A future for agent programming?

Brian Logan

School of Computer Science
University of Nottingham, UK

This should be our time …

- increasing interest in and use of autonomous intelligent systems (cars, UAVs, manufacturing, healthcare, personal assistants, etc.)

  "*autonomous systems … [are] a critical area underpinning continued growth in so many of the UK's high priority sectors*"

  – (UK KTN Aerospace, Aviation & Defence)

- special tracks (Cognitive Systems, Integrated Systems) at AAAI 2015 & 2016

- but the impact of agent programming in these areas is minimal

Instead

- AOPL and AOSE usage is limited (Dignum & Dignum 2010; Winikoff 2012, Müller & Fischer 2014)

- Winikoff (2012) notes:
  - applications (described in the AAMAS Industry/Innovative Applications tracks) do not require goal-based agents

  - focus of many applications is at the MAS coordination level (game theory, MDPs)

Current state of affairs

- Müller & Fischer (2014) report 46 ‘mature’ applications (out of 152 surveyed)
- 82% of mature applications focus on the MAS level, while only 9% focus on ‘intelligent agents’
- majority of mature applications are concentrated in a few industrial sectors: logistics & manufacturing, telecoms, aerospace and e-commerce
- only 10% of mature applications clearly used a BDI platform

What can we do?

- why is there a lack of interest in agent programming?
- what can we (should we) do about it?

Proposed solutions (Winikoff 2012)

- re-engage with industry
- stop designing AOPLs and AOSE methodologies … and instead …
- move to the “macro” level: develop techniques for designing and implement-ing interaction, integrate micro (single cognitive agent) and macro (MAS) de-sign and implementation
- develop techniques for the assurance of MAS
- re-engage with the US

Proposed solutions (Hindriks 2014)

- stop talking about BDI agents and start talking only about Cognitive Agents
- easy access to powerful AI techniques, e.g., combining AOP and planning
- demonstrate that AOP solves key concurrency and distributed computing issues
- integrate methodologies and APLs
- address AOSE issues (e.g., autonomy as a practical software property)
- mature tooling for agent development
- standard interfaces for cognitive agents
- high-performance cognitive agents
- logic based agents are simpler

The problem is not …

- the methodology
- development tools
- performance of the agents
- how the agents interact
- or even whether there is anyone from Texas in the room …
- all of these things (well most) are important, but they are not the key issue

What problem are we trying to solve?

"*Artificial Intelligence is concerned with building, modeling and understanding systems that exhibit some aspect of intelligent behaviour. Yet it is only comparatively recently -- since about the mid 1980s -- that* ***issues surrounding the synthesis of intelligent autonomous agents*** *have entered the mainstream of AI. Despite the undoubted interest on the part of the international research community, there is currently no recognised forum for presenting work in this area. … The aim of this workshop, therefore, is to provide an arena in which researchers working in all areas related to the theoretical and practical aspects of both hardware and software agent synthesis can further extend their understanding and expertise by meeting and exchanging ideas, techniques and results with researchers working in related areas.*"

– ATAL 1994 CfP

Agents = AI

- in this view agents are seen as '*autonomous computer programs, capable of independent action in environments that are typically dynamic and unpredictable'*

- agents combine multiple capabilities, e.g., sensing, problem-solving and action, in a single system

- agent programming can be seen as a means of realising flexible intelligent behaviour in dynamic environments

- essentially the same goals as early AI projects – not just 'objects with attitude' (Bradshaw 1997)

Why we are failing to make an impact

we can't solve a large enough class of AI problems well enough to be interesting to the wider AAMAS community or application developers

The BDI model

- in BDI APLs, the behaviour of an agent is specified in terms of beliefs, goals, and plans

- for each event (belief change or top-level goal), the agent selects a plan which forms the root of an intention and commences executing the steps in the plan

- if the next step in an intention is a subgoal, a (sub)plan is selected to achieve the subgoal and added to the intention

- this process of repeatedly choosing and executing plans is referred to the agent's deliberation cycle

- deferring the selection plans until the corresponding goal must be achieved allows BDI agents to respond flexibly to changes in the environment, by adapting the means used to achieve a goal to the current circumstances

What current BDI platforms can do

- select canned plans at run time based on the current context
- some support for handling plan failure (e.g., trying a different plan)
- all of which is useful …
- but everything else is left to the programmer
- cf ‘Worse is Better’ (Gabriel 1991)

What we can’t do (in a generic way)

- handling costs, preferences, time, resources, durative actions, etc.
- which plan to adopt if several are applicable
- which intention to execute next
- how to handle interactions between intentions
- how to estimate progress of an intention
- how to handle lack of progress or plan failure
- when to drop a goal or try a different approach
- etc

Why we need to solve these problems

- for specific applications, the detailed answers to these questions will vary
- but are we really saying that there are no general theories or approaches to these questions?
- if so, developing interesting agents is going to be very hard, and very expensive
- and the amount that agent programming can contribute will be limited

Where we are now

- there has been some work in these areas
- however it’s not being merged into mainstream platforms
- there is also less and less of it at EMAS, and more and more emphasis on how to engineer systems whose behaviour is often not terribly interesting (sorry)
- the support that we can currently offer is useful, particularly for some problems
- but it will never be useful enough for developers to switch platforms, even if we polish our methodologies and tools

It gets worse

- if we don’t solve these problems, others will:
    - key assumptions on which the BDI agent programming model are based are not as true as they were, allowing other AI subfields to colonise the APL space
    - some mainstream computing paradigms are starting to look like simple forms of agent programming

Reactive planning

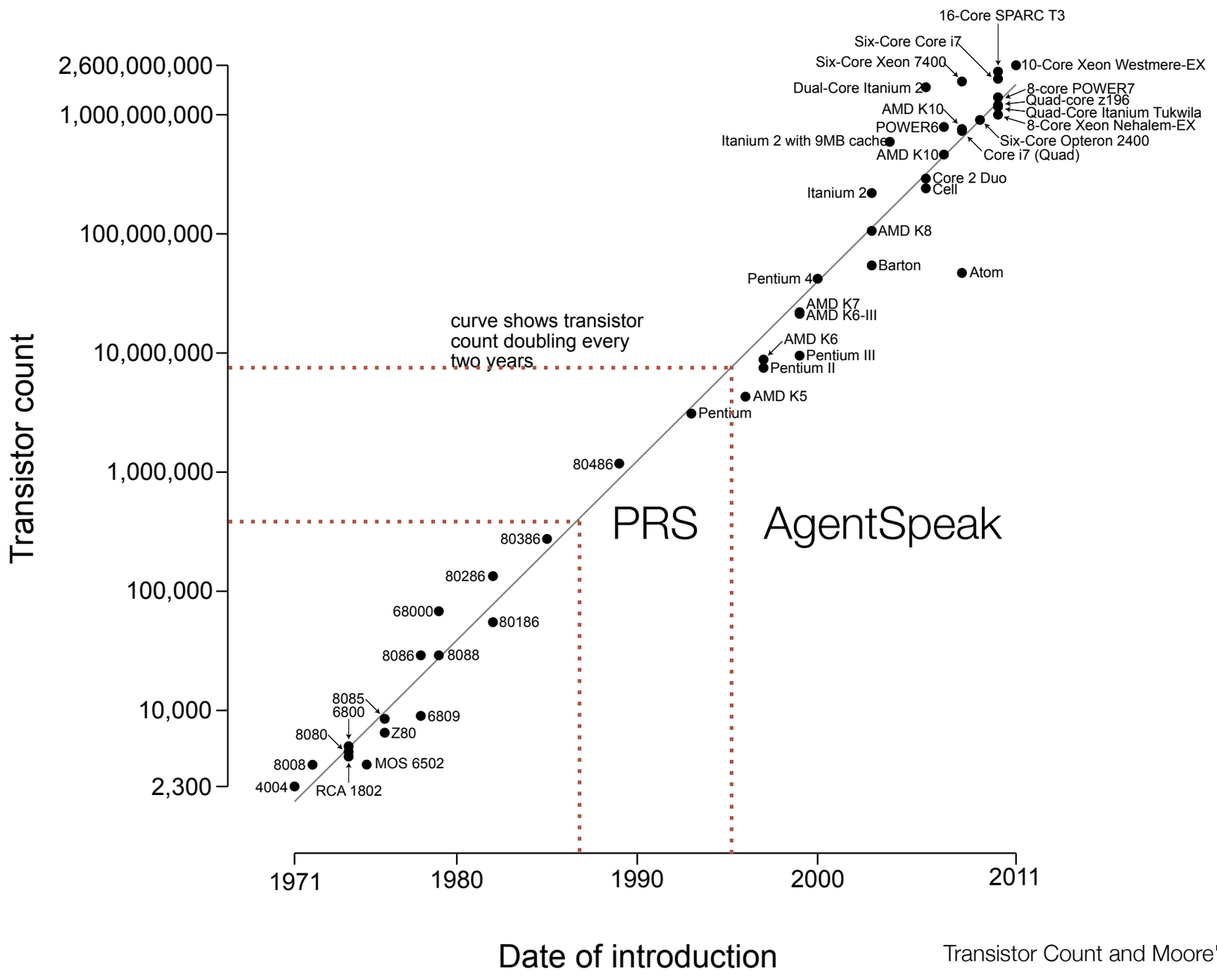
- the BDI model is based on early work on reactive planning, e.g., (Georgeff & Lansky 1987)

- reactive planning is based on a number of assumptions, including:
  - the environment is dynamic, so it's not worth planning too far ahead as the environment will change

  - choice of plans should be deferred for as long as possible – plans should be selected based on the context in which the plan will be executed

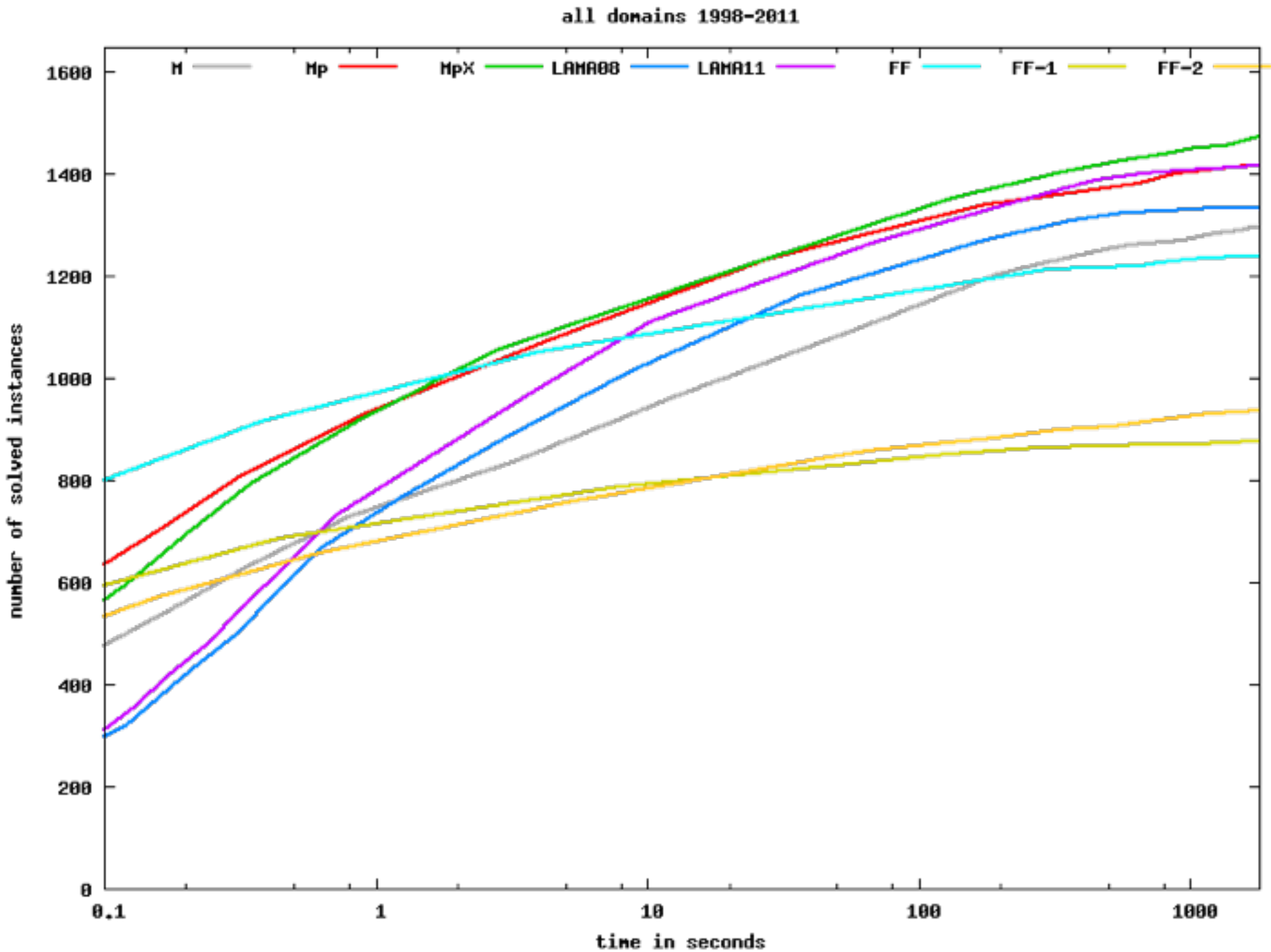
- "*[traditional] planning techniques [are] not suited to domains where replanning is frequently necessary*" (Georgeff & Lansky 1987)

Available computation



Transistor Count and Moore's Law - 2011, by Wgsimon

First principles planning



Jussi Rintanen

Reactive programming

- at the same time, work on event-driven and reactive programming (e.g., in robotics) offers similar (or better) functionality to belief triggered plans
    - well defined model of streams (immutability, sampling, pull-based computation)
    - very fast (microsecond) evaluation of simple SQL-like queries (LINQ, cqengine) that scale to very large ‘belief bases’ for evaluation of context conditions
- together can provide a simple form of event-driven reactive agent behaviour (e.g., if subgoals are seen as a stream of events)
- these paradigms are now ‘mainstream’ (e.g., ACM curriculum)

if we are not careful, these guys will eat our lunch

The good news

- advances in both hardware and related AI disciplines mean we are now in a position where we can rethink the foundations of agent programming languages

- by engaging with cutting edge AI research, we can address some of the key challenges in agent development that have been largely ignored for the last twenty years

- this will involve a change of emphasis:

  - agent programming should be more about describing the problem rather than ‘hacking code’

  - with the agent programming language/platform doing (more of) the hard bits

How to make progress

- the basic idea of programming in terms of propositional attitudes is sound (at least we have no reason to think it isn’t)

- however our models of beliefs, goals, plans, capabilities, intentions, commitments etc. are basic at best

- one way to make progress is to try to elaborate each of these components in turn

- but never in isolation – challenge is to extend the capabilities of the agent as a whole

- while being able to verify the behaviour of the resulting system

Some ideas

- **beliefs:** how and when beliefs are updated (active sensing, lazy update); handling uncertain and inconsistent beliefs (implications for plan selection)

- **goals:** goals with priorities and deadlines; maintenance and other repeating goals; when to adopt, suspend and drop goals (cf work on goal life cycles); how to tell if a goal is achieved (e.g., if beliefs are uncertain)

- **plans:** plans with durative and nondeterministic actions; plans with partially ordered steps; when (and how) to synthesise new plans

- **intentions:** how to estimate the time required to execute an intention; which intentions to progress next; how to schedule intentions to avoid interference; how to handle plan failure

- **MAS level:** how to decide when to join an open system/coalition/team; deliberation about roles, norms etc; strategic reasoning about other agents

What counts as progress

- extensions need to be integrated, e.g., uncertain and inconsistent beliefs have implications for plan selection and determining when a goal is achieved

- all of this needs to be modular, so that an agent “developer” only needs to master the features necessary for their application

- evaluation will require richer benchmark problems (or less toy versions of current problems) – but we need this anyway

- key evaluation criterion should be whether the developer has to program something at all

Summary

- agent programming isn’t (and can’t be for a long while) primarily about software engineering

- software engineering is important, but only as a means to an end

- what we need to focus on is solving more interesting AI problems in an integrated, general and tractable way

- then people will be interested …