AGENT-BASED SOFTWARE ENGINEERING

Mike Wooldridge & **Michael Fisher**

Department of Computing
Manchester Metropolitan University
United Kingdom

M.Wooldridge@doc.mmu.ac.uk

Presentation Overview

1. Introduce Shoham's *agent-oriented programming* (AOP) proposal.
2. Is AOP a good idea *in principle*?
3. If it is a good idea, then how do we…
   - *specify*
   - *implement*; and
   - *verify*

   agent based systems.
4. Some research problems (tentative).

1 The AOP Proposal

- Yoav Shoham has proposed a 'new programming paradigm, based on a societal view of computation' (Shoham, 1990).
- The key idea is to build computer systems as societies of *agents*.

  The central features are:
  - agents are autonomous, concurrently executing computer processes;
  - agents are *cognitive systems*, programmed in terms of beliefs, goals, and so on;
  - agents are *reasoning systems*, specified in terms of logic;
  - (agents communicate via *speech acts* — typed message passing *à la* (Searle, 1969)).

1.1 Specifics

- Shoham proposed that a complete AOP system would have three components:
  - a logical system for specifying the mental state and behaviour of agents;
  - an interpreted programming language for programming agents using simple versions of the logical system;
  - an 'agentification' process, for generating executable agents from specifications.

- Shoham has only published results on the first 2 components:
  - the logic is a quantified modal logic, including direct reference to time, and modalities for representing beliefs and commitments;
  - the language, AGENT0, is a (very) simple rule-based language, with rules able to refer to the beliefs and commitments of agents.

(Shoham remarked that the third component is 'somewhat mysterious', but indicated that he was thinking along the lines of Rosenschein & Kaelbling's situated automata paradigm (Rosenschein and Kaelbling, 1986; Kaelbling and Rosenschein, 1990)).

1.2 Is AOP a Good Idea?

- Let's begin by asking whether AOP is a good idea *in principle* — ignore practicalities for the moment.
- AOP is distinguished from other programming paradigms by 2 providing 2 abstractions:
  - the *autonomous agent* abstraction;
  - the *cognitive agent* abstraction.

1.3 The Autonomous Agent Abstraction

- An agent in AOP (as in DAI) is an *autonomous concurrently executing reactive process*…

Autonomy: agents execute without direct human or other intervention, and have control over their own state — they can only be affected by message passing, not by direct manipulation of state.

Concurrency: agents execute independently of other agents.

Reactive: agents maintain an ongoing interaction with their environment — they respond, in a *rational* to events that occur.

Proactive: agents do not simply respond to their environment, they generate and act to achieve their own goals.

- This notion of an agent is somewhat similar to the notion of an object as used object-based concurrent programming (OBCP).
- In OBCP…
  - an object encapsulates some state, which is inaccessible to other objects;
  - the state can only be accessed indirectly, via message passing.;
  - the behaviour of an object is determined by a method/script.
- There are obvious similarities between AOP and OBCP; it is interesting that two essentially different research communities (AI/SE) should come up with such similar ideas.
- The main difference is that agents are *proactive*.
- This notion of an agent is probably interesting and uncontentious.

1.4 The Cognitive Agent Abstraction

- The second distinguishing feature is the use of *mentalistic* notions (knowledge, belief, desire, intention, …) to *program* agents.

  This is a much less mainstream idea.

- The idea is that in everyday life, we use a *folk psychology* to explain and predict the behaviour of complex intelligent systems — people.

  Michael *intended* to prepare his slides.
  Janine *believed* it was raining.

- This *intentional stance* is an *abstraction tool* — a convenient way of talking about complex systems, which allows us to predict and explain their behaviour without having to understand how the mechanism actually works.
- Now, much of computer science is concerned with looking for abstraction mechanisms (witness procedural abstraction, ADTs, objects, …)

  So why not use the intentional stance as an abstraction tool in computing — to explain, understand, and, crucially, program computer systems?

- This is the central idea of AOP.

- There seem to be 3 points in favour of this idea:

Characterising Agents

- It provides us with a familiar, non-technical way of *defining* agents.

Nested Representations

- It gives us the potential to specify systems that *include representations of other systems.*

  It is widely accepted that such nested representations are essential for agents that must cooperate with other agents.

Post-Declarative Systems

- AOP is a kind of post-declarative programming:
  - in procedural programming, we say exactly *what* a system should do;
  - in declarative programming, we state something that we want to achieve, give the system general info about the relationships between objects, and let a built-in control mechanism (e.g., goal-directed theorem proving) figure out what to do;
  - in AOP, we give a very abstract specification of the system, and let the control mechanism figure out what to do, knowing that it will act in accordance with some built-in theory of agency (e.g., the well-known Cohen-Levesque model of intention).

An Aside

- Again, we find that researchers from a more mainstream computing discipline have adopted a similar set of ideas…
- In distributed systems theory, *logics of knowledge* are used in the development of *knowledge based protocols* (Fagin et al., 1992).
- The rationale is that when constructing protocols, one often encounters reasoning such as the following:

  IF process i knows process j has received message m_1
  THEN process i should send process j the message m_2.

- In DS theory, knowledge is *grounded* — given a precise interpretation in terms of the states of a process; return to this later…

2 Three Issues

- If AOP *is* a good proposal, then there are a number of software engineering issues we must address.

  How do we:

  1. *specify*;
  2. *implement*; and
  3. *verify*

  agent-based systems.

2.1 Specifying Agent-Based Systems

- (This issue has received the most attention, so I will give it least space.)
- Agent specifications are generally given in a quantified multi-modal logic, containing some or all of the following features:
  - modalities for information attitudes;
  - modalities for pro-attitudes;
  - representation of action;
  - representation of time.
- *The key problem in developing a framework for specifying agent theories is to sort out the relationship between these different aspects of agency.*

  For example, even the relationship between knowledge and belief is the subject of debate! The relationship between action and other attitudes is *very* troublesome.

- Here are some of the dimensions along which an agent specification language may vary.

Information attitudes

- knowledge;
- belief;
- mutual information attitudes;

Pro-attitides

- desire;
- intention;
- obligation;
- commitment;
- choice;
- joint/collective pro-attitides;
- …

Action

- direct representation, *à la* dynamic logic (Harel, 1984);
- implicit representation;

Time

- linear/branching;
- dense/discrete;
- direct reference/tense operators;
- point based/interval based.

- There are a lot of choices to be made; a good agent theory must pick on a small subset of attributes, and explain the relationships between these attributes.
- Some good current accounts: (Cohen and Levesque, 1990; Rao and Georgeff, 1991; Singh and Asher, 1991).
- (And of course, (Wooldridge, 1994)!)

2.2 Implementing Agent-Based Systems

- This is probably the key problem in AOP:

  Given a specification φ, expressed in some logical language L, how do we construct a system S such that $\{S\} \models \varphi$?

- (Note that I'm assuming we have a *logical* agent specification — I take this to be a central component of AOP.)
- There seem to be 2 possibilities:
  1. directly execute φ;
  2. compile φ into a directly executable form.

Executing Agent Specifications

- What does it mean, to execute a formula φ of logic L?
- It means *generating a logical model, M, for φ, such that $M \models_L \varphi$.*
- If this generation is done without interference — if the agent has complete control over its environment — then execution reduces to *constructive theorem proving*:

  show that φ is satisfiable by building a model for φ.

- However, agents are not interference-free; they must construct a model in the presence of input from the environment.
- Model-building is thus an iterative process:
  - environment makes something true;
  - agent responds by making something else true, in such a way as to satisfy φ;
  - environment responds;
  - …

- Execution and theorem proving are closely related

  $\Rightarrow$

  execution of sufficiently rich (quantified) languages not possible.
- At first sight, it might seem that suggesting the direct execution of complex agent specification languages is naive; that it is exactly the kind of suggestion that detractors of symbolic AI hate.
- One should be very careful about what claims or proposals we make; in particular, there has been no work done on directly executing the kind of multi-modal logic that we use to specify agents.
- However, in certain circumstances, execution of interesting agent specification languages *is* possible.

Concurrent METATEM

- (Fisher and Wooldridge, 1993) describes a DAI programming language called Concurrent METATEM, which is based on the direct execution of temporal logic.
- A Concurrent METATEM system contains a number of concurrently executing agents, each of which is programmed by giving it a TL specification of the behaviour of it is intended the agent should exhibit.
- A specification has the form

  $$\Box \bigwedge_i P_i \Rightarrow F_i$$

  where P_i is a TL formula referring only to the present or past, and F_i is a TL formula referring to the present or future.
- (The separation theorem of Gabbay tells us that an arbitrary TL formula can be rewritten into this form.)

- The general idea is

  on the basis of the past *do* the future.
- Internally, these rules are further rewritten into *separated normal form*, removing much of the structural complexity of the formula.
- Agents in Concurrent METATEM communicate by broadcast message passing; each agent is given an *interface*, which determines the messages it can send and receive.
- An example Concurrent METATEM agent definition (resource controller):

  $rc(ask)[give]$:
  $\bigcirc\!\!\!\!\bullet\ ask(x) \Rightarrow \Diamond give(x)$;
  $(\neg ask(x)\ \mathcal{Z}\ (give(x) \wedge \neg ask(x)) \Rightarrow \neg give(x)$
  $give(x) \wedge give(y) \Rightarrow (x = y)$
- Rule 1: if someone askes, then eventually give;

  Rule 2: don't give unless someone has asked since you last gave;

  Rule 3: if you give to 2 people, they must be the same person (i.e., don't give to > 1 person at a time).

Snow White and the Dwarves

- To illustrate Concurrent METATEM in more detail, here are some example programs…
- Snow White has some sweets (resources), which she will give to the Dwarves (resource consumers).
- She will only give to one dwarf at a time.
- She will always eventually give to a dwarf that asks.
- Here is Snow White, written in Concurrent METATEM:

  Snow-White(ask)[give]:
  $\bigcirc\!\!\!\!\bullet$ ask(x) $\Rightarrow$ $\Diamond$ give(x)
  give(x) $\wedge$ give(y) $\Rightarrow$ (x = y)

- The dwarf ‘eager’ asks for a sweet initially, and then whenever he has just received one, asks again.

eager(give)[ask]:

$$\textbf{start} \Rightarrow \text{ask(eager)}$$
$$\bigcirc\ \text{give(eager)} \Rightarrow \text{ask(eager)}$$

- Some dwarves are even less polite: ‘greedy’ just asks every time.

greedy(give)[ask]:

$$\textbf{start} \Rightarrow \Box\ \text{ask(greedy)}$$

- Fortunately, some have better manners; ‘courteous’ only asks when ‘eager’ and ‘greedy’ have eaten.

courteous(give)[ask]:

$$((\neg\ \text{ask(courteous)}\ \mathcal{S}\ \text{give(eager)}) \wedge (\neg\ \text{ask(courteous)}\ \mathcal{S}\ \text{give(greedy)})) \Rightarrow \text{ask(courteous)}$$

- And finally, ‘shy’ will only ask for a sweet when noone else has just asked.

shy(give)[ask]:

$$\textbf{start} \Rightarrow \Diamond\ \text{ask(shy)}$$
$$\bigcirc\ \text{ask(x)} \Rightarrow \neg\ \text{ask(shy)}$$
$$\bigcirc\ \text{give(shy)} \Rightarrow \Diamond\ \text{ask(shy)}$$

Some Remarks

- Why does executing TL work? Because TL is (amongst other things) a language for expressing *constraints* that must hold between successive states.
- Execution is thus a process of generating constraints as past-time antecedents are satisfied, and then trying to generate part of a TL model in the presence of these constraints.

- In contrast to the kind of agent specification language we discussed above, the TL we use is *very* simple (and yet is still highly undecidable). Because models for TL are so simple (linear discrete sequences of states), it is clear how we might execute a formula.
- *But how are we to execute agent specification languages, with modalities for beliefs, desires, and so on?*

  Even if it was theoretically possible, could we make it computationally tractable?

  (Note that languages like AGENT0 do *not* implement their associated logics — the correspondence between the logic and language is actually quite loosely defined.)
- We are looking at incorporating belief modalities into Concurrent METATEM — watch this space!

Compiling Agent Specifications

- An alternative to direct execution is *compilation.*
- In this paradigm, we take a specification φ, and do a constructive proof of the satisfiability of φ — this involves generating a logical model for φ.
- Models for modal logics are typically directed graphs, which closely resemble automata.

  Thus, the idea goes, generate a model for φ, and read off the automata for φ from the model that results.

  You end up (hopefully) with a very simple machine that does not do complex symbolic reasoning.

- Much similar work in mainstream computer science:
  - (Manna and Pnueli, 1984) generate *synchronisation skeletons* for concurrent systems from linear TL specifications;
  - (Pnueli and Rosner, 1989) generate reactive modules from branching TL (CTL*) formulae;

  Conclusions of this work: it's hard!

  Works well in the propositional case — not well at all in general.

  Again, we run up against undecidability of quantified logic, and the general difficulty of theorem proving.

Situated Automata

- One application of this technique in AI, is the *situated automata* paradigm (Rosenschein and Kaelbling, 1986).
- In this approach, an agent contains 2 parts:
  - a perception part;
  - an action part.

- The perception part is specified in terms of a logic of knowledge; the possible worlds underlying this logic are given a precise interpretation in terms of the states of an automaton.

  The RULER program takes a perception specification and compiles it down to an automaton, by doing a constructive proof.
- The perception part is specified in terms of *goal reduction rules*, which encode information about how to achieve goals.

  The GAPPS program takes these goal reduction rules, and from them generates, a set of situation action rules (encoded in the form of a digital circuit).

- The situated automata approach looks very attractive.
- Question: What are the limitations of the approach?
  - it can't work for arbitrary (quantified) specifiations, so how far can we go?
  - logic of situated intentions/desires?
  - can an agent learn?

3 Verifying Agent-Based Systems

- Finally, given a system S and specification φ, how do we *verify* S with respect to φ, i.e., how do we show that $\{S\} \models \varphi$?
- Two approaches:
  - semantic (model checking);
  - axiomatic.

Model Checking

- In mainstream computer science, model checking means:
  - taking a program Π, and from that program generating the model M_Π corresponding to that program;
  - showing that $M_\Pi \models \varphi$, i.e., that the specification formula φ holds in M_Π, and thus that the program satisfies its specification.
- Model checking is generally reckoned to be easier than theorem proving!

- (Rao and Georgeff, 1993) have described a process for model checking AOP systems.

  They give an algorithm for taking a logical model for their (propositional) agent specification language, and a formula of the language, and determining whether the formula is valid in the model.
- However, it is not clear where the logical model characterising an agent actually comes from — can it be derived from an arbitary program Π, as in mainstream computer science?
- *In general, it is not clear how the goal, intention, etc., accessibility relations can be associated with an agent. This is the problem of grounding abstract possible worlds semantics — giving them a concrete interpretation.*

Axiomatic Methods

- (Wooldridge, 1992) proposed an *axiomatic* method for verification.
- The idea is:
  - develop a formal model of the type of system you wish to reason about;
  - use the histories traced out in the execution of such as system as the semantic basis for a logic, which can be used for representing the properties of such systems;
  - show how the theory of a system can be derived in your logic;
  - use the proof theory of your logic to verify properties of the implemented system.
- Main problems:
  - good models of concurrency;
  - automating proof theory…

4 AOP Research Directions

- General:
  - is the intentional stance appropriate? some experiments?
- Specification:
  - automated proof methods for agent specification languages — the enabling technology for AOP?
  - agent logics that can account for action;
  - a complete agent theory.

- Implementation:
  - limitations of compilation (situated automata):
    - how far towards first-order can we take it?
    - can we give a good *grounded* semantics for goals, intentions, and the like?
    - learning? (apparently, Kaelbling has done some work on this.)
    - algorithms for synthesising agents from intentional specifications.
  - execution (@ MMU)
    - efficient implementation of Concurrent METATEM;
    - extra modalities (belief) in Concurrent METATEM;
    - groups as first-class objects in Concurrent METATEM.

- Verification:
  - model checking — can we derive a model for an agent specification language from an arbitrary program?
  - axiomatic approaches
    - good models of concurrency.;
    - realistic models of agents.

References

Cohen, P. R. and Levesque, H. J. (1990). Intention is choice with commitment. *Artificial Intelligence*, 42:213–261.

Fagin, R., Halpern, J. Y., and Vardi, M. Y. (1992). What can machines know? on the properties of knowledge in distributed systems. *Journal of the ACM*, 39(2):328–376.

Fisher, M. and Wooldridge, M. (1993). Executable temporal logic for distributed A.I. In *Proceedings of the Twelfth International Workshop on Distributed Artificial Intelligence (IWDAI-93)*, pages 131–142, Hidden Valley, PA.

Harel, D. (1984). Dynamic logic. In Gabbay, D. and Guenther, F., editors, *Handbook of Philosophical Logic Volume II — Extensions of Classical Logic*, pages 497–604. D. Reidel Publishing Company. (Synthese library Volume 164).

Kaelbling, L. P. and Rosenschein, S. J. (1990). Action and planning in embedded agents. In Maes, P., editor, *Designing Autonomous Agents*, pages 35–48. The MIT Press.

Manna, Z. and Pnueli, A. (1984). Synthesis of communicating processes from temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 6(1):68–93.

Pnueli, A. and Rosner, R. (1989). On the synthesis of an asynchronous reactive module. In *Proceedings of the Sixteenth International Colloquium on Automata, Languages, and Programs.*

Rao, A. S. and Georgeff, M. P. (1991). Modeling rational agents within a BDI-architecture. In Fikes, R. and Sandewall, E., editors, *Proceedings of Knowledge Representation and Reasoning (KR&R-91)*, pages 473–484. Morgan Kaufmann Publishers, Inc.

Rao, A. S. and Georgeff, M. P. (1993). A model-theoretic approach to the verification of situated reasoning systems. In *Proceedings of the Thirteenth International Joint Conference on*

Artificial Intelligence (IJCAI-93), pages 318–324, Chambéry, France.

Rosenschein, S. and Kaelbling, L. (1986). The synthesis of digital machines with provable epistemic properties. In Halpern, J. Y., editor, *Proceedings of the 1986 Conference on Theoretical Aspects of Reasoning About Knowledge*, pages 83–98. Morgan Kaufmann Publishers, Inc.

Searle, J. R. (1969). *Speech Acts: An Essay in the Philosophy of Language.* Cambridge University Press.

Shoham, Y. (1990). Agent-oriented programming. Technical Report STAN–CS–1335–90, Department of Computer Science, Stanford University.

Singh, M. P. and Asher, N. M. (1991). Towards a formal theory of intentions. In *Logics in AI — Proceedings of the European Workshop JELIA-90 (LNAI Volume 478)*, pages 472–486. Springer-Verlag.

Wooldridge, M. (1992). *The Logical Modelling of Computational Multi-Agent Systems.* PhD thesis, Department of Computation, UMIST,

Manchester, UK. (Also available as Technical Report MMU–DOC–94–01, Department of Computing, Manchester Metropolitan University, Chester St., Manchester, UK).

Wooldridge, M. (1994). Coherent social action. In Cohn, A., editor, *Proceedings of the Eleventh European Conference on Artificial Intelligence (ECAI-94)*. John Wiley & Sons.