What Does it Mean to Have an Architecture?*

Brian Logan

Abstract In this paper, I propose an approach to architectures which makes precise exactly what it means for an agent to 'have' an architecture, and allows us to establish properties of an agent expressed in terms of its architectural description. Using this approach, 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. I illustrate the approach with a number of examples which show both how to establish qualitative and quantitative properties of architectures and agent programs, and how to establish whether a particular agent has a particular architecture. I focus on architectures of software agents, however a similar approach can be applied to robots and other kinds of intelligent systems.

1 Introduction

The notion of an agent's 'architecture' is a fundamental idea in agent research. Architectures abstract from the details of individual agent programs, allowing us to make general statements about the properties of a whole class of agents, and hence which kinds of agents are more appropriate for particular environments. From an engineering perspective, choosing an appropriate architecture is therefore a key step in developing an agent to perform a particular task in a specified environment. From a scientific perspective, the notion of an architecture is key to understanding the range of possible ways in which (artificial) intelligence can be realised, or how it might have evolved.

Brian Logan
School of Computer Science, University of Nottingham, Nottingham NG8 1BB, UK
e-mail: bsl@cs.nott.ac.uk

* This paper is a revised an extended version of a paper presented at the 2007 AAAI *Workshop on Evaluating Architectures for Intelligence* (Alechina and Logan, 2007).

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 software agent) and a set of (usually high-level) capabilities realised by that 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 despite the importance of architectures in research on agent systems, I will argue the notion of "an architecture" remains hard to pin down. The agent literature offers little beyond descriptions of particular architectures and some fairly preliminary classifications of architectures and/or architectural components into 'reactive', 'deliberative', 'hybrid' etc. (see, for example, (Nilsson, 1998; Sloman and Scheutz, 2002)). While such classifications can be useful, it can be difficult to characterise a particular agent architecture with any precision, or to determine its properties. There are many variants of "Belief-Desire-Intention" (BDI) architectures in the agent literature, see for example, (Georgeff et al, 1999), however we lack a framework for their systematic analysis of their properties. This lack of precision makes it hard to compare or evaluate architectures or to say that a given intelligent system has a particular property as a consequence of it having a particular architecture.

In this paper, I propose an approach to architectures and their analysis 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, I focus on architectures of software agents, however a similar approach can be applied to robots and other kinds of intelligent systems. I show how to establish a precise formal relationship between an architectural property, an agent architecture and an agent model (taken to be an implementation-level description of an agent). A consequence of this 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. I illustrate the approach with a number of examples drawn from our previous work at Nottingham, 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.

2 What is an Architecture?

In AI textbooks, the behaviour of an agent is often specified by an *agent function* (also known as an action selection function) which maps a sequence of percepts to an action (see, for example, (Russell and Norvig, 2003)). Such an abstractly specified agent function is implemented by an *agent program*, which may be written in a conventional programming language, or in a special-purpose agent programming language. The agent program in turn runs on an *agent architecture*. The architecture is a (possibly virtual) machine that makes the percepts from the agent's sensors

available to the agent program, runs the program and passes the action(s) selected by the program to the agent's actuators.

In this view, agent programming is conventionally conceived as the problem of synthesising an agent function. For example, Russell & Norvig claim that: 'the job of AI is to design the agent program that implements the agent function mapping percepts to actions' (2003, p.44). While this characterisation of AI is correct, it is not very helpful: designing an agent program that "implements the agent function mapping percepts to actions" is a very difficult problem. The AI literature describes techniques and algorithms that can be used to solve parts of the problem, however there is little guidance on how these various components could or should fit together to form an integrated system.

One way of making the problem more tractable is by drawing on the notion of an agent architecture. The notion of "agent architecture" is ubiquitous in the agent literature but is not well analysed. For example, Russell & Norvig view the architecture as simply 'some sort of computing device with physical sensors and actuators' (2003, p.44).[1] Moreover, when architectures are discussed, it is often in the context of a particular agent programming language or platform, making it difficult to separate relatively minor language features from the essentials of the architecture.

For the purposes of this paper, we can think of an architecture as defining a (real or virtual) machine that runs the agent program. The architecture defines the *atomic operations* of the agent program and implicitly determines the components of the agent. It determines which operations happen automatically, without the agent program (or programmer) having to do anything, e.g., the interaction between memory, learning and reasoning. As a result, an architecture constrains kinds of agent programs we can write (easily). For example, implementing a complex search algorithm in an architecture that supports only simple condition action rules will be more difficult than in an architecture that supports backtracking. In this view, an agent architecture can be seen as defining a class of agent programs. Just as programs have properties that make them more or less successful in a given task environment, architectures (classes of programs) have higher-level properties that determine their suitability for a task environment. Choosing an appropriate architecture can therefore make it much easier to develop an agent program for a particular task environment.

In many cases, it can be difficult to say what "counts as" the architecture of an agent. A particular agent implementation may consist of a whole hierarchy of virtual machines. For example, 'the' architecture is usually implemented in terms of a programming language, which in turn is implemented using the instruction set of a particular CPU (or a JVM). Likewise some 'agent programs' together with their underlying architecture can implement a new, higher-level architecture (virtual machine), as when, for example, encapsulated robust behaviours in a hybrid architecture effectively define a new set of atomic operations for the deliberative level. In

[1] Although Russell and Norvig (2003) state that "agent = architecture + program", most of the book is about designing (components of) agent programs, and there are only ten references to "architecture" in nearly 1000 pages.

what follows, when used without qualification, 'agent architecture' means the most abstract architecture or the highest level virtual machine.

How can we make this notion of agent architecture precise? This notion of an 'agent architecture' is clearly related to the more general notion of 'software architecture'. In their classic paper on the foundations of software architectures, Perry & Wolf characterise software architecture as: 'concerned with the selection of architectural elements, their interactions, and the constraints on those elements and their interactions necessary to provide a framework in which to satisfy the requirements and serve as a basis for the design' (1992, p.43). A similar view is advanced in a recent survey by Shaw & Clements, who define software architectures as the 'principled study of the large-scale structures of software systems' (2006, p.31). However, while there has been considerable work in software engineering resulting in standard architectures for many domains and applications, e.g., *n*-tier client-server architectures, service oriented architectures, etc., I would argue (claims in (Shaw and Clements, 2006) notwithstanding) that the notion of architecture as used in software engineering is hardly any more precise than that used in AI.

The notion of an agent architecture is also related to the notion of a 'cognitive architecture' as used in artificial intelligence and cognitive science. A cognitive architecture can be defined as an integrated system capable of supporting intelligence. The term is often used to denote models of human reasoning, e.g., ACT-R (Anderson and Libiere, 1998), SOAR (Newell, 1990). However in other cases no claims about psychological plausibility are made. In this latter sense, 'cognitive architecture' is more or less synonymous with 'agent architecture', as used here. However, while a number of cognitive architectures have been extensively studied, there has been relatively little work on understanding how different cognitive architectures relate to each other, or on which features of an architecture are responsible for its performance on a particular task.

I believe that what is required is a more precise characterisation of what an 'architecture' is and what it means for an agent to have one. In the remainder of this paper, I sketch out one approach to a more formal characterisation of agent architectures that draws heavily on joint work with colleagues and students at Nottingham and elsewhere.

3 Specifying Architectures and Properties

An architecture is a way of looking at or conceptualising an agent program that 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 ac-

tion', or by higher-level capabilities such as 'perception', 'learning' or 'planning'. Each architecture exposes some of the contents of the states and some of the possible transitions to 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 state occurring in a particular task environment or problem domain. 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 beliefs of an agent imposed by its architecture may be that its 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 this 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 the 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 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 (see, for example, (Rao and Georgeff, 1991; Fagin et al, 1995; van der Hoek et al, 1999; Wooldridge, 2000)). The major difference between this work and the approach advocated here,

is our use of 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.

In the remainder of this paper I illustrate this approach by means of some simple examples drawn from our previous work.

4 Correctness Properties

In this section I briefly survey some of our previous work on establishing the correctness of agent programs and show how the same approach can be lifted to the level of agent architectures. In (Alechina et al, 2007, 2010b) we proposed a sound and complete logic for agent programs written in SimpleAPL, an APL-like (Dastani et al, 2004; Dastani and Meyer, 2008) agent programming language. The logic is a variant of Propositional Dynamic Logic (PDL) (Fischer and Ladner, 1979) extended with belief and goal operators, which allows the specification of safety and liveness properties of SimpleAPL agent programs. In that work, we showed how to define transition systems corresponding to different program execution strategies in logic, however the execution strategies were 'hard-coded' into the translation of agent programs. In (Alechina et al, 2008a, 2010a) we extended this approach to define transition systems corresponding to different program execution strategies using a single fixed translation of the agent program together with an axiomatisation of the agent's execution strategy. This more abstract formalisation makes it much easier to analyse the implications of a key aspect of the agent's architecture — its deliberation strategy. As an illustration, we showed how the choice of deliberation strategy can determine whether an agent is able to achieve it goal(s). In more recent work (Doan et al, 2009; Alechina et al, 2011), we have extended this approach further to consider agents which are able to modify their plans at run time. We consider the plan revision rules of the agent programming languages Dribble (van Riemsdijk et al, 2003), 3APL (Dastani et al, 2004, 2005) and 2APL (Dastani and Meyer, 2008; Dastani, 2008) which allow an agent to revise its plans at run time in response to changes in its environment and the effects nondeterministic action execution. We show how such agents can be formalised in logic, and how to verify safety properties of agent programs that allow plan revision. This work can be seen as a (very preliminary) formalisation of a basic reflective capability, i.e., the ability of an agent to modify a chosen course of action in response to unanticipated opportunities or failures in the execution of a plan.

5 Architectural Properties

While our previous work has touched on architectural issues such an agent's deliberation strategy and simple reflective capabilities, our focus was mainly on the correctness of agent *programs* implemented using these architectures. In this section, I will briefly outline how the same approach can be applied to 'pure' agent *architectures*, using the SimpleAPL architecture as an example. I first briefly describe the language and architecture of SimpleAPL and then sketch how to formalise them in logic.

SimpleAPL is a fragment of 3APL (Dastani et al, 2004, 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

$$\psi \leftarrow \phi \mid \pi$$

can be read as "if you have a goal ψ and you believe ϕ, then adopt plan π". SimpleAPL plans are predefined by the programmer, and 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. The SimpleAPL agent architecture attempts to achieve the agent's goals by selecting appropriate plans given the agent's beliefs. A SimpleAPL agent may have several active plans at any given time. At each execution cycle, the agent can either apply a planning goal rule to select a new plan, or execute the first step in any of its current plans.

5.1 Operational Semantics

We can define the formal semantics of SimpleAPL in terms of a transition system. Each transition corresponds to a single execution step and takes the agent from one configuration to another. Configurations consist of the beliefs, goals, and plans of the agent.

Definition 1. The configuration of an agent is defined as $\langle \sigma, \gamma, \Pi \rangle$ where σ is a set of literals representing the agent's beliefs, γ is a set of literals representing the agent's goals, and Π is a set of plan entries representing the agent's current active plans.

An agent's initial beliefs and goals are specified by its program, and Π is initially empty. Executing the agent's program modifies its initial configuration in accordance with a set of transition rules. Below we give only the transition rule for basic actions; for the full operational semantics see (Alechina et al, 2007).

Basic Actions

A basic action α can be executed if its precondition is entailed by the agent's beliefs, i,e., $\sigma \models \phi$. Executing the action adds the literals in the postcondition to the agent's beliefs and removes any existing beliefs which are inconsistent with the postcondition.

$$\frac{\alpha;\pi \in \Pi \quad T(\alpha,\sigma) = \sigma' \quad \gamma' = \gamma \setminus \{\phi \in \gamma \mid \sigma' \models \phi\}}{\langle \sigma, \gamma, \Pi \rangle \longrightarrow \langle \sigma', \gamma', \Pi \setminus \{\alpha;\pi\} \cup \{\pi\} \rangle}$$

T is a partial function that takes a belief update action α and a belief set σ, and returns the modified belief set if the precondition of the action is entailed by σ. Note that executing a belief update action causes the agent to drop any goals it believes to be achieved as a result of the update.

5.2 Logic

We can define a logic which allows us to specify properties of the SimpleAPL agent architecture. We begin by defining transition systems which capture the *capabilities* of agents as specified by their basic actions.[2] We then show how to interpret a variant of the temporal logic CTL with belief and goal operators in this semantics.

States and transitions

Let P denote the set of propositional variables used to describe agent's beliefs and goals. A state s is a pair $\langle \sigma, \gamma \rangle$, where:

- σ is a set of beliefs $\{(-)p_1, \ldots, (-)p_n : p_i \in P\}$. We assume that belief states are consistent, i.e., for no $p \in P$ both p and $-p \in \sigma$.
- γ is a set of goals $\{(-)u_1, \ldots, (-)u_n : u_i \in P\}$. The set of goals does not have to be consistent, but it has to be disjoint from σ: no element of γ is in σ (i.e., is already believed).

Let the set of basic actions be $\mathtt{Ac} = \{\alpha_1, \ldots, \alpha_m\}$. We associate with each $\alpha_i \in \mathtt{Ac}$ a set of pre- and postconditions of the form $\{(-)p_1 \in \sigma, \ldots, (-)p_n \in \sigma\}$, $\{(-)q_1 \in \sigma', \ldots, (-)q_k \in \sigma'\}$ (where ps and qs are not necessarily disjoint) which mean: if α_i is executed in a state with belief set σ that satisfies the precondition, then the resulting state s' has the belief set σ' that satisfies the postcondition (including replacing p with $-p$ if necessary to restore consistency), *the rest of* σ' *is the same as* σ, and the goal set $\gamma' = \gamma \setminus \{(-)p : (-)p \in \sigma'\}$.

[2] Note that the transition systems are more general than operational semantics of the SimpleAPL architecture presented informally above, in that they do not describe a particular agent program or execution strategy, but all possible basic transitions between all the belief and goal states of an agent.

Executing an action α_i in different configurations may give different results. For each α_i, we denote the set of pre- and postcondition pairs $\{(\mathtt{prec}_1, \mathtt{post}_1), \ldots, (\mathtt{prec}_l, \mathtt{post}_l)\}$ by $C(\alpha_i)$. We assume that $C(\alpha_i)$ is finite, that preconditions $\mathtt{prec}_j, \mathtt{prec}_k$ are mutually exclusive if $j \neq k$, and that each precondition has exactly one associated postcondition. We denote the set of all pre- and postconditions by **C**.

Language

The language L for talking about the agent's beliefs, goals and plans is the language of CTL extended with belief operator B and goal operator G. A formula of L is defined as follows: if $p \in P$, then $B(-)p$ and $G(-)p$ are formulas; if ϕ and ψ formulas, then $EX\phi$, $E\Box\phi$,[3] and $EU(\phi, \psi)$ are formulas; and L is closed under the usual boolean connectives.

Semantics

A model for L is a structure $M = (S, \{R_{\alpha_i} : \alpha_i \in \mathtt{Ac}\}, V)$, where

- S is a set of states.
- $V = (V_b, V_g)$ is the evaluation function consisting of belief and goal valuation functions V_b and V_g such that for $s = \langle \sigma, \gamma \rangle$, $V_b(s) = \sigma$ and $V_g(s) = \gamma$.
- R_{α_i}, for each $\alpha_i \in \mathtt{Ac}$, is a relation on S such that $(s, s') \in R_{\alpha_i}$ iff for some $(\mathtt{prec}_j, \mathtt{post}_j) \in C(\alpha_i)$, $\mathtt{prec}_j(s)$ and $\mathtt{post}_j(s')$, i.e., for some pair of pre- and postconditions of α_i, the precondition holds for s and the corresponding postcondition holds for s'. Note that this implies two things: first, an α_i transition can only originate in a state s which satisfies one of the preconditions for α_i; second, since pre-conditions are mutually exclusive, every such s satisfies exactly one pre-condition, and all α_i-successors of s satisfy the matching post-condition.

The relation $\models$ of a formula being true in a state of a model is defined inductively as follows:

- $M, s \models B(-)p$ iff $(-)p \in V_b(s)$
- $M, s \models G(-)p$ iff $(-)p \in V_g(s)$
- $M, s \models \neg\phi$ iff $M, s \not\models \phi$
- $M, s \models \phi \wedge \psi$ iff $M, s \models \phi$ and $M, s \models \psi$
- $M, s \models EX\phi$ iff on some computation path starting in s ϕ holds in the next state
- $M, s \models E\Box\phi$ iff there exists a computation path starting in s such that ϕ holds globally (in every state along the path including s)
- $M, s \models EU(\phi, \psi)$ iff there exists a computation path starting in s such that ϕ holds in every state along the path (including s) until ψ holds

[3] Note that we use $\Box$ rather than the more conventional G to avoid confusion with the goal operator.

Let the class of transition systems defined above be denoted $\mathbf{M_P}$ (note that $\mathbf{M}$ is parameterised by the set $\mathbf{P}$ of all possible SimpleAPL programs).

In this logic, we can state properties of the SimpleAPL architecture and prove (by semantic argument) that they are valid (true in all models). For example, we can show that the same formula cannot be both a goal and a belief

$$A\Box\neg(B\phi \wedge G\phi)$$

where $A\Box\psi \equiv \neg EU(\top, \neg\psi)$. More interestingly, we can show that a SimpleAPL agent is blindly or fanatically committed to to its goals (Rao and Georgeff, 1992), i.e., it will only abandon a goal if it believes it has been achieved

$$G\phi \rightarrow A\Box(G\phi \vee B\phi)$$

These examples are very simple. However they serve to illustrate how this approach allows the precise specification of properties of an agent architecture.

6 Resource Requirements

We can use a similar approach to compare 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 et al, 2004a,b; Alechina and Logan, 2005; Alechina et al, 2006b) and for agents reasoning in a variety of logics (Alechina et al, 2006a; Albore et al, 2006; Alechina et al, 2008b).

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 (i.e., the agent's conflict resolution strategy). This can affect the time (number of rule application cycles) that an agent takes to select an action or to assert a certain fact in memory. In (Alechina et al, 2004a,b) we axiomatised classes of transition systems corresponding to different kinds of conflict resolution strategies in Timed Reasoning Logic (TRL). TRL attaches time labels to formulas, allowing 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 et al, 2004b), we used TRL 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, 2006a; 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. In (Alechina et al, 2008b) we considered the interaction of time, memory and communication in systems of distributed reasoning agents, and showed that the agents can achieve a goal only if they are prepared to commit certain time, memory and communication resources.

7 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 examples in which we show how to correctly ascribe beliefs to a behaviour-based agent, and how to model the ascription of beliefs in systems of agents which reason about each others' beliefs.

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. 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 and 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 under logical consequence. (KD45 and S5 are widely used in formal models of agents, see, e.g., (Rao and Georgeff, 1991; Fagin et al, 1995; van der Hoek et al, 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. In (Alechina and Logan, 2009) we extended this approach to model the 'sense', 'think' and 'act' phases of an agent's deliberation cycle, allowing correct ascription of beliefs at each stage of the agent's execution. In (Alechina and Logan, 2010) we considered belief ascription for systems of agents which reason about each other's beliefs. We showed that for agents whose computational resources and memory are bounded, correct ascription of beliefs cannot be guaranteed, even in the limit. We proposed a solution to the problem of correct belief ascription for feasible agents which involves ascribing reasoning strategies, or preferences on formulas, to other agents, and showed that if a resource-bounded agent knows the reasoning strategy of another agent, then its ascription of beliefs to the other agent is correct in the limit.

This work could be seen as extending (Sloman, 1994) in explicating the ways in which "architecture dominates mechanism". However, I believe it can also help us understand the ways in which a single mechanism may simultaneously realise multiple architectures.

8 Summary

I have presented a methodology for the formal characterisation and analysis of agent architectures. The methodology consists of defining a set of transition sys-

tems corresponding to an architecture of interest, and verifying properties of this set of transition systems. A key feature of this approach is the use of syntactic belief, goal, etc. modalities rather than a standard Kripke-style semantics to model the beliefs and other propositional attitudes of agents, so avoiding the problem of logical omniscience. The characterisation is therefore of the architecture in a very direct sense. The results of the analysis hold for a particular agent (relative to some implementation-level description of the agent) insofar as the agent implements the architecture. While such an analysis does not allow us to answer all questions of interest regarding architectures, I believe it allows us to address a (surprisingly) wide range of architectural issues. As examples I sketched to how establish correctness properties, quantitative properties and how to correctly ascribe beliefs to agents. However I believe this is a potentially very productive area, and our previous work has barely begun to scratch the surface of what it means to have an architecture.

Acknowledgements

I owe a great debt to Natasha Alechina: without her logical abilities and willingness to invent nonstandard logical approaches to models of agents, none of this work would have been possible. I am also indebted to colleagues in the agent programming language community, particularly Rafael Bordini, Mehdi Dastani, Koen Hindriks and John-Jules Meyer and to former and current students in the Agents Lab at Nottingham, including Mark Jago, Neil Madden, Abdur Rakib, Nguyen Hoang Nga, Fahad Khan and Doan Thu Trang.

References

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

Alechina N, 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), ACM Press, Bologna, vol 2, pp 881–888

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

Alechina N, Logan B (2007) Formal evaluation of agent architectures. In: Kaminka GA, Burghart CR (eds) Evaluating Architectures for Intelligence: Papers from the 2007 AAAI Workshop, AAAI Press, pp 1–4, Technical Report WS–07–04

Alechina N, Logan B (2009) A logic of situated resource-bounded agents. Journal of Logic, Language and Information 18(1):79–95, DOI http://dx.doi.org/10.1007/s10849-008-9073-6

Alechina N, Logan B (2010) Belief ascription under bounded resources. Synthese 173(2):179–197, DOI http://dx.doi.org/10.1007/s11229-009-9706-6

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

Alechina N, Logan B, Whitsey M (2004b) Modelling communicating agents in timed reasoning logics. In: Alferes JJ, Leite J (eds) Proceedings of the Ninth European Conference on Logics in Artificial Intelligence (JELIA 2004), Springer, Lisbon, no. 3229 in LNAI, pp 95–107

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

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

Alechina N, Dastani M, Logan B, Meyer JJC (2007) A logic of agent programs. In: Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence (AAAI 2007), AAAI Press, pp 795–800

Alechina N, Dastani M, Logan B, Meyer JJC (2008a) Reasoning about agent deliberation. In: Brewka G, Lang J (eds) Proceedings of the Eleventh International Conference on Principles of Knowledge Representation and Reasoning (KR'08), AAAI, Sydney, Australia, pp 16–26

Alechina N, Logan B, Nga NH, Rakib A (2008b) Verifying time, memory and communication bounds in systems of reasoning agents. In: Padgham L, Parkes D, Müller J, Parsons S (eds) Proceedings of the Seventh International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2008), IFAAMAS, IFAAMAS, Estoril, Portugal, vol 2, pp 736–743

Alechina N, Dastani M, Logan B, Meyer JJC (2010a) Reasoning about agent deliberation. Autonomous Agents and Multi-Agent Systems 22(2):1–26, DOI 10.1007/s10458-010-9129-2, URL `http://dx.doi.org/10.1007/s10458-010-9129-2`

Alechina N, Dastani M, Logan B, Meyer JJC (2010b) Using theorem proving to verify properties of agent programs. In: Dastani M, Hindriks KV, Meyer JJC (eds) Specification and Verification of Multi-agent Systems, Springer, chap 2, pp 1–34

Alechina N, Dastani M, Logan B, Meyer JJC (2011) Reasoning about plan revision in BDI agent programs. Theoretical Computer Science DOI 10.1016/j.tcs.2011.05.052, (in press)

Anderson JR, Libiere C (1998) The Atomic Components of Thought. Lawrence Erlbaum Associates

Dastani M (2008) 2APL: a practical agent programming language. Autonomous Agents and Multi-Agent Systems 16(3):214–248

Dastani M, Meyer JJC (2008) A practical agent programming language. In: Dastani M, El Fallah-Seghrouchni A, Ricci A, Winikoff M (eds) Proceedings of the Fifth International Workshop on Programming Multi-agent Systems (ProMAS'07), Springer, LNCS, vol 4908, pp 107–123

Dastani M, van Riemsdijk MB, Dignum F, Meyer JJC (2004) A programming language for cognitive agents: Goal directed 3APL. In: Dastani M, Dix J, El Fallah-Seghrouchni A (eds) Programming Multi-Agent Systems, First International Workshop, ProMAS 2003, Melbourne, Australia, July 15, 2003, Selected Revised and Invited papers, Springer, LNCS, vol 3067, pp 111–130, DOI http://www.springerlink.com/content/l7dqkvqh5u941l4b

Dastani M, van Riemsdijk MB, Meyer JJC (2005) Programming multi-agent systems in 3APL. In: Bordini RH, Dastani M, Dix J, El Fallah-Seghrouchni A (eds) Multi-Agent Programming: Languages, Platforms and Applications, Multiagent Systems, Artificial Societies, and Simulated Organizations, vol 15, Springer, pp 39–67

Doan TT, Logan B, Alechina N (2009) Verifying Dribble agents. In: Baldoni M, Bentahar J, Lloyd J, van Riemsdijk MB (eds) Seventh International Workshop on Declarative Agent Languages and Technologies (DALT 2009), Workshop Notes, Budapest Hungary, pp 162–177

Fagin R, Halpern JY, Moses Y, Vardi MY (1995) Reasoning about Knowledge. MIT Press, Cambridge, Mass.

Fischer MJ, Ladner RE (1979) Propositional dynamic logic of regular programs. Journal of Computer and System Sciences 18(2):194–211

Georgeff MP, Pell B, Pollack ME, Tambe M, Wooldridge M (1999) The Belief-Desire-Intention model of agency. In: Müller JP, Singh MP, Rao AS (eds) Intelligent Agents V, Agent Theories, Architectures, and Languages, 5th International Workshop, (ATAL'98), Paris, France, July 4-7, 1998, Proceedings, Springer, Lecture Notes in Computer Science, vol 1555, pp 1–10

Hintikka J (1962) Knowledge and belief. Cornell University Press, Ithaca, NY

van der Hoek W, van Linder B, Meyer JJC (1999) An integrated modal approach to rational agents. In: Wooldridge M, Rao A (eds) Foundations of Rational Agency, Kluwer Academic, Dordrecht, pp 133–168

Newell A (1990) Unified Theories of Cognition. Harvard University Press, Cambridge, Mass.

Nilsson N (1998) Artificial Intelligence: a new synthesis. Morgan Kaufmann Publishers

Perry DE, Wolf AL (1992) Foundations for the study of software architecture. SIGSOFT Software Engineering Notes 17:40–52, DOI http://doi.acm.org/10.1145/141874.141884

Rao AS, Georgeff MP (1991) Modeling rational agents within a BDI-architecture. In: Proceedings of the Second International Conference on Principles of Knowledge Representation and Reasoning (KR'91), pp 473–484

Rao AS, Georgeff MP (1992) An abstract architecture for rational agents. In: Rich C, Swartout W, Nebel B (eds) Proceedings of Knowledge Representation and Reasoning (KR&R-92), pp 439–449

van Riemsdijk B, van der Hoek W, Meyer JJC (2003) Agent programming in Dribble: from beliefs to goals using plans. In: Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'03), ACM Press, New York, NY, USA, pp 393–400, DOI http://doi.acm.org/10.1145/860575.860639

Russell S, Norvig P (2003) Artificial Intelligence: a modern approach, 2nd edn. Prentice Hall

Shaw M, Clements P (2006) The golden age of software architecture. IEEE Software 23:31–39, DOI 10.1109/MS.2006.58

Sloman A (1994) Semantics in an intelligent control system. Philosophical Transactions of the Royal Society: Physical Sciences and Engineering 349(1689):43–58

Sloman A, Scheutz M (2002) A framework for comparing agent architectures. In: Proceedings of the UK Workshop on Computational Intelligence UKCI'02, pp 169–176

Wooldridge M (2000) Reasoning About Rational Agents. MIT Press