Action Selection and Individuation in Agent Based Modelling

Joanna J. Bryson
University of Bath
Department of Computer Science
Artificial models of natural Intelligence (AmonI)
http://www.cs.bath.ac.uk/~jjb

April 26, 2004

Abstract

This paper is a tutorial on action selection for Agent-Based Modelling (ABM). Having a clear idea of how you are organizing your agent's intelligence will make your code cleaner and easier to maintain, and your models easier to communicate to others. This paper describes four means of orgainizing agent action selection in increasing order of complexity These are: environmental determinism, finite state machines, basic reactive plans, and Parallel-rooted, Ordered, Slip-stack Hierarchical (POSH) reactive plans. Modellers should use the simplest mechanism possible — this paper describes the contexts in which more complicated mechanisms may be required, as well as suggesting coding and commenting schemes for all four systems.

This paper also addresses the issue of Individuated Agent-Based Modelling (IABM), where individual agents display different behavior. It gives examples of existing IABM systems and describes how these can be moved into more mainstream ABM simulators via two relatively simple mechanisms: either exploiting individual local variables or by specifying different priorities within the action selection mechanism. This allows individual agents to vary in their behavior while sharing the vast majority of their code.

1 Introduction

This paper discusses action selection in the framework of agent based modelling. Action selection is the means by which an autonomous agent solves

the ongoing problem of choosing what to do next. Action selection is the executive part of agent intelligence.

The remainder of agent intelligence — which is at least as important and difficult to construct — are the actions which are chosen between and the perceptions which inform the decisions and shape the acts. There is of course a tradeoff here in the granularity of control: the more an 'action' is capable of, the less complexity the executive must handle, but the less control it has. For example, the action 'walk-to attended-location' is at a much larger granularity than 'extend left-knee'. If you are developing an entire agent intelligence from scratch, then the best way to optimise this trade off is to follow an iterative development process that provides heuristics to allow refactoring when either actions or action selection become too complicated (see further Beck, 2000; Bryson and Stein, 2001b; Bryson, 2003). For most people doing agent-based modelling (ABM), however, the actions and perceptions are already provided, either by the simulation platform or in terms of existing libraries of behaviours from other developed simulations, so these issues will not be further discussed here.

Because agent-based modellers often lack experience with artificial intelligence, they frequently don't use formal action-selection mechanisms. This can make programming the agents unnecessarily complicated, and simulations more difficult to understand, maintain and extend. This paper sets out a description of a number of established idioms for action selection, each of which is useful in at least some domains. As such, it is intended to help modellers increase the clarity of their work. Another emphasis of this chapter is on relatively complex action-selection required for supporting *individuated* agent based modelling — ABM where each agent may be programmed with different behaviour. Section 2 begins with a short discussion of the utility of IABM. Section 3, which makes up the bulk of the paper, is a description of four different action-selection idioms, which may be useful for either conventional ABM or IABM. The paper then concludes with a description of engineering issues in bringing the more complex forms of action selection into familiar agent-based modelling environments, such as Swarm, RePast and NetLogo.

2 Individuated Agent Based Modelling

Much of the research which exploits ABM examines the patterns that emerge from the behaviour of very simple agents. This research has shown that such simple agents can do a good job of replicating the behaviour of com-

plex real-world actors, even humans, at a high level of abstraction. This level of abstraction seems to be most useful for large numbers of agents, where individual variation or details of decision making can be treated as random noise. In recent years, a number of software tools have been developed which support this sort of research, including Swarm, RePast and NetLogo.

There is another sort of agent-based social simulation though, where the individual agents are given relatively complex intelligent controls. This is typically done to simulate the behaviour of relatively small numbers of individual actors. One of the most spectacular examples of this kind of work is that of Tu (1999), which replicates aquatic animal behaviour from swimming through fluid dynamics up to mating and predation. Another example is the work by Hemelrijk (2002, 2000), which has made significant contributions to evolutionary theories of the differences in macaque social behaviour. Hemelrijk's work involves modelling colonies of primates with individual differences in initial rank. She has shown that differences in social organisation can emerge as a simple consequence of a single variable, the level of violence in the average antagonistic interaction (Hemelrijk, 2000).

Tyrrell (1993) built a complex simulated environment (which he creatively called the *SE*) to test a variety of action-selection mechanisms. The test agent in the *SE* is a small omnivorous animal which needs to survive and breed. This involves six types of subproblems:

- *Finding sustenance.* This includes water and three forms of nutrition, which are satisfied in varying degrees by three different types of food.
- *Escaping predators.* There are feline and avian predators, which have different perceptual capabilities and hunting strategies.
- *Avoiding hazards.* Latent dangers in the environment include wandering herds of ungulates, cliffs, poisonous food and water, temperature extremes and periodic (nightly) darkness. The environment also provides various forms of shelter including trees, grass, and a den.
- *Grooming.* Grooming is necessary for homeostatic temperature control and maintaining general health.
- *Sleeping.* The animal is blind at night and needs to sleep to maintain its health.

- *Reproduction.* The animal is male, thus its reproductive task is reduced to finding, courting and inseminating mates. Attempting to inseminate unreceptive mates is hazardous.

The success of the agent in the *SE* is counted as the number of times it mates in a lifetime. This is highly correlated with life length, but long life does not guarantee reproductive opportunities — these have to be actively sought. Tyrrell tested five well-known action-selection mechanisms from psychology and AI, one of which he chose as favourite and extended. What makes Tyrrell's work noteworthy is that there are very, very few AI or ALife domains in which a single agent must meet such a diverse set of goals. These include intrinsic and extrinsic goals, homeostatic and cyclic goals as well as opportunistic event-based ones. Of course, real animals deal with such conflicting goals and desires all the time.

The work of these three researchers has two things in common:

1. Compared to most ABM, the individual agents have relatively complex individual behaviours, and

2. They have all conducted their research in proprietary research environments which they have either developed or had developed for them at their institutions.

Many researchers would like to be able to produce models exploring their own theories or hypotheses working at this level of complexity, but are either unwilling or unable to accept the cost of constructing such simulations. I personally am aware of several theoretical biologists wanting to construct models of social insects (e.g. particular species of wasps and ants) with small numbers of members per colony, at least two groups of primatologists interested in exploring and demonstrating their own hypotheses which are at odds with Hemelrijk's, and a cross-disciplinary group working on understanding social predator communication. It would be nice if we could find a way to enable such research within existing ABM toolkits. If we cannot, it may be that a new toolkit is called for.

As indicated in the introduction, one way to get more complex behaviour from existing simulators is just to simplify code by using (and commenting) a good model of action selection. In the next section, I will go through several such models in increasing order of complexity. Following that, we will return to examining the question of whether current toolkits can support IABM.

3 Models of Action Selection

3.1 Environmental Determinism

There is no established name for what is really the simplest way to do action selection, so I have called it 'Environmental Determinism'. This model assumes that:

1. There are only a limited number of salient situations the agent can find itself in,
2. These situations are mutually exclusive, and
3. Actions can easily be mapped to situations.

While these assumptions may seem unrealistic, they have been usefully applied in sufficiently abstract models. The clearest example is probably Conway's Game of Life (Gardner, 1970), a very early ALife system, which takes place on a two dimensional grid. If I am allowed the liberty of referring to any cell of that grid rather than any live cell as an agent (which is how life is typically programmed) then Conway determined that there are nine environmental situations that matter, because the only thing that determines action in Conway's system is how many neighbours you have, and you can only have 0-8 neighbours on a 2-D grid. Figure 1 further summarises these into four situations. Too few neighbours (0-1) and the cell is dead, regardless of its previous state. For two neighbours the cell holds its current state, alive or dead. Exactly three neighbours and the cell is alive regardless of its previous state, but with four or more neighbours it is again dead regardless.

0-1	***2***	***3***	***4-8***
die	**stay**	**be-born**	**die**

Figure 1: *Environmental Determinism:* Enumerate the possible salient states of the environment, and state what action should be done in each. Example is Conway's Game of Life (Gardner, 1970).

In the event that a reader is unfamiliar with the Game of Life, I strongly recommend typing "game life" into Google^{TM}, to view the incredible variety of 'emergent' (higher-level) growth and action that results from this

simple program. There are claims that this system is a fairly realistic model of bacterial life in a petri dish, but I leave it to the reader determine their veracity.

Coding Environmental Determinism only requires a set of `if-then` statements. The environmental conditions should be made clear by using functions and function names to keep them clean, and of course comments. For example:

```
if (cell is dead) AND (number-of-neighbors is 3)
     then {set cell alive};  /* new cell is born */
if (cell is alive) AND (number-of-neighbors is 2 OR 3)
     then {no action};  /* Leave the cell alive */
if (cell is alife AND (number-of-neighbors is NOT 2 OR 3)
     then {set cell dead};  /*lonely or over-crowded cells die*/
if (cell is dead) AND (number-of-neighbors is NOT 3)
     then {no action}; /* leave cell dead */
```

3.2 Finite State Machines

In general, programmers prefer to think about agents rather than environments. Generally speaking, agents tend to be much simpler than there environment — they tend to have less possible behaviours than there are possible external situations. So programmers find it simpler to organise coding around actions, not events. The standard way of controlling many machines, notably AI game agents, is by using the abstraction of a Finite State Machine (FSM). Programming a finite state machine requires two thing:

1. Enumerating the states the agent can be in, and
2. Enumerating the causes for an agent to change state.

Again, assumptions are made that both states and transitions can be enumerated and that they are mutually exclusive.

For the purpose of ALife action selection, we can think of each state as being a situation in which an agent should perform a particular action. To go back to the Game of Life, we now have to think of the cell/agent as expressing one of two behaviours: looking alive, or looking dead (Figure 2).

Documentation for an FSM should ideally include a diagram. The coding should be the possible transitions clustered by state:

```
/* Transitions from state DEAD */
```

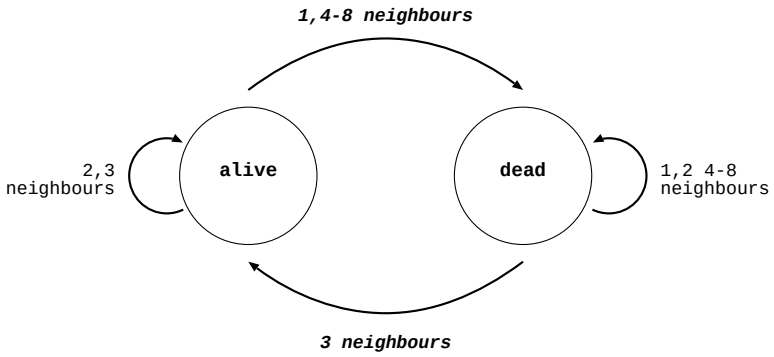

Figure 2: *Finite State Machines:* Enumerate the possible states the agent can be in / actions it might take, and the environmental contingencies that might make the agent change state. Example is also Conway's Game of Life (Gardner, 1970).

```
if (self dead) AND (number-of-neighbors is 3)
     then (self be-born);
/* Transitions from state ALIVE */
if (self alive) AND ((number-of-neighbors NOT in {2,3})
     then (self die);
```

To be formal, a finite state machine should also enumerate all possible environmental or internal events, and specify the situations in which no change takes place, like this:

The advantage of this is that one can check the code to make certain their are no possible transitions the programmer overlooked or forgot to code. The disadvantage is that it might make the code less readable, and that in most situations such a check may take intractably long. In general, programmers should try to put in as much code and comments as is required. For example, if a condition existed that either the programmer had to take a long time to think about or that another person needed explained to them, then those conditions deserve comments and possibly code to make the comments more explicit. For ALife modellers who run large and long simulations, it is almost never a good idea to code the cases where there is no change explicitly because they will take CPU time, though they can be coded for clarity and then commented out.

It's important to realise that in either environmental determinism or finite state machines, the programmer winds up needing to create discrete categories of both environmental events and behavioural acts. The only

differences are that in the FSM, the programmer also needs to enumerate states for the agent, and that actions are tied to these states rather than to the environmental categories. Often the states turn out to be a useful abstraction, but that doesn't have to be the case. For many simulations, it may be that the environment really is the more salient actor, and the agents really are simple enough that there is no reason to add theoretical entities such as internal states for the agents.

3.3 When Enumerating Transitions is Too Hard

Most agents, of course, have more than two possible actions. If there are a limited way of transitioning from each action to the next, then FSMs are a good way to describe that behaviour. However, if behaviour *is* largely driven by environmental prompting, and the environment is very dynamic and unpredictable (as when it contains many other types of complex actors), then there may be transitions from every state to every other state. This means that for every new action or capability added to an agent, as many transitions will need to be added to both it and to the other states as there are other capabilities. The number of transitions (and therefore the size of the code) grows quadratically, since all N nodes must have $N - 1$ transitions in them. It would be better to have a way to code action selection that didn't grow much faster than the number of possible actions.

Consider an example where the agents approach a more humanoid sort of intelligence than Conway's cells. Let's start with some arbitrary character chosen from a Jane Austen novel. The typical Austen character might be thought of as having four states: flirting, engaged, in church and married. A first cut at an FSM for this agent might look something like Figure 3.

The problem comes when we start trying to label the transitions. When you think about it, a good Georgian English agent wouldn't only go to church in order to get married, but might attend regularly at any stage of his or her life. On reading Austen further, one realises that marriage is not in fact a terminal state: some characters continue to flirt and may even become re-engaged. In fact, the only terminal state is death, which can occur at any time (Figure 4).

The problem with the FSM is that it is required to represent all possible transitions for an agent. But if we are trying to create a working agent, *we really only need to specify the transitions that the agent makes itself.* In fact, in the AI for abstract simulations, we generally only need to model the transitions that an agent might rationally choose to take in pursuit of its own goals.

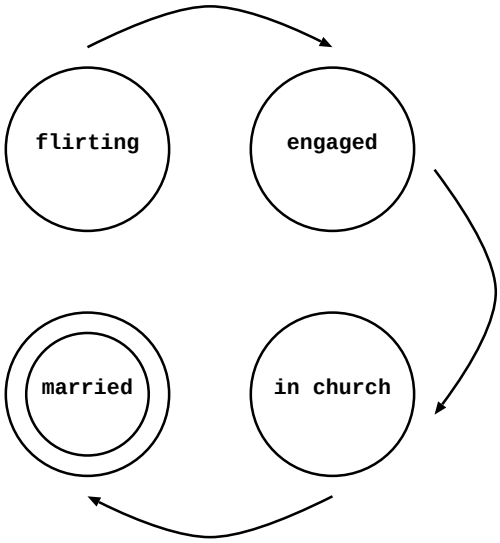

Figure 3: A first cut at an FSM for a Jane Austen character. The double circle on *marriage* indicates that it is a terminal state — the end of autonomous action selection, at least for this controller.

3.4 Basic Reactive Plans

The Basic Reactive Plan (BRP) is a name I've given to an idiom or pattern found in several (though not most) influential reactive planning architectures (Fikes et al., 1972; Nilsson, 1994; Bryson and McGonigle, 1998; Bryson and Stein, 2001a). Building a BRP requires a few assumptions. For example, it requires that one assumes that the agent has a goal, and that it has a set of actions it is capable of doing that can lead to solving that goal. The BRP is a prioritised list of these actions. The most important action is the one that consummates the goal, the second most important action is one that enables the most important act, and so on.

If the agent can already do the most important action it doesn't need to execute the other ones. A reactive plan recognises this situation, which means it can behave *opportunistically*. This means that each action is paired not only with a priority but also with a perceptual condition which allows it to know when it can execute that action. This is important not only for opportunism (skipping unnecessary steps in a conventional plan) but also for robustness (possibly repeating steps if they fail or if they simply need repeating such as digging a hole until you hit water).

Instead of having to explicitly code how to recognise every possible

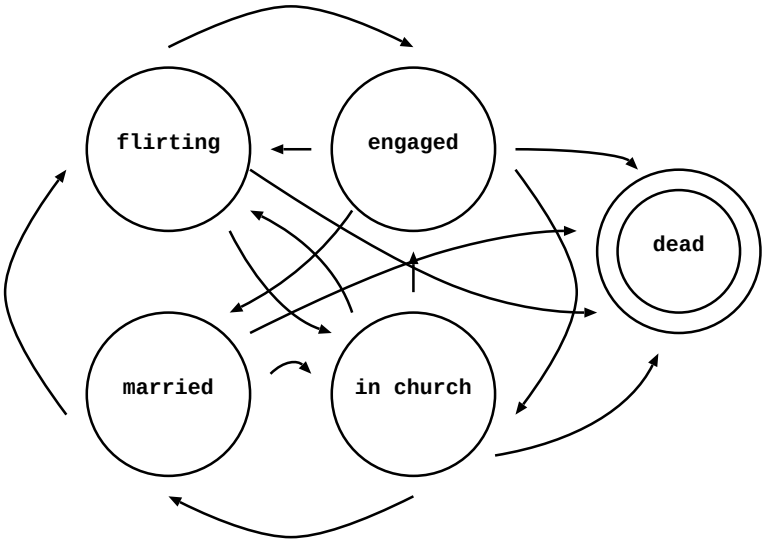

Figure 4: A somewhat more realistic draft FSM for an Austen character. Notice that labelling the transitions would still be fairly complex.

transition between action states, a BRP programmer needs only code how to recognise the situation in which each action can fire, a task made simpler by the invariant that no better (higher priority) action can fire, or the current one wouldn't be considered. This means that for each action, the programmer only has to hand-code one situation: the minimal requirements where that action might be usable. There is no reason to describe when the action should be skipped anyway, because that is encoded by its priority within the BRP.

Let's return to the Jane Austen agent. We'll assume we are programming one of the 'pure' characters with no devious intentions. The agent's highest goal is to become married. In order to become married, the agent must be in a church, but there is no point in going to the church without one's fiancé, and one can't have a fiancé without having gotten engaged. If one isn't engaged, one must flirt. Flirting of course also has preconditions, but for the sake of simplicity we'll stop here.

Here is a possible BRP:

1. (fiancé present AND in church) ⇒ marry
2. (fiancé present) ⇒ goto church
3. (engaged) ⇒ goto fiancé
4. (receiving attention) ⇒ become engaged
5. () ⇒ flirt

The numbers on the left list priority, with 1 being the highest. Thus if one is in a church but does not have a fiancé present, one does not get married, rather one either goes to be with the fiancé (if one exists) or else one flirts. Unless of course one is receiving attention already, in which case flirting can be skipped; one can become engaged and on the next iteration become married, assuming nothing has happened to remove the new fiancé from the church between program cycles. On the other hand, a character that never receives attention can flirt indefinitely, or, if suddenly receiving notice that it has become engaged (perhaps by arrangement of its parents), skip directly to going to the fiancé.

Obviously, encoding this BRP in an FSM would take at least 25 lines of code, assuming that the goal takes one line to describe and each possible transition between states could be specified in one line. Similarly, a large amount of possible environmental states have been neatly ignored as not relevant to this particular agent's pursuit of its goal.

For all its elegance, a BRP is simple to code in most programming languages. It is best coded as a switch (in Java or C) or cond (in Lisp), though it can also be coded as cascading if-then-else statements in pseudo-languages that lack this idiom. Details of building successful BRPs can be found in Bryson (2003), but essentially they are a simple derivation from a conventional, sequential plan e.g.

$$flirt \Rightarrow become\ engaged \Rightarrow go\ to\ church \Rightarrow get\ married$$

The BRP simply inverts the ordering and then specifies a mechanism for recognising when each item could be activated.

3.5 POSH Reactive Plans

The life-like agents such as I have described as Individuated above generally have more than one goal. Further many 'actions' may themselves require some multiple sub-actions to complete, and these in turn may require a BRP to organise.

If you examine the history of the action-selection literature (see for a review Bryson (2000a)), you find that there are three sorts of problems that any successful approach must address:

1. Some things must be checked at all times. For example, if you hear a loud noise, you will nearly always stop what you are doing and direct your visual attention toward the source, without any conscious processing of this attention switch.

2. Some things hardly ever need to be checked. For example, when you are walking, you have a reliable pattern for controlling your legs in turn which is characteristic of your individual gait. It would be very, very unusual for you to ever become aware of, let alone attempt to control or alter, the muscle patterns involved.

3. Some things need to be checked only in particular contexts, but then in unpredictable order. For example, if you are doing a jigsaw puzzle, you may have a set of rules about where you are piling edge pieces or pieces with a particular shade of blue on them. Unlike the walking situation, you cannot predict what the next piece your attention falls on will be like so your plan cannot be ordered in advance. This is the situation that requires a BRP.

It might seem simpler to represent everything as a sense/action pair — one enormous BRP[1]. However, it is untenable (that is, computationally intractable) to have every possible skill that requires this level of attention be equally accessible all the time. The reflexive response to the loud noise I mentioned is one of a relatively limited number of such stimuli (some learned, some innate) which seem to be stored in a separate (and fairly small) part of the brain, the amygdalic system. This is not only a consequence of combinatorics and computational limits, but also of perception and context. In another context, that shade of blue might trigger you to follow a friend wearing a particular shirt through a large crowd or to pass a ball to a teammate rather than having anything to do with puzzle pieces.

I have developed an action selection mechanism that supports all three of these sorts of situations with three types of representations and a development methodology to determine when to apply which (Bryson and Stein, 2001b; Bryson, 2001). The development methodology is called Behaviour Oriented Design (BOD), and is an iterative process for determining not only

[1]Or a large set of production rules — see e.g. (Newell, 1990).

which type of action-selection representation should be used, but also the granularity of the primitive actions. Primitives are encoded in object-like *behavior modules*, thus the name. Bryson (2003) gives a good summary of the heuristics and practicalities of using BOD.

The representational framework for BOD action selection is called Parallel-rooted, Ordered, Slip-stack Hierarchical (POSH) reactive plans. POSH reactive plans contain five types of elements. First there are the primitives *actions* and *senses*. There are only two differences between these:

- *Return values:* actions return no meaningful value, except in the case of radical failure, when a flag may break the system out of its current action-selection context, while senses return meaningful values which can be used in predicates for comparisons.

- *Duration:* some actions may take some time (though usually no more than 100ms). Senses are expected to return values very quickly, because many sensory preconditions may be checked on each cycle of the action-selection architecture, which ideally runs at at least 100Hz for real-time systems (e.g. robots), and orders of magnitude faster for simulations. Some actions (such as shifts of visual attention) also take place during sensory preconditions, but currently POSH plans have no separate type for these sense-speed acts.

The other three types of components in POSH plans are *action patterns* —- simple sequences to handle the second case above (things that almost always follow), *competences* — essentially BRPs which handle the third case above (things checked in certain contexts), and *drive collections*.

The drive collection is a special extension of the BRP. It serves as the root of the POSH plan hierarchy. It has several important special characteristics:

- There is only one drive collection, and it is checked on every iteration of the action-selection mechanism.

- Each element of the drive collection represents a separate goal for the agent. These goals may be being met in parallel, so each element of the drive collection keeps track of not only what it's immediate child is, but also what it was doing most recently — its current action-selection context.

- To facilitate parallelism, the drive elements may have associated frequencies. Thus some action (e.g. breathing, looking around) may

be very high priority every few seconds, but after initiating that action the action-selection mechanism is free to consider other, lower-priority goals over the next specified time interval.

The POSH action-selection mechanism is itself a sequential process, and it grants only course-grained parallelism and scheduling, because it is dependent on the return-time of the primitives. However, since the primitives are supported by independent modules (which may themselves be threaded) POSH agents can exhibit smooth, continuous parallel behavior.

For an example (drawn from Bryson and McGonigle (1998)), consider a robot which is moving through a cluttered space. The robot in a dynamic environment might need to resample its sensors 7 times a second, but there is no reason for it to stop moving while it does this. The primitive that creates movement can send the wheel drivers the current direction and speed for motion, with the understanding that the drivers will continue in that velocity until the next message is received[2].

4 Supporting IABM

Reflecting on the four types of action selection I've offered, let's return to the question of individuating agent based modelling. Can platforms such as Swarm, NetLogo and RePast support individuated action selection? Absolutely. The main requirement is only that each individual agent be able to have its own variable state. From here there are two possible solutions:

1. Each agent can have a copy of a common intelligence. So long as some decisions or other behaviour are made dependent on the content of the individual's behavior, their behavior will be individuated by the different values of these variables. This is effectively what Hemelrijk (2000) has done — her agent's behaviors vary only in their relative dominance ranking (their probability of success in a social contest is determined by this) and by gender (males may be influenced to approach females more frequently.)

2. Each agent may have a script describing its action-selection system as a piece of variable state. This is actually just a special case of the first solution — the common intelligence here is an action-selection mechanism which can interpret that script.

[2]Good robots also have timeouts associated with their drivers, so if the action-selection mechanism is hung or crashes, the robot will stop itself within a short time of not having had any instruction, e.g. a second.

I can demonstrate that both of these methods are plausible in existing ABM platforms. Of the three modelling platforms I've listed, Swarm and RePast have access to 'real' programming languages for describing agent intelligence (objective C and java), while NetLogo only allows modellers access to a toy / teaching language (Logo). Recently, two Bath M.Sc. students have replicated some of the most basic Hemelrijk (2000) results in NetLogo using method one above (Muhd Fathil, 2003; Wang, 2003). Thus even in the simplest of the three platforms, method one is possible.

We have not yet demonstrated method two in a major ABM platform, but Bryson (2000b) implements POSH action selection on a minor ABM platform, the *SE* developed by Tyrrell (1993). This works because of the sequential nature (described above) of the actual action-selection mechanism that exploits POSH plans. It was easy because I had a version of POSH in the native language of the *SE* available at the time. All that was required was that the POSH code was linked to the *SE* code, and then central iterator for POSH was replaced by an individual call to a single POSH program cycle from inside the modelled agent. Code for this implementation is available from the Web page for that project, currently:

`http://www.cs.bath.ac.uk/~jjb/web/edmund.html#code`

The main obstacle in the way of using POSH on a major ABM platform then is a version of POSH in the appropriate language. This would in practice be impractical done by modellers for NetLogo, because the overhead of the extra action selection mechanism would be too high — the system is already significantly slower than the other two platforms mentioned. On the other hand, there's no in principle reason that POSH couldn't be translated into java — it currently exists in C++ (Bryson and McGonigle, 1998), Lisp (CLOS) (Bryson, 2001) and Python (Kwong, 2003).

IABM by no means requires the complexity of POSH action selection — Method 1 above does not require it, nor did the Hemelrijk replications. It might also be possible to make a simpler systems for Method 2, though I have tried to keep POSH plan structures as parsimonious as possible. As I emphasised in the previous section, *any* sort of ABM can be improved and extended by simply making clear, intentional decisions about how the action selection will be handled, and by clearly coding and commenting the action-selection part of the agent's intelligence. In modelling as well as the rest of computer science, it's important to do the simplest, clearest thing possible for the problem at hand.

5 Summary

This paper has presented two related topics: how to make action selection better for agent based modelling, and how to individuate agent based modelling. The important steps of improving action selection are:

- Separate the problems of describing *how* the agent acts (coding its possible behaviors) and describing *when* it takes an action (coding it's action selection.)
- Use a standardised mechanism for describing your agent's intelligence. Here 'describing' means both coding and commenting. Four different action-selection frameworks were presented:

  1. Environmental determinism, which requires enumerating contexts the agent may find itself in then saying what it will do in each,
  2. Finite State Machines (FSM), which requires enumerating the things the agent might do, and then describing what might make it switch between possible actions.
  3. Basic Reactive Plans (BRPs), which require prioritising actions the agent might take to achieve some goal, then describing the minimal requirements for being able to take those actions.
  4. Parallel-rooted, Ordered, Slip-stack Hierarchical (POSH) reactive plans, which allow the encoding of full animal-like intelligence.

  The developer should choose the framework that most simply describes the minimum intelligence the agent will need.

The easiest way to individuate agent behaviors is to have all the agents share the bulk of their behavior code, the 'how' part, then to individuate their action selection, either by having their (shared) action selection program reference individual agent state in its decision making, or by providing different action-selection for different agents.

There is a danger that simplifying action selection coding may lead researchers to make more complicated models than necessary. Already our most difficult task as modellers is to analyse and explain the group behavior that emerges from the interactions of agents — the more difficult their behavior, the harder this explanation. I believe that having clearer code is

worth this risk. The history of science is full of examples where the first successful model was not the simplest, but given that it was the first it must have also been the most immediately intuitive. Once a good model has been built, simplifying it is part of the scientific process of analysing how it works. What good coding technique can do is to make it easier to build both the first and the simplified models.

Acknowledgements

I must acknowledge that I grossly oversimplified many Jane Austin characters. Thanks also to David Sallach and Tom Howe for encouraging me to write on this topic.

References

Beck, K. (2000). *Extreme Programming Explained: Embrace Change*. Addison-Wesley, Reading, MA.

Bryson, J. J. (2000a). Cross-paradigm analysis of autonomous agent architecture. *Journal of Experimental and Theoretical Artificial Intelligence*, 12(2):165–190.

Bryson, J. J. (2000b). Hierarchy and sequence vs. full parallelism in reactive action selection architectures. In *From Animals to Animats 6 (SAB00)*, pages 147–156, Cambridge, MA. MIT Press.

Bryson, J. J. (2001). *Intelligence by Design: Principles of Modularity and Coordination for Engineering Complex Adaptive Agents*. PhD thesis, MIT, Department of EECS, Cambridge, MA. AI Technical Report 2001-003.

Bryson, J. J. (2003). The behavior-oriented design of modular agent intelligence. In Kowalszyk, R., Müller, J. P., Tianfield, H., and Unland, R., editors, *Agent Technologies, Infrastructures, Tools, and Applications for e-Services*, pages 61–76. Springer.

Bryson, J. J. and McGonigle, B. (1998). Agent architecture as object oriented design. In Singh, M. P., Rao, A. S., and Wooldridge, M. J., editors, *The Fourth International Workshop on Agent Theories, Architectures, and Languages (ATAL97)*, pages 15–30. Springer-Verlag.

Bryson, J. J. and Stein, L. A. (2001a). Architectures and idioms: Making progress in agent design. In Castelfranchi, C. and Lespérance, Y., editors, *The Seventh International Workshop on Agent Theories, Architectures, and Languages (ATAL2000)*. Springer.

Bryson, J. J. and Stein, L. A. (2001b). Modularity and design in reactive intelligence. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence*, pages 1115–1120, Seattle. Morgan Kaufmann.

Fikes, R. E., Hart, P. E., and Nilsson, N. J. (1972). Learning and executing generalized robot plans. *Artificial Intelligence*, 3:251–288.

Gardner, M. (1970). Mathematical Games: The fantastic combinations of John Conway's new solitaire game 'Life'. *Scientific American*, 223(4):120–123.

Hemelrijk, C. K. (2000). Towards the integration of social dominance and spatial structure. *Animal Behaviour*, 59(5):1035–1048.

Hemelrijk, C. K. (2002). Self-organization and natural selection in the evolution of complex despotic societies. *Biological Bulletin*, 202(3):283–288.

Kwong, A. (2003). A framework for reactive intelligence through agile component-based behaviors. Master's thesis, University of Bath. Department of Computer Science.

Muhd Fathil, N. H. (2003). The study of the evolution of macaque complex despotic societies. Master's thesis, University of Bath. Department of Computer Science.

Newell, A. (1990). *Unified Theories of Cognition*. Harvard University Press, Cambridge, Massachusetts.

Nilsson, N. J. (1994). Teleo-reactive programs for agent control. *Journal of Artificial Intelligence Research*, 1:139–158.

Tu, X. (1999). *Artificial Animals for Computer Animation: Biomechanics, Locomotion, Perception and Behavior*. Springer.

Tyrrell, T. (1993). *Computational Mechanisms for Action Selection*. PhD thesis, University of Edinburgh. Centre for Cognitive Science.

Wang, J. J. (2003). Sexual attraction and inter-sexual dominance among virtual agents — replication of hemelrijk's domworld model with netlogo. Master's thesis, University of Bath. Department of Computer Science.