"What is an Agent and Why Should I Care?"

Tim Menzies[1], Adrian Pearce[2], Clinton Heinze[3], Simon Goss[3]

[1] Lane Department of Computer Science, West Virginia University,
PO Box 6109, Morgantown, WV, 26506-6109, USA, tim@menzies.com
[2] Department of Computer Science and Software Engineering The University of Melbourne,
Victoria, 3010, Australia, pearce@cs.mu.oz.au
[3] Air Operations Division, Aeronautical & Maritime Research Laboratory, Melbourne,
Australia, {clinton.heinze|Simon.Goss}@dsto.defence.gov.au

Abstract. A range of agent implementation technologies are reviewed according to five user-based criteria and via a comparison with object-oriented programming. The comparison with OO shows that some parts of object technology are a candidate implementation technique for some parts of agent systems. However, many other non-object-based implementation techniques may be just as useful. Also, for agents with mentalistic attitudes, the high-level specification of agent behavior requires numerous concepts outside the object paradigm; e.g. plans, communication, intentions, roles, and teams.
KEYWORDS: evaluation, agent-oriented, object-oriented.

1 Introduction

Is there anything really new in agent-oriented software? Are agents a bold step forward into the future of software? Or is agency just "new wine in old bottles"?

Our users demand answers to these questions, and others. One gruff user always asked "what are agents and *why should I care*?". To such users, the issue in italics is the key question. Agent technologies are interesting to users *only* if those technologies address issues of interest to the users.

After explaining agents to this gruff user, this users next comment was "this sounds just like OO to me; what's new here?". Such comments motivate this article. Our response to these comments is in three parts:

1. We carefully define the core concepts of agent-oriented software and object-oriented software.
2. Next, we review the diverse range of software labelled "agents".
3. This software is then assessed these concepts with respect to certain user-oriented issues.

The user issues used in this article come from the Australian Workshops on Agent-Based systems. Those workshops have debated the relative merits of the agent implementation technologies shown in Figure 1. In those debates, the technologies were assessed with respect to the problem of building agents for the Air Operations Division (AOD) of the Australian Defense Science Technology Division.

Name : *Notes*	*Introduced in...*
OO : Object-oriented	§2.1
Standard BDI : BDI= beliefs, desires, intentions	§2.2
FORTRAN : How we used to build agents	§3.1
dMARS : A commercial agent-oriented BDI tool	§3.1
Command agents : Heinze and Pearce's extension to dMARS	§3.1
Behavioural cloning : Machine learning to build agents	§3.1
Petri nets :	§3.3
TACAIR/ SOAR/ PSCM : The problem space computational model (PSCM) is how the rule-based system called SOAR implements TACAIR, an agent system.	§3.4
G2 : Gensym's rule-based expert system shell: includes powerful interface tools.	§3.5
MBD-based : The model-based diagnosis system used in NASA's remote agent experiment (RAX).	§3.6

Fig. 1. Agent implementation technologies discussed in this article

For several years, the Australian Defense Forces have been using agent-oriented software to assess potential new hardware purchases. Buying planes and helicopters for the Air Force implies a major commitment to a particular platform. AOD uses operational simulation for answering specific questions about very expensive equipment requisitions, component capabilities and rehearsing dangerous tactical operations. In pilot-in-the-loop flight simulation, intelligent pilots (agents) interact with each other in the computer simulation, as well as the human pilot in the virtual environment. These dynamic, interactive multi-agent simulations pose a challenge for the integration of valid pilot competencies into computer controlled agents. This involves modeling pilot perception through recognition of actions and events that occur during simulation. Such simulators are often used after purchase as training tools. Hence, a core task within DSTO is the construction and maintenance of agent-oriented systems. These AOD agent simulations push the state-of-the-art:

– AOD agents interact at high frequency in a dynamic environment with numerous friendly and hostile agents. For example, AOD agents engage in complex aerial maneuvers against hostile high-speed aircraft.
– AOD agents co-ordinate extensively to achieve shared goals. For example, a squadron of fighters may collaborate to shepherd a cargo ship through enemy lines.
– AOD agents may change their roles at runtime. For example, if the lead of a fighter formation is shot down, then the wing-man may assume the role of fighter lead. As roles change, agents must dramatically alter their plans.

After discussions with AOD users, the following concerns were identified. These concerns are the basis for our user-oriented discussion of the merits of different agent technologies:

– Easy of construction/ modification.
– Provable reliability.

– Execution speed.
– Easy of explanation of agent behaviour.
– Support for *teaming*. Teaming is a special AOD requirement which is described below and may not be widely relevant.

The rest of this article debates the merits of the agent implementation technologies of Figure 1 for AOD applications. After that debate, we will see that agent-oriented systems requires much that does not exist in standard software engineering methods:

– AOD agent programmers do not spend their time fretting about encapsulation, classification, etc.
– Instead, they concern themselves with concepts such as plans, communication, perception, intentions, roles, and teams.

This is not to say that standard methods such as (e.g.) object-oriented are irrelevant to agent construction. At the end of this article we will briefly describe how objects are used within AOD agents for building environments in which we specify agent behavior.

2 Agents Versus Objects

Before proceedings, we digress for a brief introduction to agents. In order to simplify the reader's introduction to agents, we will stress the similarities and differences of agent technology to object technology. In what the literature calls *weak agents* maps neatly into object technology. However, object technology is incomplete for implementing what the literature calls *strong agents*.

2.1 Weak Agency

A spectrum of agents types is offered by Woolridge and Jennings [22]. Within that spectrum we say that AOD's agents are at least *weak agents* since they are *autonomous, social, reactive* and *proactive*. They are *autonomous* since, while the simulations execute, AOD agents act within intervention by a human operator. They are are also *social* in the sense that AOD's agents interact extensively with other agents. Also they are *reactive* since AOD analysts develop different scenarios for their agents. Within each scenario, AOD's agents must react to changing circumstances. Further, they are *proactive* since AOD agents can take the initiative within a simulation.

Woolridge and Jennings comment that concurrent object-oriented languages provide much support for weak agents. According to the UML community (e.g. Booch [1]), OO technology consists of at least *identity, state, behavior, classification, inheritance, polymorphism* and *encapsulation*. These terms, and there relations to agents, are described below.

An object is something that can be clearly distinguished from other concepts in the design. Agents can use this *identity* to distinguish themselves from other agents.

Objects are repositories of data. Synonyms for this *state* include attributes or data.

Objects are also repositories of pre-defined *behaviour*. Synonyms for behaviour include methods, functions, procedures, or code. Agents can use state and behaviour to

model their beliefs and actions on the world. Further, identity-specific behaviour lets us implement agent *pro-activeness*; each agent-object can carry with itself an agenda of tasks to be performed.

Each object can be categorized into exactly one class. This *classification* defines the state and behaviour that is valid for that object. Agents can use classification to simplify their reasoning about other objects in the domain. For example, if an agent recognizes that the object `LauraBush` belongs to the `Person` class, then they can access default knowledge about `LauraBush` from `Person`.

Objects can be defined via extensions to parent objects. Services and invariants of such parent objects can be relied on within all the parent's children since parent properties are *inherited*. Agents can use classification to simplify their own internal implementation as well as extending their classification knowledge.

Behaviors can be implemented in different objects using the same name. This *polymorphism* implies some kind of message-passing system; i.e. the results of a particular message is computed from the name of message and the nature of the receiver of that message. The same message sent polymorphically to different objects may generate different responses. Agents can use polymorphism to group together the services offered by other agents. For example, an agent might assume that all other agents in their domain respond to messages such as "position" or "velocity". Polymorphism means agents can assume simple and common interfaces to other agents; i.e. polymorphism simplifies the implementation of agent *social ability*.

Clients of an object should not need access to the internal details of that object. Such clients can ask an object to perform a named service without needing to know the details of how such services are implemented. Agents can use this *encapsulation* to ensure that, when they run *autonomous*ly and asynchronously, their internal knowledge is not mixed up with the knowledge of other agents.

2.2 Strong Agency

According to Wooldridge and Jennings, *strong agents* may possess mentalistic attitudes or be emotional or animated. AOD agents lack emotion but are mentalistic. AOD agents are based on the *beliefs*, *desires* and *intentions* (BDI) paradigm of [17]. The current state of entities and environment as perceived by the agent (abstract labels or percepts) are the agent's *beliefs*. A *desire* is some future states agent would like to be in (a.k.a. goals). *Intentions* are some commitment of an agent to achieve a goal by progressing along a particular future path that lead to the goal (a.k.a. a plan). One advantage with using intentions is that the effort associated with creating them need not be repeated every time they are required. Intentions can be pre-computed and cached. Each intention can be tagged with a trigger describing some situation in which this intention should be accessed and applied.

In this BDI paradigm, deliberation is done through the selection of a goal, selection of a plan that will be used to form an intention, selection of an intention, and execution of the selected intention. All these decisions are based on the beliefs the agent has about the current state of the environment. The process of selecting the forming plan is known as means-end reasoning.

The BDI means-end reasoning approach has proven very beneficial to the AOD, particularly the OPERATOR AGENT that is used to simulate military operations, it is used in support of multi-billion dollar requisitions [20]. This is partly reflected in the maturity of BDI, and the fundamental limitations it overcomes [17]. This includes ways of relaxing the need to have perfect knowledge of opponents' plans.

The BDI approach has made it easier for DSTO to codify tactics and build simulations, and importantly, to get explanations out of them of what happens. The DSTO has found it easier to construct and modifying scenarios, as the Operator Agent is a generalized agent capable of simulating different entities, aircraft, ships etc. Implemented using the dMARS procedural reasoning language [4]), BDI improves abstraction for representation of declarative and procedural knowledge. This allows for improved explanation, as the success of these operational simulations is measured in terms of what all stakeholders receive, whether experts or operators.

The BDI approach has also scaled up for large scale operational simulation-the Operator Agent at the Air Operations Division (AOD) has been used to run operational simulations involving 32 pilots, 400 plans, 10 goals, 25 intentions concurrently, each receiving 8 contacts every 50ms [20].

3 A Range of Agent Technologies

In this section, we explore a range of agent implementation technologies considered at AOD. All our implementation options are compared to some mythical ideal AOD agent system in the *repertory grid* of Figure 2. Figure 2 shows various implementation options (e.g. G2) ranked on different dimensions (e.g. easy/hard to modify). Repertory grids are a knowledge acquisition technique extensively explored by Shaw and Gaines [19, 5]. Their key benefit is that superfluous comparisons between examples are ignored. Dimensions are only added to a repertory grid if they help to distinguish examples. That is, dimensions with no information content are excluded. A side-effect of this approach will be that we will not define in detail different agent implementation technologies. Instead, we will just discuss how these technologies differ according the repertory grid dimensions. However, we do provide references for each method so the interested reader can explore further.

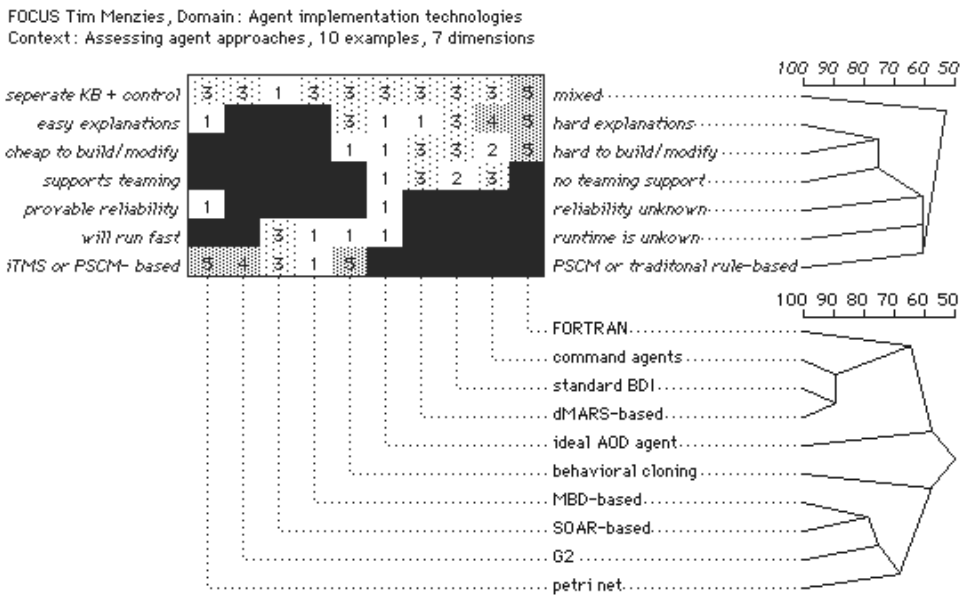
The rest of this section defines the dimensions of Figure 2. The desired end of each dimensions will be shown as headings while the opposite, and undesirable, end of each dimension will be shown in italics.

3.1 Easy to Build/Modify

Opposite= hard to build/modify.

AOD's agents are used to handle specialized what-if queries. What-if queries are notoriously ad hoc. Ideally, we should be able to quickly change our agents to accommodate new what-if queries.

Originally, AOD ran its agent simulations using standard procedural languages to implement the decision logic of a human operator. In procedural systems (e.g. FORTRAN) such decision logic may comprise less than 20% of the total system. However,



A repertory grid maps examples *into a range of* dimensions*. For any row in the table, if an example scores "5", it is near the right-hand end of a dimension. If it scores "1", it is near the left-hand end of a dimension. For example, bottom left, petri nets scores a "5"; i.e. it is more a rule-based system than a iTMS or PSCM-based system. Black areas denote "don't know". Trees on far right show distances between each example and dimension; e.g. standard BDI and dMARS-based systems are very similar.*

Fig. 2. A comparison of implementation issues for agents. Generated from the web-based repertory grid server at `http://gigi.cpsc.ucalgary.ca/`.

such code can consume up to 80% of the maintenance effort. For example, new programmers working on a FORTRAN simulator could take up to one year before be able to modify the decision logic.

In response to this maintenance cost, AOD turned to high-leval BDI-based agent-oriented languages. Agent-oriented software such as dMARS [4] offers a succinct and high-level representation of the decision logic. Using such agents, AOD has been very successful in reacting to customer "what if" queries from their customers. With the FORTRAN systems, in might have taken weeks before AOD could report simulation results from different operator actions. With the agent systems, such reports can be generated in days or even hours.

Standard BDI is a conceptual framework that must be supported by a tool such as dMARS in order to be practical for real-world practioners. dMARS offers high-level support for BDI and extensive graphical facilities. dMARS reduced the cost of creating AOD agents. However, even though costs were significantly reduced, AOD still finds

agent design to be very expensive. Two technologies that address the cost issue are *command agents* and *behavioral cloning of agents*, discussed below.

Command Agents Pearce and Heinze added another layer on top of dMARS to process the patterns of standard agent usage seen at AOD [14, 7]. Their *command agents* divide reasoning into the following . Firstly, *situation awareness* extracts the essential features from the environment. Next, the *assessment* layer ranks the extracted features. These ranking represents a space of options explored by *tactic selection:*. lastly, in the *selection of operational procedure* layer, the preferred option is mapped onto the available resources.

In repeated applications of this framework, Pearce and Heinze report that the command agents framework offers a significant productivity increase for AOD personnel over standard dMARS.

Behavioral Cloning of Agents *Behavioral cloning* is a machine learning technique. A qualitative model of a domain is generated. The qualitative model is imprecise and so contains areas of uncertainty. A scenario is described. This scenario serves to restrict the ranges of some of the uncertain values within the model. The model is now executed within the constraints offered by the scenario. Where uncertainty exists, the simulator either picks one option at random or backtracks over all options (if such backtracking is tractable). This simulation generates an experience base: a file of examples of the operation of the model. The experience base is then classified according to some domain criteria; e.g. we never ran low on fuel. A machine learner then generates some summary of the classified examples. This technique has been used to generate knowledge of cardiac function [2], satellite electrical systems [15], how to fly a plan [18], and how to run a software project [11].

Behavioral cloning often throws the intermediaries between inputs and outputs. Hence, a disadvantage of behavioral cloning is that the learnt theory can't queried for a rich description for *how* some output was reached. Hence, behavioral clones may be hard to explain.

On the other hand, behavioral cloning scores highly on the "easy to build/modify" scale. Once the background qualitative model has been built, the knowledge required for a new agent can be generated automatically merely by adopting new scenario constraints.

Easy to Modify: Summary Clearly FORTRAN scores the lowest on this dimension and behavioral cloning, theoretically anyway, scores the highest. dMARS is mid-range on this scale and command agents are easier to modify than standard dMARS systems.

3.2 Supports Teaming

Opposite= no teaming support

Teams are a natural means of modeling collaborating defense forces. Teams act as if they share a belief set even though sometimes team members may have subtlety different beliefs. While team member A may not know everything known by team member B, some abstract co-ordination agent knows everything that the team members know.

The current generation of agent software used at AOD is designed for modeling the interaction of solo agents in a shared environment. Coding such tools gets complex when teams co-ordinate and share beliefs. In particular, every exception case where the team believes X while the team member believes Y must be encoded.

At present, AOD makes little use of team-based agent simulations: i.e. most of its systems are at the first-person singular level. Consequently, the spaghetti sections are not large. However, it is anticipated that in the near future, AOD will be making extensive use of team-based simulations.

Some initial experiments have been performed with implementing teaming in a dMARS framework [21]. These initial studies may also supply command agents with a teaming ability (since command agents are built on top of dMARS). Apart from that, the AOD experience is that most agent systems have very little support for teaming.

3.3 Provable reliability

Opposite= reliability unknown.

Currently, AOD's agents are only used in the research labs. However, this may change. Each simulator stores extensive tactical knowledge that could be usefully applied in the field. However, before a research prototype can be deployed into the field, its reliability must be certified. In the context of this article we say that reliability is some probability that, after watching the system run and seeing some errors, that in for future time T we will see N errors.

Within AOD there has been some discussions on using petri nets to implement agents. A petri net [16] is a directed graph interconnecting nodes of different types on which multiple tokens are permitted to travel. There are two types of nodes: places and transitions. Arcs connect places to transitions and transitions to places; transitions are never connected to transitions, and places are never connected to places. While Petri nets consist of a small number of elements and the algorithms for evaluation can be expressed very simply, they are sufficiently formal to allow mathematical analysis. These tend to be both detailed and complex. A large suite of reliability results exist for petri nets (as used in Markov chains [10, p759-765]) or otherwise [9]. Agents based on petri-nets/markov models have been used in domains as complicated as Robocup[6] (though Pearce, Heinze and Goss comment that such single state systems do not adapt well when there are multiple entities operating in the simulation [14, 7]).

Petri nets are a convenient tool for reasoning about concurrent systems. Unlike a state transition diagram, where execution is sequential, traversing one state after another, with petri nets execution is fully concurrent. Hence, it has been argued that petri nets are a sound basis for constructing reliable agents.

On balance, it is doubtful that AOD will adpot the petri net approach, due to the next point.

3.4 Separate KB and Control

Opposite= Mixed.

AOD has a strong commitment to cognitive modeling; i.e. generating human-like behaviour from explicit high-level symbolic representations. Hence, an ideal AOD agent uses such representations.

Different approaches to agent-based systems take a different approach to their representations. General BDI is a conceptual framework, not a specific implementation. Hence, it makes no commitment to levels of control flexibility. Procedural languages like FORTRAN mix domain knowledge and inference knowledge like spaghetti. Recall that it was the maintenance effort associated with such spaghetti code that drove AOD away from FORTRAN. Knowledge based systems take a separate approach: inference and domain knowledge are distinct and can be modified separately. All the non-FORTRAN agent systems being discussed here can be divided up according to how flexible is their control knowledge.

At one end of the control flexibility-scale are agents based on the SOAR architecture (e.g. TACAIR [8]). A SOAR knowledge base has two distinct parts: a standard declarative rule section and a second section describing the control knowledge in rule-based terms. In SOAR, agents seek operators which might take them closer to their goals. Conflict resolution is used to resolve conflicts between competing operators. Conflict resolution is SOAR, the operators, and the domain rule knowledge are all implemented in the same uniform rule-based manner. This means that the knowledge engineer can customize not only the rule base but also the control of how the rules are fired. Rules in SOAR are organized into problem spaces: zones within the kb that discuss the same sets of operators. If a SOAR problem space can't resolve operator clashes, then a *impasse* is declared and SOAR forks a nested problem space with the task of resolving that conflict. Hence, at runtime, a SOAR system conducts a recursive descent through the problem spaces. This is called the problem space computation model (or PSCM [23]). Recursive descent is only an approximation of PSCM. Each problem space is an autonomous group of rules that might "wake-up" and execute if their pre-conditions are ever satisfied. The authors of TACAIR use this feature extensively. TACAIR views its knowledge as an intricate goal hierarchy. Problems spaces act like min-agents; each with their own autonomous agenda that sleep till some trigger condition makes them execute.

At the other end of the control flexibility-scale are petri nets and the decision trees generated by behavioral cloning. Petri nets and decision trees permit no meta-level control of the inference net. In marked contrast to SOAR, inference is via inflexible local propagation rules. It would be a simple matter to add a meta-interpreter to (e.g.) petri nets that could customize the propagation rules. However, such meta-level control would change the runtime properties of a petri net and invalidate all the petri net reliability results.

Halfway between the total flexibility of SOAR and the rigid inflexibility of petri nets/decision trees lies the BDI support systems (dMARS, command agents). These system restrict the knowledge engineer into creating planning systems. However, within that restriction, considerable flexibility is offered.

3.5 Easy explanations

Opposite= hard explanations

Defense personnel audit AOD agent knowledge bases to see if they reflects current Australian defense tactical maneuvers. Hence, it is important that the inner workings of agents can be explained to a domain expert.

Explaining the behaviour of a agent generated from behavioral cloning is difficult. The learnt theory is a mapping of inputs to outputs with all internal connections thrown away. Such internal connections are useful in building rich explanation structures. Still, the simplicity of the decision tree generated from behavioral cloning makes cloned agents easier to understand than those written in some procedural language (e.g. FORTRAN).

AOD supports at least two methods for generating intricate explanations. Firstly, they claim that the agents knowledge base is close in concept to how human operators control defense hardware. At AOD, agency is not merely some implementation trick. Rather, by explicating agent knowledge, AOD is hoping that it is also explicating human knowledge at a comprehensible level of abstraction. Further, the command agents paradigm sits well with standard operational practice within the Australian defense forces. Hence, AOD argues, explaining an BDI/command agents tool is fundamentally simpler than browsing some other conceptual model (e.g. FORTRAN code).

Secondly, AOD uses explanations via graphical visualizations. The dMARS tool makes extensive use of graphical interfaces. Another implementation tool of note is the G2 system. Clancey *et.al* have used G2 to build elaborate agent systems in which the behaviour of an organization is emergent from the behaviour of the agents representing the workers in that environment [3]. G2 offers extensive support for a traditional rule-based execution paradigm. G2 is also a powerful tool for building elaborate graphical interfaces (e.g. Figure 3).

3.6 Tractability/speed results known

Opposite= tractability/ speed is an open issue.

Repeating the above point on reliability, if a research prototype is to be deployed into the field, we need some assurance that it will execute at least as fast as events in the field.

Behavioral cloning builds decisions trees and such trees execute very quickly at runtime. While this is a significant advantage of behavioral cloning, we note that this comes at the cost of all the disadvantages mentioned above; e.g. poor explanatory capacity.

In terms of reasoning efficiency, an interesting middle ground between decision trees and BDI is the agent technology used by NASA in its remote agent experiment (RAX). NASA deployed RAX in the asteroid belt in May 1999. RAX built its own autonomous plans to react to flight events [12]. The representations within RAX were much simpler than (e.g.) the AOD BDI plans. The RAX development team claim that, after several years with exploring model-based diagnosis (MBD), that very simple representations can model even complex domains. Further, given regular propositional encodings of that domain knowledge, an incremental truth maintenance system (iTMS) [13] can build satisfactory plans very quickly.

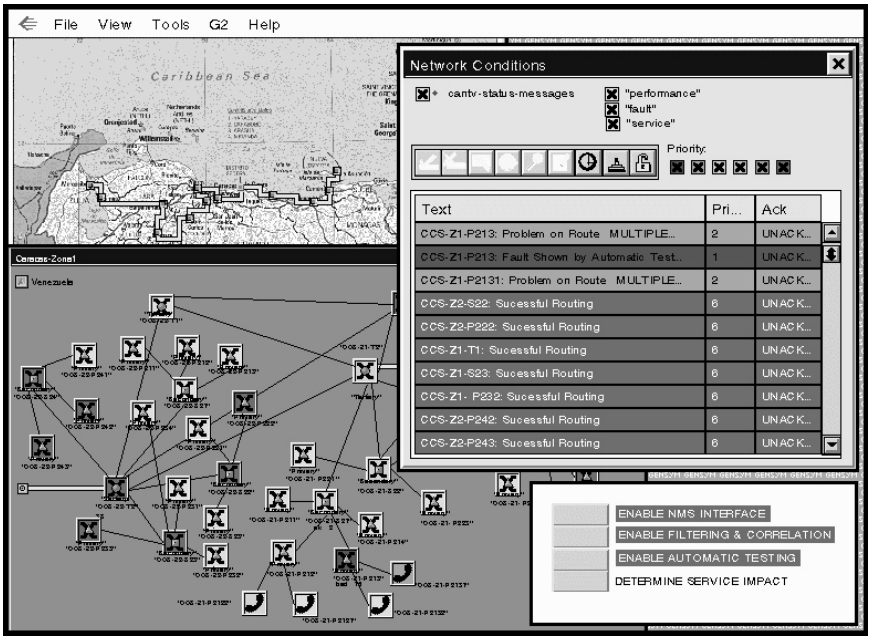
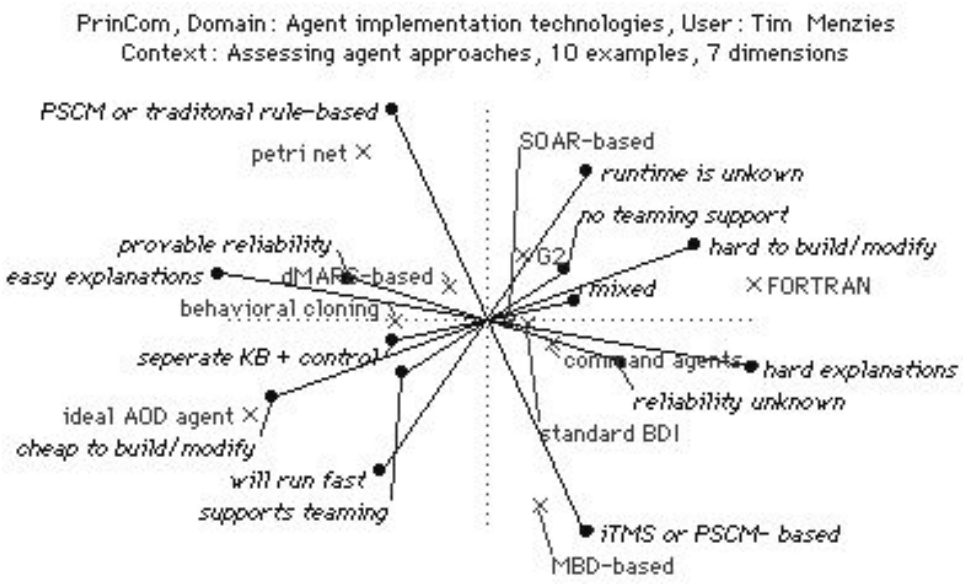


Fig. 3. Sample screensap from a G2 application. From `http://www.gensym.com/products/operationsexpert.htm`.

4 Conclusion

The goal of this paper was to assess agent-oriented systems via two kinds of user-oriented queries. Firstly, our users were worried about the (non)-distinction between agents and objects. As seen above, the "OO mindset" contains about half a dozen key concepts: class, instance, encapsulation, inheritance, polymorphism, and so on. However, the key concepts of the AOD "agent-oriented" mindset make little reference to the key OO concepts. For example, OO makes no reference to the concept that AOD regards as central to agency: intentionality. Nor do the core OO concepts refer to explicit knowledge representation of domain and control knowledge, teaming, plans, communication, perception, beliefs, and desires. Further, the assessment criteria we would apply to agent systems are orthogonal to the key OO concepts and include ease of construction and maintenance, ability to explain agent knowledge, and promises of runtime performance, reliability.

Secondly, our users were concerned about the range of issues and the range of technologies clustered in Figure 4. This figure is a 2-D display of a n-dimensional space and so cannot be a totally accurate representation. However, it does clearly show the distance of FORTRAN to AOD's ideal agent technology. Further, the low-level representations of petri nets and the MBD-based RAX system make them poles apart from AOD's ideal agent. Behavioral cloning is surprisingly close to AOD's ideal; perhaps only because of its ease of construction.



This figure is generated from Figure 2 via a principle components analysis to grid the data. 2-D positions are generated via first isolating combination of factors that are strongly correlated. These factors define a transform space into which the various technologies described in this article can be mapped into a 2-D space. This diagram can be used to gain an intuitive view of what technologies are similar (they appear closer together on this diagram).

Fig. 4. Clusters of agent technologies. Generated from the web-based repertory grid server at `http://gigi.cpsc.ucalgary.ca/` using the dimensions of Figure 2.

Apart from Figure 4, we have also seen that different design goals select different agent implementation technologies:

- If the goal is human cognitive modeling, then avoid low-level representations. High-level representation schemes mentioned here were SOAR and the BDI-based tools.
- If committed to intentionality, then use BDI-based tools such as dMARS or the command agents extension.
- If the goal is raw execution speed, use low-level representations plus a fast theorem prover. Two such candidates are the decision trees from behavioral cloning or propositional encoding with the iTMS.
- If the goal is explanation/justification of system behaviour, use some technology that offers high level views of the system. Two such systems reviewed here are the graphical interfaces of G2 or systems that use high-level representations that match local domain knowledge (e.g. command agents).
- If the goal is highly-reliable systems, then use petri nets.

– If the goal is to model elaborate teaming interactions, then hire AI specialists and conduct a research program to extend the state-of-the-art in agent systems.
– If the goal is to reduce maintenance costs, then consider automatically generating the agent using behavioral cloning. Alternatively, if the goal is to increase revenue through maintenance requests, then use FORTRAN.

Acknowledgements

The contents of this paper were greatly enhanced by the lively discussion at the Australian workshop on agent-oriented systems organized by Leon Sterling (Melbourne University) and Simon Goss (AOD). All views expressed here are the authors and may not reflect the consensus view of that workshop.

This research was partially conducted at West Virginia University under NASA contract NCC2-0979. The work was sponsored by the NASA Office of Safety and Mission Assurance under the Software Assurance Research Program led by the NASA IV&V Facility. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not constitute or imply its endorsement by the United States Government.

References

1. G. Booch. *Object-Oriented Design with Applications (second edition)*. Benjamin/ Cummings, 1994.
2. I. Bratko, I. Mozetic, and N. Lavrac. *KARDIO: a Study in Deep and Qualitative Knowledge for Expert Systems*. MIT Press, 1989.
3. W. Clancey, P. Sachs, M. Sierhuis, and R. van Hoof. Brahms: Simulating practice for work systems design. In P. Compton, R. Mizoguchi, H. Motoda, and T. Menzies, editors, *Proceedings PKAW '96: Pacific Knowledge Acquisition Workshop*. Department of Artificial Intelligence, 1996.
4. M. d'Inverno, K. M., L. D., and M. Wooldridge. A formal specification of dmars. In A. Singh and M.Wooldridge, editors, *Intelligent Agents IV: Proc. of the Fourth International Workshop on Agent Theories. Architectures and Languages, Springer Verlag*, 1998.
5. B. Gaines and M. Shaw. Comparing the conceptual systems of experts. In *IJCAI '89*, pages 633–638, 1989.
6. K. Han and M. Veloso. Automated robot behaviour recognition applied to robot soccer. In *Proceedings of the Sixteenth Interntional Joint Conference on Artificial Intelligence. Workshop on Team Behaviour and Plan Recognition*, pages 47–52, 1999.
7. C. Heinze, S. Goss, T. Josefsson, K. Bennett, S. Waugh, I. Lloyd, G. Murray, and J. Oldfeild. Interchanging agents and humans in military simulation. *AI Magazine*, 23(2), Summer 2002.
8. R. M. Jones, J. E. Laird, P. E. Nielsen, K. J. Coulter, P. G. Kenny, and F. V. Koss. Automated intelligent pilots for combat flight simulation. *AI Magazine*, 20(1):27–41, 1999.
9. O. Kummer. The petri nets bibliography; keyword: reliability. `http://www.informatik.uni-hamburg.de/TGI/pnbib/keywords/r/reliability.h%tml`, 2000.
10. M. Lyu. *The Handbook of Software Reliability Engineering*. McGraw-Hill, 1996.

11. T. Menzies, E. Sinsel, and T. Kurtz. Learning to reduce risks with cocomo-ii. In *Workshop on Intelligent Software Engineering, an ICSE workshop, and NASA/WVU Software Research Lab, Fairmont, WV, Tech report # NASA-IVV-99-027*, 2000. Available from `http://menzies.us/pdf/00wise.pdf`.
12. N. Muscettola, P. P. Nayak, B. Pell, and B. Williams. Remote agent: To boldly go where no ai system has gone before. *Artificial Intelligence*, 103(1-2):5–48, August 1998.
13. P. P. Nayak and B. C. Williams. Fast context switching in real-time propositional reasoning. In *Proceedings of AAAI-97*, 1997. Available from `http://ack.arc.nasa.gov:80/ic/projects/mba/papers/aaai97.ps`.
14. A. Pearce, C. Heinz, and S. Goss. Meeting plan recognition requirements for real-time air-mission simulations, 2000.
15. D. Pearce. The induction of fault diagnosis systems from qualitative models. In *Proc. AAAI-88*, 1988.
16. W. Reisig. *Petri Nets*. Springer Verlag, 1982.
17. A. Roa and M. Georgeff. Bdi agents: From theory to practice. In *Proceedings of the First International Conference on Multi-Agent Systems, San Francisco, CA, June*, 1995.
18. C. Sammut, S. Hurst, D. Kedzier, and D. Michie. Learning to fly. In D. Sleeman, editor, *Ninth International Conference on Machine Learning*, pages 385–393. Morgan Kaufmann, 1992.
19. M. Shaw. *WebGrid: a WWW PCP Server*. Knowledge Systems Institute, University of Calgary, `http://Tiger.cpsc.ucalgary.ca/WebGrid/WebGrid.html`, 1997.
20. G. Tidar, C. Heinze, S. Goss, G. Murray, D. Appla, and I. Lloyd. Using intelligent agents in military simulations or 'using agents intelligently.'. In *Proc. of Eleventh Innovative Applications of Artificial Intelligence Conference, American Association of Artificial Intelligence*, 1999.
21. G. Tidhar, C. Heinze, and M. Selvestrel. Flying together: Modelling air mission teams. *Applied Intelligence*, 8(3):195–218, 1998.
22. M. Wooldridge and N. Jennings. Intelligent agents: Theory and practice. *The Knowledge Engineering Review*, 10(2):115–152, 1995. Available from `http://www.cs.umbc.edu/agents/introduction/ker.ps.Z`.
23. G. Yost and A. Newell. A Problem Space Approach to Expert System Specification. In *IJCAI '89*, pages 621–627, 1989.