary/Decimal Counter 50%

Insight into principles

Very many misconceptions evident throughout ...

Definitive languages are similar to declarative languages in that it is also a formal method and it is described in definitions (equations and relations) ??

"definitive notation as formal language" misconstrued?

not clear about the status of redefinition of array elements

Definitive notation is similar to Z notation but with one main difference; Z notation is based on logic and set theory whilst definitive is based on definitions. (!)

Definitive notations are event-driven ...

not appreciating essence of approach! e.g. conclusion
~> programming languages based on icons

Contribution to critique

Too tenuous a grasp of the ideas to contribute here

Use of software tools (Scout / DoNaLD / EDEN)

Relatively unambitious but well-executed example

Use of specification techniques

Follows the digital watch style model ... generally tidy, but simple

definition of protocol poorly layed out? – previous_clk = clk

(Is this part of the function of the component at all? – issue re whether the electronic component stores history. Otherwise more appropriate to handle history in a different way – e.g. within clock() or with reference to clk' to be interpreted in analog variable manner.)

Notes for examiner

State chart analysis is very inaccurate
interrupts at high-level ?? - wrong semantics?
do have parallel composition in state charts
don't nec have exponential blow-up
Has misread Harel's paper in this respect .....

Animation does exist ... have Statemate prototyping tools ... fails to appreciate the issues re iconic languages

Does Harel really describe State Charts as inherently "flat" and so unsuitable for top-down and bottom-up design? ... I think you mean state diagrams ... likewise "inherently sequential in nature etc etc"

Comments

I felt your hardware simulation example worked out well, though it could perhaps have gone further. There is some difficulty in modelling engineering components in terms of perceptions and responses, as it doesn't bring out the richest aspects of LSD specification.

I had reservations about your written work that may stem from what I see as some confusion on your part about what agent-oriented modelling etc is aiming at. Definitive notation is a formal language in the same sense that Pascal is a formal language (ie it has a well-defined syntax), but you seem to equate our approach too closely with use of formal methods than I think is appropriate.

I also felt that you hadn't been careful enough in finding out about statecharts, and failed to appreciate the distinction between state charts and more conventional state-transition diagrams that Harel criticises in his On Visual Formalisms paper.

Andrew Satterthwaite

A Comparison of Z and LSD for Software Specification 70%
LSD specification for a simplified tennis game 30%

Insight into principles

Appreciates the role of LSD in specification quite well

Identifies formal methods problems as
1) barrier of language, ease of development from spec
2) fallibility of specifier, despite formal notation
3) customer can't appreciate what the Z-spec says, so developing and modifying the specification can't so easily involve customer

Main idea: LSD overcomes these points
easy to read and interpret
more likely to spot mistakes
easier to adapt, communicate with customer, develop incrementally

Contribution to critique

Dubious about whether we are comparing like with like when we get to the comparison as methods of specifying a fixed requirement.

Use of software tools (Scout / DoNaLD / EDEN)

None undertaken

Use of specification techniques

OK as far as it goes, but doesn't address the perception dimension on the behaviour of agents

Notes for the examiner

Good insight into place of Z and LSD in conventional approaches to software spec

Does your diagram reflect statement that "iterative refinement goes through as far as actual system implementation"?

Comments

Thanks for writing this abstract ... it is a useful paper because it makes some important points without going deeply into the metaphysics of software development. It's the sort of paper that would appeal to an industrial audience perhaps, because it doesn't demand too great a change of perspective ... formal methods vs prototyping is mainstream thinking ...

... it's not a paper I would have written myself (also a good thing!), and I would be concerned about the perspective it suggests, especially where the presumption that like is being compared with like is concerned. Though I don't wish to invent spurious distinctions, it is just as important in solving our software development problems to establish fundamental differences between approaches and I think that we can't do justice to the Z vs LSD discussion without exposing some fairly deep conflicts in outlook.

In my view, we should think of LSD + something corresponding to the "act of faith" as appropriate for comparison with Z; look forward to debating / exploring this issue further in your project!

To what extent did your decision not to venture implementation reflect your view of LSD? It seems to me that the rather impersonal agents in the simple tennis game are already a step – away from the real focus of LSD – in the direction of anonymous computation.

Nam Sang Benny Lam

Dynamic System Modelling by Agent-oriented Prog Paradigm
Agent-oriented Programming : 2 illustrative examples

Insight into principles

"Complex software systems are, moreover, *things* that *act* that *move* and that *work.*"
Brooks
... agent is the concept of the thing that acts that moves that works.

Conceptual Encapsulation (= circumscription)
Guttag: "unfortunately, the nature of the abstractions that can be obtained through use of subroutines is limited ... not particaulr well-suited to description of abstract objects."
(Booch quotes in paper devoted to OOD.)

"interaction relations of the objects are still flying in the air and open for the programmer's imagination"

P Wegner and real vs object-oriented cats!

"agent encapsulates its dynamic behaviour in relation to the environment and other agents. It is an intrinsic property of the agent. " [whence] oracle, state, handle.

Issue: is model execution the key? if so, does it matter what kind? Harel would be using Statemate!

what really matter is just how we interpret the content and the method

Contribution to critique

Brings out the conceptual advantages of agent-oriented thinking

Use of software tools (Scout / DoNaLD / EDEN)

Has used the ADM intelligently for prototyping + other tools
Shows awareness of the idea of an abstract programming principle

Use of specification techniques

No LSD spec, but expresses motivation for such specs in write-up

Notes for the examiner

Essay on programming paradigm essentially: focus on comparison with OOP is loose, but develops as paper goes on.

Brooks and Harel revisited
unexpected behaviour:
bug vs lack of detail in requirements specification

Conceptual Encapsulation as process represented in development of programming languages from machine level to object-oriented paradigm.

Good quotes, esp on real cats from P Wegner

Comments

Potential for a good paper here I think. I like the development to OOP and the quotes. I especially liked the "real cat" quote. Your work shows good appreciation of some of the fundamental issues for agent-oriented modelling – there are problems for our present tools

and models in the dynamic configuration of dependencies that is illustrated to an extent in your 2d sorting example.

Look forward to exploring this further, and would like to write up a report on the work at some stage.

Tak Ming Philip Chan

State Charts vs Agent-oriented Modelling **40%**
Tape-deck Simulation + Statechart **60%**

Insight into principles

"LSD for automatic code generation" suspect concept?

as is " LSD is just a notation for behaviour specification "

... this is what I really I maintain it isn't! This perception of LSD may account for concerns in critique below:
e.g. are agents [agent actions?] orthogonal?

Contribution to critique

In a matter of hours, I am able to simulate a tape player and its behaviour.

LSD not very well defined. (Does this mean *well-defined*?)
Issues
other ways to view an agent
are agents orthogonal?
can functionality of agents overlap?
can we build a hierarchy of agents?

LSD seems to assume all variables are global

Use of software tools (Scout / DoNaLD / EDEN)

Good work in the digital watch modelling mould

Use of specification techniques

Interesting detail re definitive relationship between tape (as measured by number of songs) on reels 1 and 2. Can define reel2has initially, but will need to achieve symmetry to account for reverse direction play etc.

Notes for the examiner

Badly presented essay - layout very messy, and mixed-up references to statecharts and LSD.

Comments

I felt you missed out a little bit through not being around at the end of the module when I explored the significance of LSD more fully. You're certainly right to identify obscure points in the design of LSD, but I'd want to be careful that some of the considerations you have in mind aren't outside the intended scope of the notation. (Execution and control issues are an essentially separate story from agent perceptions and privileges, in my view, and great confusion will result from trying to tie them together. Indeed, that is what I think is at the root of many present tensions!)

The tapedeck is a good example of the kind that it would be good to write up properly and introduce as a standard demo. You / we should look to write a departmental report on this. I'd also be glad to discuss the LSD spec etc in more detail at some stage.

Monica Farkas

Definitive Methods Related to Automatic Systems **30%**
Modelling the Billiard Game **70%**

Insight into principles

Theme: separation of data representation from control (need to think about this, but definitely focussing on a very relevant issue: model state first then consider control later) not evident in simple mathematical computations, but is clearly emphasised in the case of scenario-oriented systems

"solution ... as a consequence of the correct description of the problem"

from *command structure* to *control structure* very useful concept:
command structure system instability {connects with difficulty of circumscription – it's the problem of open-loop design at the higher-level of abstraction}

Very significant and useful parallel being developed in this essay betwen the way that the study of control systems has developed from command structure, to preprogrammed feedback control structures, to intelligent feedback on-line. This is very much in keeping with the paradigm for development being exploited in the ADM.

Contribution to critique

[control logic over definitive reps of state]
"seems to be the change needed in the way we think about programming in order to exploit the coming generation of massively parallel multiprocessors" ?did I make this claim!

Use of software tools (Scout / DoNaLD / EDEN)

Very skilful adaptation of the dynamic modelling techniques introduced in the VCCS

Use of specification techniques

LSD agent format incorrect in places
– e.g. cue agent has inappropriate {}'s
score has newhit as a handle ?
Is end_of_a_hit used anywhere?
Do variables such as cue_enable get represented in the eden program?

Notes for the examiner

Don't understand some of the references in Part A:

philosophical system?
more extensional approach? – is declarative intended?

variables which pierce the interface

Is separation of data and control the correct diagnosis? Presumably the method of representing state is crucial to any representation of this separation in our work as a *new* concept.

Don't understand the picture of the introduction of a new component to the dashboard display .... wouldn't new physical object normally bring in more definitions, so augmenting P1, P2? And isn't interconnection between scripts more easily controlled than additions to the control logic?

Is the figure with ball2 and Ball2 meant to be a statechart or some other kind of formal state transition diagram?

The English, text and equation processing are not very satisfactory. Format of LSD specification / EDEN program poor in places.

Comments

Judged by the ideas and the quality of your practical implementation, this is quite outstanding work. Your contributions are in two directions:

1) drawing attention to a helpful and potentially fruitful parallel between systems design and the program development style represented in the ADM
2) implementing a complex system of interacting agents that is significantly more complicated than anything we have attempted of this kind in the past.

These excellent achievements deserve to attract a high mark. I hope that they can also be the basis for a publication. I should point out though that there are many things that need to be improved in respect of English, presentation, format of LSD specification before you can really do yourself justice and produce a report that is suitable for wider circulation.

Look forward to understanding your LSD specification in greater detail – at present there remain some obscure features, so far as I can see not represented in the eden implementation. It should become a standard demo!

Visal Phoeuk

DP vs FP wrt Parallel Programming **50%**
A Flock of Birds using LSD and ADM **50%**

Insight into principles

Don't favour the idea that definitive programming is oriented towards implicit parallelism *per se*. Presumably the (conceptual) "ideal" is that we embrace parallel machine models in the program development, and model the state of parallel machines using our standard principles.

Analysis is superficial, e.g.: ADM program is quite suitable for execution on a parallel architecture [with distributed memory].

Obscure reference to parallelisation of (the *variable*) can_see: does this refer to function evaluation in underlying algebra?

Some pretty controversial statements:

Appears to regard fine grain parallelism as necessarily good: experience of FP suggests that too fine-grain can introduce silly communication overheads.

"Data-flow machines are relatively new, but it sound very promising."

Contribution to critique

Use of software tools (Scout / DoNaLD / EDEN)

Has grappled with the ADM + DoNaLD

Use of specification techniques

Shows the right approach to the LSD specification

Notes for the examiner

Essay part

Initial presumption that what is needed is means to get compiler to generate parallelism even from a language without parallel constructs ... difficulty is to detect what parts of the code can be executed in parallel, whence data dependency etc. [At this point, there's some idea of a language that doesn't admit implicit parallelism.]

Alternative (?) have pl that admits implicit parallelism: 2 possible approaches are definitive programming and functional programming.

Consistently uses procedure where function in underlying algebra is probably intended.

Rather superficial account of parallelism in DP and FP.

Not much original thought ventured!

Comments

I'm sorry that you didn't get this example to work out in practice. We should try to put some work into figuring out where it fails. In principle, it's of more interest to model flocks of birds than to deal with more deterministic engineering devices, and there should be more scope to use the really interesting aspects of LSD specification. Perhaps there is

scope for cooperation with Benny Lam here – he's been looking at systems of insects in his practical study and it could be good to put these things together in a joint report. Your general approach seemed good though.

For the written bit, I would have liked to hear more from you personally by way of original thinking. I remain uncertain about the prospects for implicit parallelism ... at any rate modelling parallel machines using definitive methods would be my first thought in tackling the issue!

Hon Voon Phong

Comparative Study of Parallel Programming

Insight into principles

The treatment is basically bookwork, with well-organised body of standard ideas on a variety of architectural and programming paradigm issues.

Contribution to critique

Use of software tools (Scout / DoNaLD / EDEN)

Has used the visualisation effectively to depict the screen, but there is no evidence of animation beyond random motion of the players and shuttlecock within a restricted region of the court.

Use of specification techniques

Parts of the LSD specification look very plausible, but there appears to be no associated animation of the agent actions in EDEN. Introduction looks like wishful thinking about how implementation *might* be. No agent instantiation, only multiple specs for player1() and player2().

Notes for the examiner

1.0 Intro
Trends in parallel computing
currently most parallel programming techniques require the programmer to explicitly partition the task
non-portable
need to know about underlying architecture

2.0 PLs
procedural vs declarative
declarative languages are fundamentally *command* languages?

2.1 von Neumann Languages

2.2 Functional Languages

2.3. Definitive Languages

Most of the material is bookwork, directly extracted from lecture notes + sources.

Comments

I'm presuming that the badminton simulation is as yet incomplete. I like the graphical display you've developed, and think that the LSD specification looks promising. I hope that there will be a chance to discuss it with you more fully at some stage.

Your written is well-organised and structured, but I would have liked to hear more about your own ideas, or at least a fuller exploration of some of the ideas you cited. I have serious doubts about the fitness of a programming paradigm for parallelism in the narrow sense that the implicit parallelism in declarative programming school has promoted. My guess is that models of parallel machines have to come into the picture before serious progress can be made, and that modelling activity is in a sense already within the scope of what definitive programming entails.

Sirraj Kara

Definitive Methods for Concurrent Systems Simulation in relation to engineering of complex software systems 40%
Digital Watch Simulation 60%

Insight into principles

Interesting start:
[to program effectively] it is necessary to adopt a design methodology

cf oscilloscope probes circuit to display signals with the way in which printer talks to a computer in order to print an image

plugability and wireability

allow the user to think of which functions to be performed and which events should trigger these functions

Contribution to critique

Parallel with digital systems design "flip-flops" trigger the functions

Discusses what would be involved in OOP C++ approach

Useful comments on Harel's statechart

Use of software tools (Scout / DoNaLD / EDEN)

A digital watch simulation in EDEN with good quality of development on a par with VCCS. Clearly has applied his knowledge of a digital design paradigm!

Use of specification techniques

Fine LSD specification

Notes for the examiner

Strategy for paper

Consider problem issues for software design:
- Requirements Specification
- Complexity
- Cooperation
- Conformity
- Invisibility

Examine each in relation to definitive methods

Comments

Very good work indeed ... your paper is most well-conceived and organised. I liked the content of your paper very much ... brings out features that I identify with from my development work, and expresses them very clearly. The parallel with hardware design is new to me, but I have long suspected that electronic state is more like definitive state than Von Neumann program model state. I hope that you / we can write something on this as a report or for other publication forum at some stage.

It would be good to have a further discussion / explore your digital watch model in more detail at some stage. I look forward to finding time for this.