Formal Evaluation of Agent Architectures

N. Alechina and B. S. Logan
School of Computer Science
University of Nottingham
Nottingham NG8 1BB, UK
{nza,bsl}@cs.nott.ac.uk

Abstract

In this position paper, we present a methodology for the formal evaluation of agent architectures. We argue that any agent architecture can be formally represented by a set of state transition systems. We then show how both qualitative and quantitative properties of an architecture can be evaluated using logical formulas interpreted in this set of state transition systems. In addition, we also provide a precise notion of what it means for an agent described in implementation specific terms to have or implement a given architecture.

Introduction

An architecture can be thought of as specifying both a set of concepts which can be used to talk about an intelligent system (e.g., a robot or an intelligent agent) and a set of (usually high-level) capabilities realised by a system. For example, if we say that a system has "beliefs", we are saying something about its internal representations of states of affairs. If we say that it can "plan", we mean that it is capable of generating a representation of a sequence of future actions, which, if executed in a state of affairs in which its beliefs hold, will achieve its goal(s). However, while such informal high-level descriptions of a system are useful, they can be hard to pin down when we are trying to determine if a given intelligent system has a particular property as a consequence of its architecture. There are many variants of "Belief-Desire-Intention" (BDI) architectures in the agents literature for example. This lack of precision makes it hard to compare or evaluate architectures or to say that a given system has a particular property as a consequence of it having a particular architecture.

In this position paper, we propose an approach to architectures and their evaluation which both makes precise exactly what it means for a system to 'have' an architecture, and allows us to establish properties of a system expressed in terms of its architectural description. In what follows, we focus on architectures of software agents, however a similar approach can be applied to robots and other kinds of intelligent systems. We show how to establish a precise formal relationship between an evaluation criterion, an agent architecture and an agent model (taken to be an implementation-level description of an agent). A consequence of our approach is that it is possible to verify whether two different agent programs really have the same architecture or the same properties, allowing a more precise comparison of architectures and the agent programs which implement them. We illustrate our approach with a number of examples which show how to establish both qualitative and quantitative properties of architectures and agent programs, and how to establish whether a particular agent has a particular architecture.

Specifying Architectures and Properties

An architecture is a way of looking at or conceptualising an agent program which defines the possible states of the agent and transitions between them. We can think of an architecture as providing a language of concepts and relations for describing the contents of agent states and their possible transitions. Different architectures characterise the contents of the agent's state in different ways, e.g., as beliefs or goals or 'affective' states such as 'hungry' or 'fearful' or simply in terms of the contents of particular memory locations. Similarly, the possible transitions may be characterised in terms of basic agent capabilities, such as performing a 'basic action', or by higher-level capabilities such as 'perception', 'learning' or 'planning'. Typically some of the contents of the states and some of the possible transitions will be under the control of the agent developer. For example, an agent programming language or toolkit may allow the representation of beliefs about any state of affairs, whereas a particular agent implementation may only have beliefs about customer orders for a given range of goods. Those transitions which are not under the control of the developer typically correspond to the execution of some 'basic' control cycle specified by the architecture. For example, a particular architecture may specify that the execution of a basic action or plan step is always followed by a 'sensing' step.

In this view, the 'pure' architecture of an agent consists of a language for describing its states and transitions, and a set of constraints which limit the possible states and the transitions between them to those allowed by the architecture. For example, a possible constraint on the agent's beliefs imposed by the architecture may be that beliefs are always consistent, or that they always include, e.g., a belief about the current time or the agent's battery level.

Since instances of an architecture are computer programs, they can be represented as state transition systems. The states of the state transition system will be determined by

the possible states defined by the architecture. Similarly, the transitions between states are determined by the kinds of transitions supported by the architecture. The set of all possible transition systems corresponding to instances of an architecture provides a formal model of the architecture itself. Note that we assume that this set is not given extensionally, but is defined by a set of constraints on possible states and possible transitions between states.

The key idea underlying our approach is to define a logic which axiomatises the set of transition systems corresponding to the architecture of interest. This logic can then be used to state properties of the architecture (typically couched in terms of the concepts defined by the architecture). An architecture has a particular property if a formula in the logic expressing this property is true in all transition systems defined by the architecture. For example, we can formulate precisely in a suitable logic (e.g., CTL* or PDL extended with for example belief and goal modalities) various *qualitative* properties of architectures such as, 'Can an agent have inconsistent beliefs?', 'Can an agent drop a goal?' or 'Can an agent drop a goal which is not achieved?', and check which of these properties are satisfied by which architectures. Perhaps more surprisingly, we can also use this approach to evaluate various *quantitative* properties of architectures. For example, 'how much time would choosing an action take in this agent architecture?', or 'how much memory is required to perform deliberation?'. Note that by appropriate choice of the logic, we can perform an evaluation at different levels of detail. For example we can establish properties of the 'pure' architecture, or of a particular agent program, or a particular agent program with particular inputs.

There exists a substantial amount of work on modelling agent systems using state transition systems and expressing correctness properties in logic (for example, (Rao & Georgeff 1991; Fagin *et al.* 1995; van der Hoek, van Linder, & Meyer 1999; Wooldridge 2000)). The major difference between this work and our approach, is that we use syntactic belief, goal etc. modalities rather than a standard Kripke-style semantics to model agent beliefs and other propositional attitudes, thus avoiding the problem of logical omniscience (that the agents are modelled as believing all logical consequences of their beliefs) (Hintikka 1962). Since reasoning takes time and memory in any implemented agent system, formal models based on possible worlds semantics may incorrectly ascribe properties to real agent systems.

Evaluating Correctness Properties

In the remainder of this paper we illustrate our approach by means of some simple examples drawn from our previous work. In this section we briefly survey some previous work in (Alechina *et al.* 2007), in which we proposed a sound and complete logic for SimpleAPL, an APL-like (Dastani *et al.* 2004; Bordini *et al.* 2005) agent programming language. The logic is a variant of Propositional Dynamic Logic (PDL) (Fischer & Ladner 1979), which allows the specification of safety and liveness properties of SimpleAPL agent programs. In (Alechina *et al.* 2007) we showed how to define transition systems corresponding to different program execution strategies in that logic. Here, we first briefly describe the language of SimpleAPL and two possible execution strategies (atomic and interleaved) and then sketch how to formalise them in logic.

SimpleAPL is a fragment of 3APL (Dastani *et al.* 2004; Bordini *et al.* 2005). A SimpleAPL agent has a set of goals (ground literals), beliefs (ground literals), plans, and planning goal rules. A planning goal rule of the form

$$\psi \leftarrow \phi | \pi$$

can be read as "if you have a goal ψ and you believe ϕ, then adopt plan π". SimpleAPL plans are built from basic actions using sequential composition, conditionals and iteration. Basic actions have a finite set of pre- and postconditions. For simplicity, to avoid modelling the environment, we assume that the agent's beliefs are always correct and actions always successful, which allows us to express pre- and postconditions in terms of the agent's beliefs.

An important choice in defining the architecture of a SimpleAPL agent is to decide whether the agent has an atomic or an interleaved plan execution strategy. By an atomic plan execution strategy we mean the following two conditions:

- the agent has a single plan at any given time; and
- the agent only applies a planning goal rule after it has completely executed its current plan.

By an interleaved execution strategy we mean:

- the agent may have several active plans at any given time; and
- at each execution cycle, the agent can either apply a planning goal rule, or execute the first step in any of its current plans.

For both strategies, we can represent the resulting agent system, (i.e., a set of planning goal rules plus an execution strategy) as a PDL program expression χ. We need to make a minor extension to the language of PDL to include syntactic modalities B and G. ($B\phi$ means that the agent believes ϕ, and $G\phi$ means that the agent has ϕ as a goal. Both modalities are interpreted by simply checking whether ϕ is in the agent's beliefs or goals respectively.) For the interleaving strategy, we also need to extend PDL with the interleaving operator introduced in (Abrahamson 1980). We can then express correctness properties such as 'after every execution of the agent's program, ϕ holds as $[\chi]\phi$.

Clearly, the behaviour of two agents with the same set of planning goal rules and the same initial state, but using different execution strategies, may differ. In particular, if an agent starts with beliefs $\phi_1, \ldots, \phi_n$ and goal ψ, it may be guaranteed to achieve the goal under the atomic execution strategy, but not under the interleaved execution strategy (e.g., it may be possible under the interleaved strategy to always execute a planning goal rule and never execute any basic actions). In logic, we can formalise this precisely. Let χ be the translation of the agent's program executed atomically and χ' be the translation of the same program executed in the interleaved fashion. Then the formula

$$B\phi_1 \wedge \ldots B\phi_n \wedge G\psi \rightarrow [\chi]B\psi$$

will be valid in all transition systems which satisfy pre- and postconditions on the basic actions, and

$$B\phi_1 \wedge \ldots B\phi_n \wedge G\psi \rightarrow [\chi']B\psi$$

will be not valid.

Evaluating Resource Requirements

In this section we consider the problem of comparing the relative resource requirements of architectures. For example, given two agents with different architectures, how much time or memory do they require to solve the same problem? Or, given two multi-agent systems possessing the same information, how many messages do the agents in each system have to exchange before they solve a given problem? We have studied these problems for rule-based agents (Alechina, Logan, & Whitsey 2004a; 2004b; Alechina & Logan 2005; Alechina, Jago, & Logan 2006) and for agents reasoning in a variety of logics (Alechina *et al.* 2006; Albore *et al.* 2006).

An important aspect of the architecture of a rule-based agent is how the rule instances which are applied to produce the next state are selected from the set of all matching rule instances (conflict resolution and rule application strategy). This can affect the time (number of rule application cycles) which an agent takes to select an action or to assert a certain fact in memory. In (Alechina, Logan, & Whitsey 2004a; 2004b) we axiomatised classes of transition systems corresponding to different kinds of conflict resolution strategies. The logic we used to axiomatise those classes of transition systems, Timed Reasoning Logic, uses time labels attached to formulas as in step logic (Elgot-Drapkin & Perlis 1990). This enabled us to express in the logic the fact that a certain statement will be derived by the agent within n timesteps. For example, in (Alechina, Logan, & Whitsey 2004b), we used Timed Reasoning Logic to model two rule-based agents with the same set of rules and the same set of initial observations, but with two different conflict resolution strategies. One agent used the *depth* conflict resolution strategy of CLIPS, which favours more recently derived information, while the other used a *breadth* conflict resolution strategy which favours older information. We showed how the times at which facts are derived differ for the two agents, and also that facts will be derived sooner if the two agents communicate.

As well as considering time requirements of agents with different architectures, we can also consider their respective memory requirements. A simple measure of the agent's memory requirement is the size of the agent's belief base, which can be identified either with the number of distinct beliefs in the belief base, or the total number of symbols required to represent all beliefs in the belief base. Logics for agents with bounded memory were studied in (Alechina *et al.* 2006; Albore *et al.* 2006), where we showed the existence of trade-offs between the agent's time and memory requirements: for example, deriving the same conclusion with a smaller memory may require more derivation steps.

Correct Implementation

The approach outlined above allows us to study properties of architectures in the abstract and of agent programs expressed in terms of architecture specific concepts. However it leaves open the question of whether a particular agent program described in implementation-level terms can be said to realise a particular architecture. In this section we sketch what it means in our view for an agent to implement or realise an architecture, and give an example in which we show how to correctly ascribe beliefs to a behaviour-based agent.

Given our definition of (a formal model of) an architecture as the set of all transition systems corresponding to programs which implement the architecture, there is a trivial sense in which a particular agent program can be seen as implementing an architecture: this holds if a transition system corresponding to the program is in the set of transition systems corresponding to the architecture (if it satisfies the constraints imposed by the architecture). However, in practice the problem is often more complex, in that the "natural" descriptions of the architecture and its implementation may use different concepts and levels of detail. For example, the language for specifying constraints on transition systems corresponding to a BDI architecture would involve ways of referring to beliefs, desires and intentions, while the description of an agent program may be in terms of basic programming constructs and data structures. We therefore define the notion of 'implementation' in terms of a mapping between the transition systems corresponding to an agent program and an architecture, and say that a particular agent implements a particular architecture if the transition system describing the agent program can be mapped into one of the transition systems in the architecture set. For example, if the low-level model of an agent involves two boolean state variables x and y, we may decide to translate $x = 1$ as 'the agent has belief p' and $y = 1$ as 'the agent has goal q'. Similarly we may collapse several of the agent's low-level actions into a single 'basic action' at the architecture level. If such a mapping into one of the transition systems corresponding to an architecture can be carried out (namely, there exists a transition system S in the set corresponding to the architecture, such that initial states of the agent translate into initial states of S, and every time when there is a sequence of low-level transitions in the agent's description, then there is a corresponding basic action in S, again leading to the matching states), we will say that the agent program implements the architecture.

This 'implementation relation' defines precisely when a given agent can be said to have or implement a particular architecture and hence satisfies all the properties of the architecture. Clearly, the same agent program may implement several different architectures under this definition. Indeed an agent can correctly be said to implement any architecture for which there exists a mapping from the agent model to the architectural description.

As an example, in (Alechina & Logan 2002) we showed how to correctly ascribe beliefs to agents which do not employ explicit representations of beliefs, e.g., where the behaviour of the agent is controlled by a collection of decision rules or reactive behaviours which simply respond to sensor inputs from the agent's environment. In particular, we showed that if we only ascribe an agent beliefs in literals then it is safe to model the agent's beliefs in a logic such as KD45 or S5 in which beliefs are modelled as closed un-

der logical consequence. (KD45 and S5 are widely used in formal models of agents, see, e.g., (Rao & Georgeff 1991; Fagin *et al.* 1995; van der Hoek, van Linder, & Meyer 1999; Wooldridge 2000).) However, it is not safe to ascribe beliefs in complex formulas to an agent, because then the actual state of the agent (which does not contain all the consequences of its beliefs) and the corresponding formal model (which assumes its beliefs to be logically closed) would not match.

Summary

We have presented a methodology for the formal evaluation of agent architectures. Our evaluation methodology consists of defining a set of transition systems corresponding to an architecture of interest, and verifying properties of this set of transition systems. The evaluation is therefore of the architecture in a very direct sense. The results of the evaluation hold for a particular agent (relative to some implementation-level description of the agent) insofar as the agent implements the architecture. While our analysis does not allow us to answer all questions of interest regarding the evaluation of architectures, we believe it allows us to address a (surprisingly) wide range of architectural evaluation questions. As examples we sketched to how establish correctness properties, response time properties and how to correctly ascribe beliefs to agents. However we feel that this is a potentially very productive area, and previous work has barely begun to scratch the surface of what is possible in terms of the evaluation at the architecture level.

Acknowledgements

This work was supported by EPSRC grant no. EP/E031226.

References

Abrahamson, K. R. 1980. *Decidability and expressiveness of logics of processes*. Ph.D. Dissertation, Department of Computer Science, University of Washington.

Albore, A.; Alechina, N.; Bertoli, P.; Ghidini, C.; Logan, B.; and Serafini, L. 2006. Model-checking memory requirements of resource-bounded reasoners. In *Proceedings of the Twenty-First National Conference on Artificial Intelligence (AAAI 2006)*, 213–218. AAAI Press.

Alechina, N., and Logan, B. 2002. Ascribing beliefs to resource bounded agents. In *Proceedings of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2002)*, volume 2, 881–888. Bologna: ACM Press.

Alechina, N., and Logan, B. 2005. Verifying bounds on deliberation time in multi-agent systems. In Gleizes, M. P.; Kaminka, G.; Nowe, A.; Ossowski, S.; Tuyls, K.; and Verbeeck, K., eds., *Proceedings of the Third European Workshop on Multiagent Systems (EUMAS'05)*, 25–34. Brussels, Belgium: Koninklijke Vlaamse Academie van Belgie voor Wetenschappen en Kunsten.

Alechina, N.; Bertoli, P.; Ghidini, C.; Jago, M.; Logan, B.; and Serafini, L. 2006. Verifying space and time requirements for resource-bounded agents. In Edelkamp, S., and Lomuscio, A., eds., *Proceedings of the Fourth Workshop on Model Checking and Artificial Intelligence (MoChArt-2006)*, 16–30.

Alechina, N.; Dastani, M.; Logan, B.; and Meyer, J.-J. C. 2007. A logic of agent programs. In *Proceedings of the Twenty-Second National Conference on Artificial Intelligence (AAAI 2007)*. AAAI Press. (to appear).

Alechina, N.; Jago, M.; and Logan, B. 2006. Modal logics for communicating rule-based agents. In Brewka, G.; Coradeschi, S.; Perini, A.; and Traverso, P., eds., *Proceedings of the 17th European Conference on Artificial Intelligence (ECAI 2006)*, 322–326. IOS Press.

Alechina, N.; Logan, B.; and Whitsey, M. 2004a. A complete and decidable logic for resource-bounded agents. In Jennings, N. R.; Sierra, C.; Sonenberg, L.; and Tambe, M., eds., *Proceedings of the Third International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2004)*, volume 2, 606–613. New York: ACM Press.

Alechina, N.; Logan, B.; and Whitsey, M. 2004b. Modelling communicating agents in timed reasoning logics. In Alferes, J. J., and oao Leite, J., eds., *Proceedings of the Ninth European Conference on Logics in Artificial Intelligence (JELIA 2004)*, number 3229 in LNAI, 95–107. Lisbon: Springer.

Bordini, R. H.; Dastani, M.; Dix, J.; and El Fallah Seghrouchni, A. 2005. *Multi-Agent Programming: Languages, Platforms and Applications*. Berlin: Springer.

Dastani, M.; van Riemsdijk, M. B.; Dignum, F.; and Meyer, J.-J. C. 2004. A programming language for cognitive agents: Goal directed 3APL. In *Proc. ProMAS 2003*, volume 3067 of *LNCS*, 111–130. Springer.

Elgot-Drapkin, J. J., and Perlis, D. 1990. Reasoning situated in time I: Basic concepts. *Journal of Experimental and Theoretical Artificial Intelligence* 2:75–98.

Fagin, R.; Halpern, J. Y.; Moses, Y.; and Vardi, M. Y. 1995. *Reasoning about Knowledge*. Cambridge, Mass.: MIT Press.

Fischer, M. J., and Ladner, R. E. 1979. Propositional dynamic logic of regular programs. *J. Comput. Syst. Sci.* 18(2):194–211.

Hintikka, J. 1962. *Knowledge and belief*. Ithaca, NY: Cornell University Press.

Rao, A. S., and Georgeff, M. P. 1991. Modeling rational agents within a BDI-architecture. In *Proceedings of the Second International Conference on Principles of Knowledge Representation and Reasoning (KR'91)*, 473–484.

van der Hoek, W.; van Linder, B.; and Meyer, J.-J. C. 1999. An integrated modal approach to rational agents. In Wooldridge, M., and Rao, A., eds., *Foundations of Rational Agency*. Dordrecht: Kluwer Academic. 133–168.

Wooldridge, M. 2000. *Reasoning About Rational Agents*. MIT Press.