

Model Checking Agent Programming Languages*

Louise A. Dennis* Rafael H. Bordini[†] Berndt Farwer[†] Michael Fisher*

*Department of Computer Science, University of Liverpool, UK [†]Department of Computer Science, Durham University, UK

1 Introduction

The last decade has seen significant growth in both the amount and maturity of research being carried out in the area of *agent-based systems*. An agent can be seen as an *autonomous* computational entity, making its own decisions about what activities to pursue. *Rational agents* make such decisions in a rational and explainable way and, since agents are autonomous, understanding *why* an agent chooses a course of action is vital. Therefore, a key new aspect in the design and analysis of such systems is the need to consider not just what agents do but *why* they do it.

While program verification is well advanced, for example Java verification using Java PathFinder [6; 9], verification of agent-oriented programs poses new challenges that have not been adequately addressed, particularly in the context of practical model-checking tools. In agent verification, we have to verify not only what the agent does, but *why* it chose that course of action, *what* the agent believed that made it choose to act in this way, and what its intentions were in doing so.

Rather than providing an approach for *one* particular programming language, we here describe an architecture for a system allowing the verification of a wide range of agent-based programs, produced using various high-level agent-oriented programming languages. Our previous work [1; 2] has concentrated on model checking techniques for agent-based systems written in the logic-based agent-programming language AgentSpeak [8]. As described above, it is vital to verify not only the behaviour that the agent systems has, but to verify *why* the agents are undertaking certain courses of action. Thus, the temporal basis of model-checking captures the *dynamic* nature of agent computation, but is extended with *intensional modal operators* capturing the *informational* ('beliefs'), *motivational* ('desires') and *deliberative* ('intentions') aspects of a rational agent. Such pioneering work on *model checking* techniques for the verification of agent-based systems often based on standard model checkers such as Spin or JPF, has appeared, for example, in [1; 2; 7].

2 Architecture of the AIL

One of the problems with existing approaches to model checking agent programs is that one had to find ways of encoding beliefs, goals, etc., within the state of the JPF or Spin state machine. This is a complex task, and one that would need to be (at least partly) done again to allow model checking for other agent programming languages. As in other fields of Computer Science, with the emergence of various modelling formalisms (in particular here new agent programming languages), a need for a unifying framework arises.

We have investigated the key aspects underlying several BDI (Belief/Desire/Intention) programming languages [4]. Based on that, we have developed the Agent Infrastructure Layer (AIL) [5; 3], a collection of Java classes that: (i) enables implementation of interpreters for various agent languages, (ii) contains adaptable, clear semantics, and (iii) can be verified through AJPF, an extended version of the open source Java model checker JPF [9]. AJPF is a customisation of JPF that was optimised for AIL-based interpreters.

The AIL can be viewed as a platform on which agents programmed in different programming languages co-exist, and together with AJPF this provides uniform model checking techniques for various agent-oriented programming languages. This is further extended with the *MCAPL*¹ *interface* which allows programming languages that do not have their own AIL-based interpreters to be model checked against specifications written in the same property specification language. (However, these will not benefit from the efficiency improvements that the AJPF customisations provide.) Figure 1 provides a diagrammatic representation of the AIL architecture.

The AIL is not intended as a new language in its own right, but as an intermediate layer incorporating the main features of various existing (practical) agent languages. We have identified the key *operations* that many (BDI-)languages use and treat these operations as part of an *AIL toolkit*. The semantic rules in [4] are a part of this toolkit but any given language can use a selection of these rules in its own AIL-based interpreter and may choose to add its own custom rules built from the basic operations made available. These operations and rules have formal semantics and are implemented in Java.

*Work supported by EPSRC grants EP/D054788 (Durham) and EP/D052548 (Liverpool).

¹*Model Checking Agent Programming Languages*.

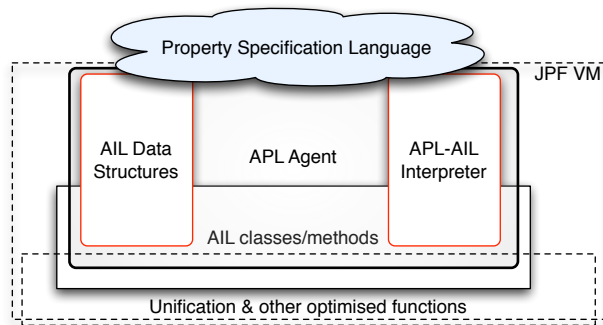


Figure 1: Overview of the AIL Architecture.

We assume that agents in any target agent programming language all possess a *reasoning cycle* consisting of several (≥ 1) stages. Each stage is a disjunction of rules that define how an agent's state may change during the execution of that stage. The combined rules of the stages of the reasoning cycle define the operational semantics of that language. The construction of an interpreter for a language involves the implementation of these rules (which in some cases might simply make reference to the pre-implemented rules) and a reasoning cycle. This means that the AIL can be viewed as a collection of Java classes/methods that provide the building blocks for custom programming of agent language interpreters, with the particular advantage of making model checking possible (and more efficient). In this way, we can implement, for example, an AgentSpeak interpreter following the AgentSpeak operational semantics but using the AIL operations rather than using Java from scratch.

Common to all language interpreters implemented using AIL methods are the AIL-agent data structures for beliefs, intentions, goals, etc., which are accessed by the model checker and on which the modalities of the property specification language are defined. The implicit data structures of a given BDI language need to be translated into the AIL's data structures. In particular, the initial state of an agent has to be translated into an AIL agent state.

In addition to the AIL toolkit, we also provide a set of Java interfaces that we call the *MCAPL interface*. As mentioned earlier, this allows agents to be model checked using the same property specification language even if no AIL-based interpreter for that language has been developed. An implementation of the MCAPL interface must define the required operators of the property specification language. For instance, agents implementing the MCAPL agent interface must provide a method which succeeds when the agents believes that the given parameter (represented as a "formula") is true. In other words, the implementation of such a method effectively corresponds to the semantics for the *belief* modality in that specific language. The AIL implements these interfaces and so defines an AIL-specific semantics for the property specification language; supported languages that use the AIL must ensure that they do so in a way that is consistent with their own semantics of those modalities.

References

- [1] R. H. Bordini, M. Fisher, W. Visser, and M. Wooldridge. Model Checking Rational Agents. *IEEE Intelligent Systems*, 19(5):46–52, 2004.
- [2] R. H. Bordini, M. Fisher, W. Visser, and M. Wooldridge. Verifying Multi-Agent Programs by Model Checking. *J. Autonomous Agents and Multi-Agent Systems*, 12(2):239–256, 2006.
- [3] R. H. Bordini, L. A. Dennis, B. Farwer, and M. Fisher. Automated Verification of Multi-Agent Programs. In A. Ireland and W. Visser, eds., *Proc. 23rd IEEE/ACM International Conference on Automated Software Engineering (ASE'08)*. 2008, To Appear.
- [4] L. A. Dennis, B. Farwer, R. H. Bordini, M. Fisher, and M. Wooldridge. A Common Semantic Basis for BDI Languages. In *Proc. 7th Int. Workshop on Programming Multiagent Systems (ProMAS)*, 2007.
- [5] L. A. Dennis, B. Farwer, R. H. Bordini, and M. Fisher. A flexible framework for verifying agent programs (short paper). In Padgham, Parkes, Muller, and Parsons, eds., *Proc. of the 7th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2008)*. pp. 1303–1306, ACM, 2008.
- [6] <http://javapathfinder.sourceforge.net>.
- [7] F. Raimondi and A. Lomuscio. Automatic Verification of Multi-agent Systems by Model Checking Ordered Binary Decision Diagrams. *J. Applied Logic*, 5(2):235–251, 2007.
- [8] A. Rao. AgentSpeak(L): BDI Agents Speak Out in a Logical Computable Language. In *Proc. 7th European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW)*, volume 1038 of LNCS, pages 42–55. Springer, 1996.
- [9] W. Visser, K. Havelund, G. P. Brat, S. Park, and F. Lerda. Model Checking Programs. *Automated Software Engineering*, 10(2):203–232, 2003.x