
Autonomous Agents and Multi-Agent Sytems, 10, 41–66, 2005
© 2005 Springer Science + Bussiess Media, Inc. Manufactured in The Netherlands.

Temporal Development Methods for Agent-Based Systems

MICHAEL FISHER m.fisher@csc.liv.ac.uk
Department of Computer Science, University of Liverpool, Liverpool L69 3BX, United Kingdom

Abstract. In this paper we overview one specific approach to the formal development of multi-agent systems. This approach is based on the use of *temporal logics* to represent both the behaviour of individual agents, and the macro-level behaviour of multi-agent systems. We describe how formal specification, verification and refinement can all be developed using this temporal basis, and how implementation can be achieved by directly executing these formal representations. We also show how the basic framework can be extended in various ways to handle the representation and implementation of agents capable of more complex deliberation and reasoning.

Keywords: agent-based systems, formal methods, temporal and modal logics.

1. Introduction

Computational power is increasing, and increasingly available, for example via the development of ubiquitous computing. Once large numbers of computational elements can communicate with each other, via wireless networks or the INTERNET, then new problems arise in engineering software for such systems. By representing these computational elements as agents, we can provide a simple and intuitive metaphor for both individual and collective computation. However, software designers need to have appropriate, and semantically clear, mechanisms for representing not only how individual agents adapt and evolve, but also how agents interact and combine to form new systems. Without this ability to represent and control agents not only will the practical development of complex multi-agent systems remain difficult, but agents themselves will not be trusted for use in critical applications.

Many advocate the use of *formal logic* as a way to represent the behaviour of both individuals and organisations, since it provides an unambiguous notation whereby logics can be designed for many scenarios and where formal properties of logical descriptions are well understood. In particular, choosing the appropriate logic provides a level of abstraction close to the key concepts of the software. Consequently, a number of logical theories of rational agency have been developed, such as the BDI [68] and KARO [63] frameworks. These frameworks are usually represented as complex non-classical logics. In addition to their use in agent theories, where the basic representation of agency and rationality is explored, these logics form the basis for agent-based formal methods.

The leading agent theories and formal methods in this area all share similar logical properties. In particular, the logics used have:

– an *informational* component, such as being able to represent an agent's beliefs or knowledge,
– a *dynamic* component, allowing the representation of dynamic activity, and,
– a *motivational* component, often representing the agents desires, intentions or goals.

Thus, the predominant approaches use relevant logical combinations: the BDI approach combines dynamic aspects via propositional temporal logic, informational aspects via modal logic KD45, and motivational aspects via modal logic KD; the KARO approach combines dynamic aspects via propositional dynamic logic, informational aspects via modal logics S5/KD45, and motivational aspects via modal logic KD. Unfortunately, these combinations can often be complex, both in terms of understandability and reasoning.

We take a simplified view and base our approach initially on just *temporal logic*. Temporal logic is an extension of classical logic with the notion of temporal order built in. With such logics we can describe many dynamic properties, but they reduce to describing what we must do *now*, what we must do *next*, and what we guarantee to do at *some* point in the future. This, seemingly simple, language gives us the flexibility to represent a wide range of computational activities. In particular, we are able to describe individual components, in terms of their dynamic behaviour, how their knowledge/information evolves, and how their goals/motivations evolve.

Thus, this paper provides an overview of our approach to using temporal logic in the specification, verification and implementation of agent-based systems, and discusses how this basis can be extended to handle more complex varieties of agent. Although details of many aspects of this work have appeared previously (see references in text), this paper brings these aspects together and provides an overall framework for this variety of agent-based formal methods. In this sense, the current paper can be seen as a much broader treatment than that given in [34], which was primarily concerned with specification and verification using primarily temporal approaches.

In order to provide a context for the work described in this paper, we briefly review three important aspects. First, what we consider an agent to be, then what types of agent based systems we are interested in and, finally, what particular techniques we require to be present within our framework.

1.1. Agents

The question "what is an agent?" has spawned considerable debate in the philosophical and agent research communities [36]. Rather than producing yet another definition, we choose to follow the definition provided by Wooldridge and Jennings [67], namely that an agent is an autonomous component that is *pro-active* (i.e. can initiate computation on its own behalf), *reactive* (i.e. can respond, in a timely manner, to environmental stimuli), and *social* (i.e. can, and often must, communicate with other agents in order to achieve its goals).

Later we will see how the formal framework we have developed can cope with stronger notions of agency, for example characterising an agent's behaviour directly

in terms of mental attitudes [7]. However, at least initially, we will consider agents that are captured by the above definition.

1.2. Rational agents

The use of software agents is expanding rapidly, primarily within applications related to the INTERNET. While such agents are often relatively simple, more complex, *rational*, agents have been shown to particularly useful in dealing flexibly and dynamically with complex and rapidly changing environments. Such agents are often constructed using complex software architectures, and have been applied in areas such as real-time process control [45, 61]. A particularly impressive example is the use of a rational agent architecture as part of the real-time fault monitoring and diagnosis carried out in the Deep Space One mission [54].

With agents beginning to be used in such critical applications, it is clear that more precise, and logically well-founded, development techniques will be required for agent-based applications in the future. However, as Bradshaw et al. note [6],

a large and ugly chasm still separates the world of formal theory and infrastructure from the world of practical ... agent-system development.

Thus, our intention throughout this work is to develop formal techniques that can provide the basis for specification, verification and implementation of critical agent-based systems, for example in areas such as process control, electronic commerce and information management.

1.3. Formal development framework

The formal framework we arc concerned with consists of a number of elements that support the principled development of agent-based systems. In particular, the framework comprises

- a logic in which the high-level behaviours (of both agent and system) can be concisely specified,
- techniques for (safely) refining these high-level specifications into (one or more) sub-specifications,
- mechanisms for verifying the logical properties of the specifications, and
- a programming language providing concepts close to the specification notation used.

In Section 3 we will show how this framework can be instantiated for the development of simple autonomous agents by using a basically temporal approach while, in Section 4, we show how a temporal framework can also be provided for the development of *multi-agent* systems. In Sections 5 and 6, we extend our techniques from a purely temporal basis to cover more sophisticated individual agents incorporating reasoning about beliefs and deliberation, while in Section 7, we extend our work on multi-agent systems to consider more complex organisational aspects. Finally, in Section 8, we provide conclusions from this work and outline future directions of this research.

In developing all this work, our aim has been for the framework to be powerful, yet both simple and intuitive. Consequently, the core formalism we use is *temporal logic*, which provides clear concepts capturing the dynamic aspects of systems [56]. We begin, in Section 2, with a brief overview of the basic temporal logic that underlies our approach.

2. Temporal logic

Rather than providing a detailed description of the temporal logic we use, we will initially review the key aspects of the logic which will be useful in this paper. For a more thorough exposition of the formal properties of this logic, see [14], while for examples of the use of temporal logic in program specification in general, see [51].

2.1. Review of syntax and semantics

Temporal logic is an extension of classical logic, whereby time becomes an extra parameter when considering the truth of logical statements. The variety of temporal logic we are particularly concerned with is based upon a discrete, linear model of time, having both a finite past and infinite future, i.e.

$$\sigma = s_0, s_1, s_2, s_3, \ldots$$

Here, a model (σ) for the logic is an infinite sequence of states which can be thought of as 'moments' or 'points' in time. Since we will only consider propositional temporal logic here, then, associated with each of these states is a valuation for all the propositions in the language.

The temporal language we use is that of classical logic extended with various modalities characterising different aspects of the temporal structure above. Examples of the key operators include '$\bigcirc\varphi$' which is satisfied if the formula φ is satisfied at the *next* moment in time, '$\Diamond\varphi$', which is satisfied if φ is satisfied at *some* future moment in time, and '$\Box\varphi$' which is satisfied if φ is satisfied at *all* future moments in time.

More formally, the semantics of the language can be defined with respect to the model (σ) in which the statement is to be interpreted, and the moment in time (i) at which it is to be interpreted. Thus, the semantics for the key temporal operators is given as follows.

$$\begin{array}{lll}
\langle\sigma, i\rangle \models \bigcirc A & \text{iff} & \langle\sigma, i+1\rangle \models A \\
\langle\sigma, i\rangle \models \Box A & \text{iff} & \text{for all } j \geq i.\ \langle\sigma, j\rangle \models A \\
\langle\sigma, i\rangle \models \Diamond A & \text{iff} & \text{exists } j \geq i.\ \langle\sigma, j\rangle \models A \\
\langle\sigma, i\rangle \models A\mathcal{U}B & \text{iff} & \text{exists } k \geq i.\ \langle\sigma, k\rangle \models B \text{ and for all } k > j \geq i.\langle\sigma, j\rangle \models A \\
\langle\sigma, i\rangle \models A\mathcal{W}B & \text{iff} & A\mathcal{U}B \text{ or } \Box A
\end{array}$$

The temporal operators '$\mathcal{U}$' ("until") and $\mathcal{W}$ ("unless") above characterise intervals within the temporal sequence during which certain properties hold. Thus $\psi\mathcal{U}\varphi$ means that φ must be satisfied at some point in the future and, at every moment between now and that point, ψ must be satisfied.

In addition to temporal operators referring to the *future*, it is also possible to utilise various temporal operators relating to the *past*, such as ■ (“always in the past”), ♦ (“sometime in the past”), ⊙ (“in the previous moment in time”), and $\mathcal{S}$ (“since”). However, adding past-time operators here does not extend the expressive power of the language and, as we will see in Section 3, such operators can all be removed by translation of arbitrary temporal formulae into a specific normal form.

Finally, we add a nullary operator ‘**start**’, which is only satisfied at the “beginning of time”:

$$\langle \sigma, i \rangle \models \textbf{start} \quad \text{iff} \quad i = 0$$

Next we will motivate our choice of temporal logic as the basis for our formal framework.

2.2. *Why temporal logic?*

Temporal logic allows the concise expression of useful dynamic properties of individual agents. For example, the formula

$$request \Rightarrow reply\,\mathcal{U}\,acknowledgement$$

characterises a system where, once a *request* is received, a *reply* is continually sent up until the point where an *acknowledgement* is received.

In addition, pre-conditions, such as

$$\blacksquare\neg started \Rightarrow \bigcirc\neg moving$$

can easily be described in this logic.

As the temporal model on which the logic is based comprises a linear sequence of moments, then the logic can also be used to express the *order* in which activities occur, for example

$$hungry \Rightarrow (buy\_food \wedge \bigcirc cook\_food \wedge \bigcirc\bigcirc eat).$$

While the logic is clearly useful for representing the dynamic activity of individual agents, and this relates very closely to traditional applications of temporal logic in program specification [51], the formalism is also useful for characterising properties of the multi-agent system itself. For example, the formula

$$broadcast(msg) \Rightarrow \forall a \in Agents.\ \Diamond receive(msg, a)$$

describes the message-passing behaviour within a group of agents (characterised by the finite set *Agents*). Thus, temporal logic can be used to represent both the internal behaviour of an agent and the macro-level behaviour of groups of agents.

A further aspect of temporal logic, which turns out to be particularly useful when specifying agent-based systems, is the '$\Diamond$' operator (meaning "at some point in the future"). For example, we can view formulae such as $\Diamond rich$ as designating atoms (in this case, *rich*) that the system must *eventually* satisfy. Essentially, formulae such as these can be seen as representing the *goals* of the agent.

In addition, we can also specify goal pre-conditions. For example, the formula

$$(\neg have\_passport \vee \neg have\_ticket) \Rightarrow \bigcirc \neg board\_aircraft$$

expresses the pre-condition that $\Diamond board\_aircraft$ will not be satisfied until *have _passport* and *have_ticket* are both satisfied. As we will see in Sections 5 and 6, this allows us to represent an agent in terms of its basic dynamic activity, together with both the goals that it is trying to satisfy and their pre-conditions.

3. Individual agents: The temporal framework

We will now consider how, using a purely temporal framework, we can specify, verify, refine and implement individual agents. We begin by considering the formal representation of agents in the temporal logic described above.

3.1. Specification

The specification of an agent's behaviour is given as a temporal formula, then transformed into a simple normal form, called Separated Normal Form (SNF) [21]. In this normal form, the majority of the temporal operators are removed, and formulae are represented as

$$\Box \bigwedge_{i=1}^{n} R_i$$

where each R_i, termed a *rule*, is in one of the following forms.

$$\textbf{start} \Rightarrow \bigvee_{b=1}^{r} l_b \qquad \text{(an } \textit{initial} \text{ rule)}$$

$$\bigwedge_{a=1}^{g} k_a \Rightarrow \bigcirc \left[\bigvee_{b=1}^{r} l_b \right] \qquad \text{(a } \textit{step} \text{ rule)}$$

$$\bigwedge_{a=1}^{g} k_a \Rightarrow \Diamond l \qquad \text{(a } \textit{sometime} \text{ rule)}$$

Note, here, that each k_a, l_b, or l is a literal.

This normal form gives a simple and intuitive description of what is true at the beginning of execution (via initial rules), what must be true during any execution step (via step rules), and what constraints exist on future execution states (via sometime rules). For example, the formulae below correspond to the three different types of SNF rules.

$$\begin{array}{ll} \underline{\text{INITIAL}}: & \textbf{start} \Rightarrow sad \vee optimistic \\ \underline{\text{STEP}}: & (sad \wedge \neg optimistic) \Rightarrow \bigcirc sad \\ \underline{\text{SOMETIME}}: & optimistic \Rightarrow \Diamond \neg sad \end{array}$$

Once we have such a representation, we can simply extend this to give the behaviour of an individual agent by providing a definition of the agent's interface with its environment [18]. Such an interface simply describes the messages that the agent can receive (i.e. those that come `in`) and send (i.e. those that go `out`). For example, consider a "car" agent which can be told to *go, turn* and *stop*, but can also notify other agents that it has run out of fuel *(empty)* or has overheated *(overheat)*. The (partial) definition of such an agent can be given as follows.

$$\begin{array}{rl} \texttt{car}() & \\ \texttt{in}: & go, stop, turn \\ \texttt{out}: & empty, overheat \\ \texttt{rules}: & \textbf{start} \Rightarrow \neg moving \\ & (moving \wedge go) \Rightarrow \bigcirc(overheat \vee empty) \\ & go \Rightarrow \Diamond moving \end{array}$$

This represents an agent, named `car`, that recognises the messages *go*, *stop* and *turn* and can potentially send out *empty* and *overheat* messages. The behaviour of the agent is then specified by the temporal formula

$$\Box \left[\begin{array}{r} \textbf{start} \Rightarrow \neg moving \\ \wedge \\ (moving \wedge go) \Rightarrow \bigcirc(overheat \vee empty) \\ \wedge \\ go \Rightarrow \Diamond moving \end{array} \right]$$

As we will see in Section 4, both the messages received and the messages sent are interpreted as propositions (or predicates) that can be used as part of the agent's specification. In addition, standard propositions (predicates) appear in the specification, for example *moving* in the description above. These atoms are essentially internal and do not directly correspond to communication activity by the agent or its environment.

3.2. *Refinement*

Refinement of an agent's specification can be carried out in a standard way [51]. Thus, a new specification for an agent, S_2, is a refinement of the original specification. S_1, if the following proof obligation is satisfied:

$$\vdash S_2 \Rightarrow S_1.$$

This means that any system satisfying the new specification (S_2) will also satisfy the original specification (S_1).

For example, given the specification of the `car` agent above (call it S_1), then a refined specification can be given as follows (call it S_2).

$$\Box \left[\begin{array}{rcl} \textbf{start} & \Rightarrow & \neg moving \\ & \wedge & \\ (moving \wedge go) & \Rightarrow & \bigcirc overheat \\ & \wedge & \\ go & \Rightarrow & \bigcirc \bigcirc moving \end{array} \right]$$

In S_2 some of the non-determinism within S_1 has been removed. In particular, if *moving* and *go* are both true then *overheat* will become true in the next state. In addition, if *go* becomes true then we are certain, not just that *moving* will become true at some time in the future, but that it will become true in exactly two steps time. Thus, S_2 is a refinement of S_1 and $\vdash S_2 \Rightarrow S_1$ can be proved.

While the refinement of individual agent specifications is clearly useful, it is common in agent-based systems to refine a single agent specification into a specification for a group of agents that together achieve appropriate behaviour. We will consider this, more complex, form of refinement in Section 4.2.

3.3. Verification

Next, we consider the verification of our agent specifications, which, in addition to being useful for checking the properties of agents, is also required in order to support the types of proof obligations generated during agent refinement. The verification of the temporal properties of individual agents can be carried out in a number of ways. For systems where finite-state models are available, then model-checking is often used [41]. There are many scenarios in which model-checking techniques may be appropriate [37] but, as we are manipulating logical specifications, we choose to use a particular proof technique for temporal logic based upon clausal resolution [16, 26]. While this approach has been shown to be relatively efficient and easy to mechanise [10], it has the added advantage that it is based upon *exactly* the same normal form as we use in our specifications [21].

Thus, in order to prove the validity of a temporal formula, say φ, we negate the formula, giving $\neg\varphi$, translate it into a set of SNF rules, and attempt to derive a contradiction using specific resolution rules. Recall that SNF comprises three different types of rule: initial rules, step rules and sometime rules. The resolution method we have developed ensures that

1. *initial resolution* occurs between initial rules,
2. *step resolution* occurs between step rules, and,
3. *temporal resolution* occurs between one sometime rule and a *set* of step rules.

The three varieties of resolution operation that act upon SNF rules are simply

$$\text{INITIAL RESOLUTION:} \quad \frac{\begin{array}{l} \textbf{start} \Rightarrow A \vee l \\ \textbf{start} \Rightarrow B \vee \neg l \end{array}}{\textbf{start} \Rightarrow A \vee B}$$

$$\textsc{Step Resolution:}\quad \frac{\begin{array}{c} C \Rightarrow \bigcirc(A \wedge l) \\ D \Rightarrow \bigcirc(B \wedge \neg l) \end{array}}{(C \wedge D) \Rightarrow \bigcirc(A \vee B)}$$

$$\textsc{Temporal Resolution:}\quad \frac{\begin{array}{c} C \Rightarrow \Diamond l \\ D \Rightarrow \bigcirc\Box\neg l \end{array}}{C \Rightarrow (\neg D)\mathcal{W}l}$$

Note that the temporal resolution operation is actually applied to a set of step rules that together characterise $D \Rightarrow \bigcirc\Box\neg l$ (this formula itself is not in SNF) and that the resolvent produced from this operation must still be translated into SNF.

Rather than go through this resolution method in detail, we will follow through a simple example derived from [26].

3.3.1. Example Resolution Refutation. Assume we wish to show that the following set of SNF rules is unsatisfiable.

1. $\textbf{start} \Rightarrow f$
2. $\textbf{start} \Rightarrow goes\_fast$
3. $\textbf{start} \Rightarrow danger$
4. $f \Rightarrow \Diamond\neg danger$
5. $f \Rightarrow \bigcirc goes\_fast$
6. $goes\_fast \Rightarrow \bigcirc(overheats \vee coolant)$
7. $overheats \Rightarrow \bigcirc goes\_fast$
8. $overheats \Rightarrow \bigcirc danger$
9. $goes\_fast \Rightarrow \bigcirc danger$
10. $goes\_fast \Rightarrow \bigcirc\neg coolant$

The original specification represented the behaviour of a (not very efficient) `car` agent. The key elements of the specification are that

– if the car *goes_fast* then, in the next moment, it either *overheats* or will be supplied with *coolant* (represented as rule 6 above),
– if the car *overheats* then, in the next moment, it will *goes_fast* in order to try to cool the engine (represented as rule 7 above),
– if the car either *overheats* or *goes_fast*, then there is *danger* in the next moment (represented as rules 8 and 9 above),
– if the car *goes_fast*, then *coolant* can not be supplied in the next moment (represented as rule 10 above), and
– initially we know that there is *danger*, that the car *goes_fast* in the second moment in time and that there will be some moment in the future when there will be no *danger* (represented as rules 1, 2, 3, 4, and 5 above).

Note also that f is a new proposition symbol introduced as part of the process of translating to SNF [21].

Now, as we wish to show that this specification is unsatisfiable, we begin with step resolution as follows.

11. $goes\_fast \Rightarrow \bigcirc overheats$ [6, 10 Step Resolution]

In order to apply the temporal resolution operation, we must identify a set of step rules that characterise the appropriate $\Box$-formula that we require (note that, efficient techniques for carrying out this search have been considered in [10]). By combining rules 7, 8, 9, and 11, we can identify

$$(goes\_fast \vee overheats) \Rightarrow \bigcirc(danger \wedge (goes\_fast \vee overheats))$$

which means that, once $goes\_fast \vee overheats$ is satisfied, then *danger* will be satisfied at every state in the future, i.e.

$$(goes\_fast \vee overheats) \Rightarrow \bigcirc\Box danger.$$

Applying the temporal resolution operation to this and rule 4, we derive

$$f \Rightarrow (\neg(goes\_fast \vee overheats))\mathcal{W}\neg danger$$

which, when translated to SNF, gives (amongst others) 12:

12. **start** $\Rightarrow \neg f \vee \neg goes\_fast \vee \neg danger$ [4, 7, 8, 9, 11 Temporal Resolution]
13. **start** $\Rightarrow \neg f \vee \neg goes\_fast$ [3, 12 Initial Resolution]
14. **start** $\Rightarrow \neg f$ [2, 13 Initial Resolution]
15. **start** $\Rightarrow$ **false** [1, 14 Initial Resolution]

Hence, as a contradiction is obtained, the original set of rules is unsatisfiable. (The completeness result for this form of temporal resolution is described in detail in [26].)

3.4. Execution

Once we have the temporal specification of an agent, as given in Section 3.1, an implementation can be developed in a number of ways, for example refinement to a standard programming language [53] or automatic synthesis of an automaton [57]. However, the route we choose is to directly execute the temporal specification in order to provide an implementation [20].

This approach follows the *imperative future* paradigm [2], and applies an iterative *forward-chaining* process to a set of SNF rules in order to construct a model for the specification. Since the model is a linear, discrete sequence of states, and since forward chaining is applied, model construction in this way mimics execution in more standard programming languages [18]. This approach is captured in the

METATEM programming language [1], and can be utilised here to animate agent specifications.

Recall the '`car`' specification given above, where the behaviour of the agent was specified by the three rules

$$\begin{aligned} \textbf{start} &\Rightarrow \neg moving \\ (moving \wedge go) &\Rightarrow \bigcirc(overheat \vee empty) \\ go &\Rightarrow \Diamond moving \end{aligned}$$

Execution using the METATEM approach involves forward chaining via step rules from initial rules, while constraining the execution in an attempt to satisfy sometime formulae (e.g. $\Diamond moving$). Thus, in the above example, *moving* is false at the beginning of time and whenever *moving* and *go* are both true together, then either *overheat* or *empty* will be made true in the next moment in time. Finally, whenever *go* is true a commitment to eventually make *moving* true is given.

The underlying mechanism here is relatively straightforward, with the forward chaining through step rules attempting to construct a model. If a disjunction, such as $overheat \vee empty$ above is executed, a choice must be made. If this choice leads to a contradictory situation, then backtracking occurs and an alternative choice is made.

The main complication (and the expressive power) comes from execution of $\Diamond$-formulae (or eventualities). When a formula such as '$\Diamond\varphi$' is executed, the system must attempt to ensure that φ *eventually* becomes true. As such eventualities might not be able to be satisfied immediately, a record of the outstanding eventualities must be kept, so that they can be re-tried as execution proceeds. The standard heuristic used is to attempt to satisfy as many eventualities as possible, starting with the oldest outstanding eventuality [33].

An important result concerning execution of propositional SNF rules is that, if the above execution procedure is followed, then a model (an execution sequence) will be constructed if, and only if, the formula (specification) is satisfiable [1]. In this respect, propositional METATEM is a complete theorem-prover for propositional temporal logic. However, even though METATEM has this property, it is rarely used in this way. In particular, as soon as we allow environment interaction and extend the logic to first-order temporal logic, which is itself generally incomplete, then the execution mechanism becomes incomplete anyway.

4. Multiple agents: The temporal framework

We now turn to the specification, refinement, verification and implementation of *multi-agent* systems. This extends our approach to incorporate communication, concurrency and the representation of the local state of distributed agents.

4.1. Specification

Agents within a multi-agent system are specified in the same way as we specified individual agents in Section 3.1. However, now we must additionally consider both

the environment in which agents are situated and any interaction the agents might have. Recall the '`car`' example:

$$
\begin{array}{rl}
\texttt{car}() & \\
\texttt{in}: & go, stop, turn \\
\texttt{out}: & empty, overheat \\
\texttt{rules}: & \\
 & \textbf{start} \Rightarrow \neg moving \\
 & (moving \wedge go) \Rightarrow \bigcirc(overheat \vee empty) \\
 & go \Rightarrow \Diamond moving
\end{array}
$$

We now note that *go, stop*, and *turn* are all atoms that can be referred to in the agent's rules, but whose values are supplied by the agent's environment; these are called *environment propositions* and they are made true once a message of the same name is received. The propositions *empty* and *overheat* can also occur in the agent's rules and their values are controlled directly by the agent; these are called *component propositions*. However, once such a proposition is made true, a message of the same name is sent to the agent's environment. In this way, communication is incorporated seamlessly into the logical interpretation of the agents.

The key component that provides clear and simple interaction between agents is the use of *broadcast* message-passing [17]. Thus, when an agent makes one of its component propositions true, the corresponding message is broadcast to all other agents. This mechanism not only provides a simple method of communication for use in open systems, but presents a close link between execution and distributed deduction [22].

A variety of constraints can be specified concerning concurrency and communication within the multi-agent system, each of which is again reflected in both the semantics and implementation. For example,

$$broadcast(msg) \Rightarrow \forall a \in Agents.\ \Diamond receive(msg, a)$$

specifies that, once a message is broadcast, it will *eventually* be received by all the other agents. In addition, agents can execute either synchronously or asynchronously with respect to each other. The choice between these affects the complexity of both the semantics and execution mechanism (see below).

Once we move to a multi-agent scenario, the semantics of our system becomes more complex. If we consider '$[\![\ ]\!]$' to represent a function providing temporal semantics, then the behaviour of a system of multiple (in this case, two) communicating agents, is typically given by

$$[\![ag_1 \| ag_2]\!] = [\![ag_1]\!] \wedge [\![ag_2]\!] \wedge comms(ag_1, ag_2).$$

where $comms(ag_1, ag_2)$ is a temporal specification of the communication properties, such as that given for broadcast message-passing above. However, we must also be careful to distinguish the information that ag_1 deals with from that which ag_2 deals with. Consequently, $[\![ag_1]\!]$ is *not* just the temporal rules contained within the agent ag_1 and so we must enhance these rules in some way in order to capture the multiplicity of agents.

There are several mechanisms that have been used to give such semantics; three of these are described in the following sections.

4.1.1. Purely Temporal Semantics. An obvious way to ensure that, for example, proposition p in agent ag_1 is distinguished from p in ag_2, is to rename the propositions in the agent's rules ensuring that each one is 'tagged' by the name of the agent in which it occurs. Thus, we would have propositions such as p_{ag_1} and p_{ag_2}; this is exactly the approach used in [19].

A further important aspect of the semantics given in [19] is that it allows for *either* synchronous or asynchronous models of concurrency within the framework. In the case of synchronously executing agents, the semantics of each agent is given as a (tagged) temporal formula in a discrete, linear temporal logic of the type described in Section 2. However, once we consider asynchronously executing agents, the semantics is given as a formula in the Temporal Logic of the Reals (TLR) [3], which is a temporal logic based upon the Real, rather than Natural, Numbers. The density of this Real Number model is useful in representing the asynchronous nature of each agent's execution.

4.1.2. Temporal Logic of Belief. A different approach to characterising the information that each agent has is to represent this information in terms of the agent's *beliefs*. Thus, for example, if p is satisfied in agent ag_1's computation, then we can assert that ag_1 believes p. By extending our basic temporal logic with a multi-modal KD45 dimension representing belief, we can represent this as $B_{ag_1}p$ [69], i.e. that agent ag_1 believes p. Importantly, the truth of, for example, $B_{ag_1}\ p$ is distinct from that of $B_{ag_2}p$. Such a logical formulation is used in [34] to provide a semantics for agents of the above form.

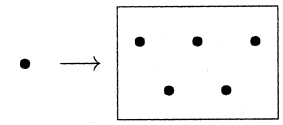
4.1.3. Temporal Logic of Knowledge. An alternative, but related, approach to the representation of information within distinct agents is to use the abstraction of *knowledge*, rather than belief. Thus, by extending our temporal logic with a multi-modal S5 logic, rather than the multi-modal KD45 logic used to characterise belief, we produce a *temporal logic of knowledge* [13, 15]. Such a logic is used, in [70], in order to give a knowledge-theoretic semantics of the types of systems we are considering. While there are (at least) these three options, the choice between them is, in reality, often down to what form of logic the specifier wishes to utilise. In addition, potential complexities surface, with the first approach being the simplest and the last having the largest danger of added complexity.

4.2. Refinement

Once we address multi-agent scenarios, a more general variety of refinement can be considered. This involves refining one multi-agent system (consisting of $a_1, a_2, \ldots, a_n$) into a different system (consisting of $b_1, b_2, \ldots, b_m$). The corresponding proof obligation is

$$[\![b_1 \| b_2 \| \ldots \| b_m]\!] \Rightarrow [\![a_1 \| a_2 \| a \| \ldots \| a_n]\!]$$

A particularly interesting instance of this is where $n = 1$, and so a single agent is being refined into a group of agents with comparable behaviour, e.g.



Such development is particularly common, for example, in distributed problem-solving where we often require a set of agents to achieve, in a distributed manner, some goal that was originally undertaken by a single agent.

For example, consider a single agent that is specified to be able to *buy_product*, i.e. with specification (termed S_1)

$$\textbf{start} \Rightarrow \Diamond buy\_product.$$

We now wish to refine this single agent specification into three separate specifications, representing agents that can select a product (agent 1), get funds (agent 2) and carry out the purchase (agent 3), respectively. These can be specified simply as follows.

1. $\textbf{start} \Rightarrow \Diamond select\_product$
2. $have\_product \Rightarrow \Diamond get\_funds$
3. $have\_funds \Rightarrow \Diamond buy\_product$

Each of these agents then implements its behaviour and, as long as we can be sure that communication occurs appropriately, then the combination of these agents implements $\Diamond buy\_product$. Typical communication constraints are that if a message is sent from one agent then it will eventually be received at the destination agent. For the above, we might have

$$\begin{aligned} comms(1,2): &\quad select\_product \Rightarrow \Diamond have\_product \\ comms(2,3): &\quad get\_funds \Rightarrow \Diamond have\_funds \end{aligned}$$

Thus, the combination of these communication constraints, together with the three agent specifications, gives the specification, S_M of the multi-agent system:

$$\begin{aligned} &\Box(\textbf{start} \Rightarrow \Diamond select\_product) \wedge \\ &\Box(have\_product \Rightarrow \Diamond get\_funds) \wedge \\ &\Box(have\_funds \Rightarrow \Diamond buy\_product) \wedge \\ &\Box(select\_product \Rightarrow \Diamond have\_product) \wedge \\ &\Box(get\_funds \Rightarrow \Diamond have\_funds) \end{aligned}$$

Then we can prove

$$\vdash S_M \Rightarrow S_I$$

and so the second system (S_M) is a refinement of the first (S_I).

While the approach outlined above follows a standard mechanism for refinement based upon discharging proof obligations, we have also investigated the use of a fixed set of transformations that can be used for developing multi-agent systems within our framework [24]. These transformations involve syntactic modifications, such as splitting an agent's rules into two agents, altering the agent's interface definition, or adding auxiliary rules. Each of these transformations is guaranteed to preserve the behaviour of the original system. Thus, by using a restricted set of refinements, characterised by these transformations, we can avoid the need to carry out refinement proofs. Investigating the extent to which such transformations can be used in the full, and realistic, development of multi-agent systems is part of our future research.

4.3. Verification

As we have seen in Section 4.1, the semantics of a set of communicating agents in our framework can be given via a number of approaches. For each of these varieties, we have appropriate proof methods, all based upon the temporal resolution approach introduced in Section 3.3.

Once we have a 'tagged' semantics of the type described in Section 4.1.1, then we have a choice of whether to produce a discrete temporal logic formula (representing synchronous concurrency) or a TLR formula (representing asynchronous concurrency). If a discrete temporal formula is produced, then the temporal resolution method described in Section 3.3 can be applied directly. If a TLR formula is produced, then we can utilise a result from [49] and translate this formula into discrete temporal logic (and so again use our resolution method).

For the semantics given in Sections 4.1.2 and 4.1.3, based upon the extension of temporal logic with various modalities [4], there are several potential proof methods, such as [15, 67]. However, our temporal resolution method can also be extended to handle such multi-modal extensions of temporal logic. Again, rather than duplicating the detailed descriptions given in [12, 13, 35], we will simply give an overview of the approach. The basic idea, in extending the clausal resolution method to combinations of modal and temporal logics, is to

1. separate the modalities within an extended form of SNF, and
2. define resolution rules for each of the different modalities.

Thus, if we consider that the basic form of SNF, defined in Section 3.1, being represented as the template

$$\begin{aligned} &\textbf{start} \Rightarrow \vee \ldots \\ &\wedge \ldots \Rightarrow \bigcirc \vee \ldots \\ &\wedge \ldots \Rightarrow \Diamond \ldots \end{aligned}$$

then SNF for temporal logics extended with modalities M_1, M_2, etc. could be represented as

$$
\begin{array}{ll}
\textbf{start} & \Rightarrow \vee \ldots \\
\wedge \ldots & \Rightarrow \bigcirc \vee \ldots \\
\wedge \ldots & \Rightarrow \Diamond \ldots \\
\wedge \ldots & \Rightarrow \mathsf{M}_1 \vee \ldots \\
\wedge \ldots & \Rightarrow \neg \mathsf{M}_1 \neg \vee \ldots \\
\wedge \ldots & \Rightarrow \mathsf{M}_2 \vee \ldots \\
\wedge \ldots & \Rightarrow \neg \mathsf{M}_2 \neg \vee \ldots \\
\vdots & \quad \vdots
\end{array}
$$

Thus if, for example, we have a temporal logic of knowledge (with an S5 knowledge modality, K), the normal form for this logic could be:

$$
\begin{array}{ll}
\textbf{start} & \Rightarrow \vee \ldots \\
\wedge \ldots & \Rightarrow \bigcirc \vee \ldots \\
\wedge \ldots & \Rightarrow \Diamond \ldots \\
\wedge \ldots & \Rightarrow \mathsf{K} \vee \ldots \\
\wedge \ldots & \Rightarrow \neg \mathsf{K} \neg \vee \ldots
\end{array}
$$

Now, while the step and temporal resolution operations apply to the temporal rules, modal resolution applies to modal rules, for example

$$
\frac{\begin{array}{r} C \Rightarrow \mathsf{K}(A \vee l) \\ D \Rightarrow \mathsf{K}(B \vee \neg l) \end{array}}{(C \wedge D) \Rightarrow \mathsf{K}(A \vee B)}
$$

and

$$
\frac{\begin{array}{r} C \Rightarrow \mathsf{K}(A \vee l) \\ D \Rightarrow \neg\mathsf{K}\neg(B \vee \neg l) \end{array}}{(C \wedge D) \Rightarrow \neg\mathsf{K}\neg(A \vee B)}
$$

There are a variety of resolution rules characterising the specific properties of the modal extension used, although the exact form of rule is more complex than the version given above [13]. In addition, in [43], a *translation* approach is used for the modal dimensions, whereby modal formulae are translated to classical first-order logic and classical resolution is carried out [55].

The link between the modal and temporal dimensions is provided by the new propositions introduced during renaming. As with most research into proof methods for combinations of logics, our current work concerns the flexible and efficient (if possible) representation of interactions between logical dimensions [11].

4.4. Execution

Just as the METATEM programming language was used to execute individual agent specifications, the Concurrent METATEM programming language [17] allows the

execution of multi-agent specification. Concurrent METATEM comprises two elements: the representation and implementation of an individual agent's behaviour using the mechanism described in Section 3.4; and an operational framework providing both concurrency and communication.

Within the operational model, agents execute independently and asynchronously. As defined in the semantics above, *broadcast* message-passing is central, providing a link between the individual agent execution and the multi-agent operation. Rather than describe the operational model in detail, we will give a simple example of a multi-agent program derived from one in [17].

4.4.1. Example Concurrent METATEM Program. This example concerns three agents representing professors bidding to a funding body (again represented by an agent) for resources. Each professor's behaviour is summarised by their names: `eager`; `mimic`; and `jealous`. The actual definitions of each agent are given below (note that we use predicates over a finite domain, rather than propositions, in order to shorten this description and that X and Y represent universally quantified variables over this domain).

`funder()`	
`in:`	*apply*
`out:`	*grant*
`rules:`	$grant(X) \wedge grant(Y) \Rightarrow X = Y$
	$apply(X) \Rightarrow \Diamond grant(X)$
`eager()`	
`in:`	
`out:`	*apply*
`rules:`	**start** $\Rightarrow apply(eager)$
	true $\Rightarrow \bigcirc apply(eager)$
`mimic()`	
`in:`	*apply*
`out:`	*apply*
`rules:`	$apply(eager) \Rightarrow \bigcirc apply(mimic)$
`jealous()`	
`in:`	*grant*
`out:`	*apply*
`rules:`	$grant(eager) \Rightarrow \bigcirc apply(jealous)$

The basic idea here is that each of the three professor agents may send an *apply* message to the `funder` in an attempt to secure a grant. These agents send their own name as the only argument to this message. The `funder` agent must then decide which applications to grant, given that it is only allowed to grant to at most one professor at any moment in time.

This simple example exhibits both safety and liveness rules in `funder`, ensuring that at most one *grant* is made at every moment (safety) and that an *apply* will eventually be fulfilled (liveness). This example also shows the power of broadcast communication as not only can several agents send and receive the same message, but the behaviour of one agent can be defined to be dependent on messages sent to/from others (for example, `mimic` and `jealous` both base their behaviour on `eager`'s communications).

5. Individual agents: Reasoning about belief

In the previous two sections we have described our framework for specifying, verifying, refining and implementing both individual agents and multi-agent systems. We now consider ways in which this framework has been extended in the case of individual agents.

5.1. *Specification*

As we saw earlier, the semantics of a multi-agent system can be given in terms of a temporal logic, extended with modal dimensions of either knowledge or belief. However, the agent descriptions were provided in a purely (discrete, linear) *temporal* logic. As described above, our logical basis can be extended with standard modal logics, in particular a modal logic (typically KD45) of belief. SNF, extended in this way, looks like

$$
\begin{aligned}
&\textbf{start} \Rightarrow \mathit{vee} \ldots \\
&\wedge \ldots \Rightarrow \bigcirc \vee \ldots \\
&\wedge \ldots \Rightarrow \Diamond \ldots \\
&\wedge \ldots \Rightarrow \mathsf{B}_1 \vee \ldots \\
&\wedge \ldots \Rightarrow \neg \mathsf{B}_1 \neg \vee \ldots \\
&\wedge \ldots \Rightarrow \mathsf{B}_2 \vee \ldots \\
&\wedge \ldots \Rightarrow \neg \mathsf{B}_2 \neg \vee \ldots \\
&\quad . \qquad\qquad . \\
&\quad . \qquad\qquad . \\
&\quad . \qquad\qquad .
\end{aligned}
$$

where B_i is a belief modality meaning "agent i believes". Again, arbitrary specifications can be transformed into this normal form, thus giving a clear basis for specifying dynamic agents that can reason about their (and other agent's) beliefs [12].

5.2. *Verification*

Clearly, once an agent specification is transformed to our normal form, then we can apply our resolution rules to this combined temporal logic of belief. Indeed, we have

used such an approach to provide a basis for verification of the powerful KARO framework [42] in which agents can reason about information dynamically. However, combined modal and temporal logics of this form can be complex [15] and we are working on improving efficiency of the proof process wherever possible. This takes two forms. Firstly, we are considering a hybrid proof apprach that uses translation methods [55] for the modal fragment of these logics and clausal resolution for the temporal fragment [43], thus giving the prospect of improved underlying efficiency. Second, we are examining the areas where high complexity typically occurs, in particular *interaction axioms*. Such axioms enforce a close interaction between the modal and temporal dimensions and often lead to severe computational problems. Thus, we are extending our resolution method to seamlessly handle a variety of standard interactions, beginning with those studied in temporal logics of knowledge [11].

5.3. *Execution*

While we can, in principle, execute agent specifications given in the above temporal logics of belief, this, as we have suggested, can be quite expensive. In addition, we would like to be able to limit the amount of reasoning that a logic-based agent could carry out in certain situations. Thus, in more recent work we have investigated the extension of the agents' descriptions to incorporate an explicit multi-context belief dimension [27]. In many ways, such multi-context logics of belief are similar to standard modal logics of belief, the key difference being that we can closely control the number of contexts (and, so, depth of reasoning) allowed.

The syntax of this belief extension is the same as that given above and, again, arbitrary specifications can be transformed into our normal form, with the key addition of a bound on the depth of nesting of B operators allowed. This logic is investigated in [27] and part of a travel assistant example (again shortened by using predicates) taken from that paper is

$$ask(you, X) \Rightarrow \mathsf{B}_{me}\Diamond go(you, X).$$

This is interpreted as "if you ask for information about destination X, then I believe that you will go to X in the future". (Here, B_{me} is a multi-context belief operator.) Thus, such a logical extension allows each agent to be represented in a logic incorporating belief and provides the agent itself with a mechanism for reasoning, in a resource-bounded way, about its (and other agents') beliefs.

As we saw above, the normal form extends that of basic temporal logic with B_i and $\neg\mathsf{B}_i\neg$ rules characterising belief, where 'B_i' is now a multi-contextual belief modality with standard KD45 properties. To simplify this further, we add the axiom

$$\vdash \neg\mathsf{B}_i\neg\varphi \Rightarrow \mathsf{B}_i\varphi$$

which ensures that each agent has beliefs about everything. This reduces the normal form to

$$
\begin{array}{ll}
\textbf{start} \Rightarrow \vee \ldots & \\
\wedge \ldots \Rightarrow \bigcirc \vee \ldots & \\
\wedge \ldots \Rightarrow \Diamond \ldots & \\
\wedge \ldots \Rightarrow \mathsf{B}_1 \vee \ldots & \\
\wedge \ldots \Rightarrow \mathsf{B}_2 \vee \ldots & \\
\cdot & \cdot \\
\cdot & \cdot
\end{array}
$$

We can now extend execution to this variety of normal form, thus allowing resource-bounded reasoning about beliefs [27]. This involves executing the normal form for temporal logic combined with a multi-context logic of belief. Formulae are thus represented by exploring a belief model at every moment during the temporal execution.

Further, the ability to dynamically modify the depth bound on reasoning about beliefs [29] leads to many potential applications, one of which is considered in [28], There, a RoboCup agent is specified using our logic in such a way that it must reason about its beliefs in order to decide whether to ''pass'' or ''shoot''. Allowing a large amount of reasoning enables the agent to infer that it should pass; limiting reasoning leads the agent to shoot.

6. Individual agents: Deliberation

6.1. Execution

We have been concerned, in examining extensions of agent execution, to try to capture the *deliberative* aspects of agents. We choose to carry this out via a simple re-interpretation of certain logical formulae, together with a modified process for ensuring their satisfaction.

Recall that we can view formulae such as $\Diamond rich$ as goals of the agent, and that the basic execution mechanism (see Section 3.4) uses a heuristic that attempts to satisfy as many goals as possible, starting with the oldest outstanding one [33]. In order to give the system designer more control over the order in which goals are achieved (and attempted), in [23] we replaced this fixed heuristic by user definable priority functions. Thus, at each moment in its execution, the agent uses these priority functions to reorder its list of outstanding goals (i.e. formulae of the form $\Diamond\varphi$ where φ has yet to be satisfied) and then begins attempting to satisfy these goals based upon the re-ordered list. Consequently, the agent is able to react to changing situations by ordering the goals it must achieve depending upon the current requirements.

One of the benefits of this approach is that it allows us to characterise a variety of BDI-like systems. By viewing both desires and intentions as eventualities, and adding a belief component (as in Section 5), we can see the above execution as a (very) simplified version of BDI deliberation. Desires represent statements that must eventually be satisfied; intentions represent statements that the system is *actively* trying to satisfy. Thus, we can characterise goals at the front of the list of outstanding goals as intentions and those nearer the end as desires. Deliberation is

then the mechanism by which the agent decides how to make desires into intentions (i.e. ordering functions) and how to implement intentions [23].

7. Multiple agents: Groups and teams

7.1. *Specification*

While our main concern with respect to multi-agent systems has been to develop the execution mechanism further, this has also involved extending specification techniques. Thus, rather than representing a multi-agent system simply as a set of agents of the above form, we have developed various mechanisms for structuring the agent space based upon *agent grouping*. Groups are entities that contain both agents and other groups, and can be overlapping, dynamic, open, etc. These groups restrict the extent of broadcast communication, and so allow the representations of societies of agents within the overall system. A key aspect of the representation and implementation of such systems is to be able to specify and execute constraints concerning the internal characteristics of a group. For example, while one group may involve synchronously executing agents, another might involve asynchronously executing agents communicating in a different way. An example, taken from [25], is that communication may be specified by one of

- $send(msg) \Rightarrow \forall a \in Agents.\ \Diamond receive(a, msg)$
- $send(msg) \Rightarrow \forall a \in Agents.\ receive(a, msg)$
- $send(msg) \Rightarrow \forall a \in Agents.\ \bigcirc receive(a, msg)$
- $send(msg) \Rightarrow \forall a \in Agents.\ \Diamond receive(a, msg) \ \vee \ lost(a, msg)$
- etc...

This provides the system designer with powerful abstractions that can be used to represent complex local phenomena within multi-agent systems, and has been particularly targeted at agent architectures [25].

7.2. *Verification*

With more complex properties of multi-agent systems being specified, it is clearly important to extend the verification methods further. In particular, we have extended the temporal basis of our resolution method to *branching* temporal logics [5], and are continuing work on extending the approach to more powerful logics, such as the temporal μ-calculus [65] and first-order temporal logic [9, 14]. In addition, our future work will examine modal, μ-calculus, which allows the representation of fixpoints in modal logic and so can capture concepts such as *common knowledge*.

7.3. *Execution*

Our recent work in this area has concerned the development of a powerful, but logically well-founded, model for execution within this extended framework [30].

This treats groups and agents identically, with an agent's contents being the empty set and a group's rules characterising the global temporal behaviour of the set of agents it contains. This work is being extended in order to characterise more complex constructs such as teams and organisations, within this framework. For example, in [31], we indicate how dynamic team formation, and the development of joint beliefs, can be achieved in this powerful model.

The view of computation as agents (or objects), in groups, communicating through broadcast message-passing is very appealing. It is particularly useful for dynamic, open, and complex systems. As well as the group structuring aspects mentioned above, we are concerned with the dynamic evolution of systems. In particular, we have introduced a mechanism for agent creation, based on *cloning* [32], and are developing the basis for agent migration [46, 47]. As an example of cloning, consider a modified version of the `car` agent given earlier:

$$
\begin{array}{rl}
\texttt{car(Colour,Cost)} & \\
\texttt{in}: & \mathit{go}, \mathit{stop}, \mathit{turn} \\
\texttt{out}: & \mathit{empty}, \mathit{overheat} \\
\texttt{rules}: & \mathit{price}(\texttt{Cost}) \Rightarrow \bigcirc \mathit{details}(\texttt{Cost}, \texttt{Colour})
\end{array}
$$

Once this agent is executing, invocation of *clone*(*red*, 5000) will make an independent copy of the car agent and commence its execution. In particular, `Colour` and `Cost` will be bound to *red and 5000*, respectively, in the new agent.

8. Concluding remarks

We have described our work on developing formal methods for agent-based systems, which has comprised a basic temporal approach, extended with various aspects of agency and multi-agent environments. This, together with extensions to handle more complex agents and systems, provide an coherent view of the formal specification, verification, refinement and implementation of agent-based systems. In particular, for relatively simple agents, we have a complete framework for developing both individual agents and multi-agent systems.

8.1. Related work

While there have been many agent theories developed, several formal methods for agent-based systems and a number of high-level programming languages for rational agents, there have been few complete frameworks comprising specification, verification, refinement and implementation. Three approaches which have many of these aspects are represented by the work on BDI, KARO and Golog. As mentioned above, BDI is the predominant rational agent theory [68], and has been very influential. However, there is little direct connection between the BDI theory, in terms of modal and temporal logics [59] and the BDI architecture [61], although some progress has been made in bridging this gap [60] and higher-level languages have been devised [58]. The KARO framework [63] is based on dynamic logic [38], extended

with knowledge, belief and wishes. Agent-based systems have been specified and verified [44] using this approach and a range of programming languages have been developed with connection to this theory, including 3APL [8, 39], Dribble [64], Goal [40]. Finally, Situation Calculus [52] has been used to provide not only a logical theory of rational agents, but also as the basis for programming languages for single [50] and multi [62] agent systems.

8.2. Future work

Throughout this paper we have described future work for specific areas. Two further items that we also intend to investigate concern *knowledge-based programs* and *coordination languages*. In particular, as well as refining our work on using temporal logics of knowledge and belief in specifying and implementing agents, we aim to develop the relationship between our approach and that captured by the *knowledge-based programs* of [15]. In addition, the multi-agent approach described in Section 4 can itself be used as a high-level coordination mechanism for agents implemented in other programming languages [48]. This obviates the need for executable version of the specifications, but still requires that verification of the programs used with respect to the required specifications is carried out. Part of our current work is developing a practical agent-based programming language based on this work.

In general, the basis of our current and future work is to continue extending this framework in order to see how many aspects of complex agent scenarios we can capture, and to evaluate these techniques in larger scale, practical applications.

Acknowledgments

I am indebted to all those who have been involved in this work over the last few years, including Clare Dixon, Chiara Ghidini, Benjamin Hirsch, Tony Kakoudakis, Adam Kellett and Mike Wooldridge. I also wish to acknowledge the support of EPSRC in previous years.

References

1. H. Barringer, M. Fisher, D. Gabbay, G. Gough, and R. Owens, "METATEM: An introduction," *Formal Aspects Comput*., vol. 7, no. 5, pp. 533–549, 1995.
2. H. Barringer, M. Fisher, D. Gabbay, R. Owens, and M. Reynolds (eds.), *The Imperative Future: Principles of Executable Temporal Logics*, Research Studies Press: Chichester, United Kingdom, 1996.
3. H. Barringer, R. Kuiper, and A. Pnueli, "A really abstract concurrent model and its temporal logic," in *Proceedings of the Thirteenth ACM Symposium on the Principles of Programming Languages*, St. Petersberg Beach, Florida, 1986.
4. P. Blackburn and M. de Rijke, "Why combine logics?," *Studia Logica*, vol. 59, pp. 5–27, 1997.
5. A. Bolotov and M. Fisher, "A clausal resolution method for CTL branching-time temporal logic," *J. Exp. Theor. Artif. Intell*., vol. 11, pp. 77–93, 1999.
6. J. Bradshaw, M. Greaves, H. Holmback, T. Karygiannis, B. Silverman, N. Suri, and A. Wong, "Agents for the masses?," *IEEE Intell. Syst*., vol. 14, no. 2, pp. 53–63, 1999.
7. M. E. Bratman, "What is intention?," in P. R. Cohen, J. L. Morgan, and M. E. Pollack (eds.), *Intentions in Communication*, The MIT Press: Cambridge, MA, 1990, pp. 15–32.

8. M. Dastani, F. de Boer, F. Dignum, and J.J. Meyer, "Programming agent deliberation: An approach illustrated using the 3APL language," in *Proceedings of the Second International Conference on Autonomous Agents and Multiagent Systems* (AAMAS '03), Melbourne, July 2003, ACM Press 2003.
9. A. Degtyarev, M. Fisher, and B. Konev, "Monodic temporal resolution," in *Proceedings of the CADE-19*. Available as Technical report ULCS-03-001 from http://www.csc.liv.ac.uk/research/, 2003.
10. C. Dixon, "Temporal resolution using a Breadth-First Search Algorithm," *Ann. Math. Artif. Intell.*, vol. 22, pp. 87–115, 1998.
11. C. Dixon and M. Fisher,"Clausal resolution for logics of time and knowledge with synchrony and perfect recall," in *Proceedings of Joint International Conference on Temporal Logic and Advances in Modal Logic (AiML-ICTL)*, Leipzig, Germany, 2000.
12. C. Dixon, M. Fisher, and A. Bolotov, "Resolution in a logic of rational agency," *Artif. Intell.* (Elsevier Science) vol. 139, no. 1, pp. 47–89, 2002.
13. C. Dixon, M. Fisher, and M. Wooldridge, "Resolution for temporal logics of knowledge," *J. Logic Comput.*, vol. 8, no. 3, pp. 345–372, 1998.
14. E. A. Emerson, "Temporal and modal logic," in J. van Leeuwen (ed.), *Handbook of Theoretical Computer Science*, Elsevier, 1990, pp. 996–1072.
15. R. Fagin, J. Halpern, Y. Moses, and M. Vardi, *Reasoning About Knowledge*, MIT Press, 1996.
16. M. Fisher, "A resolution method for temporal logic," in *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence (IJCAI)*, Sydney, Australia, 1991.
17. M. Fisher, "A survey of concurrent METATEM – the language and its applications," in *First International Conference on Temporal Logic (ICTL)*, Bonn, Germany. (Published in *Lecture Notes in Computer Science*, vol. 827, Springer-Verlag), 1994.
18. M. Fisher, "Representing and executing agent-based systems," in M. Wooldridge and N. R. Jennings (eds.), *Intelligent Agents*, 1995.
19. M. Fisher, "A temporal semantics for concurrent METATEM," *J. Symb. Comput.*, vol. 22, no. 5/6, pp. 627–648, 1996a.
20. M. Fisher, "An introduction to executable temporal logics," *Knowl. Eng. Rev.*, vol. 11, no. 1, pp. 43–56, 1996b.
21. M. Fisher, "A normal form for temporal logic and its application in theorem-proving and execution," *J. Logic Comput.*, vol. 7, no. 4, 1997a.
22. M. Fisher, "An open approach to concurrent theorem-proving," in J. Geller, H. Kitano, and C. Suttner (eds.), *Parallel Processing for Artificial Intelligence*, Elsevier/North-Holland, vol. 3, 1997b.
23. M. Fisher, "Implementing BDI-like systems by direct execution," in *Proceedings of International Joint Conference on Artificial Intelligence (IJCA1)*, 1997c.
24. M. Fisher, "Towards the refinement of executable temporal objects," in H. Bowman and J. Derrick (eds.), *Formal Methods for Open Object-Based Distributed Systems*, 1997d.
25. M. Fisher, "Representing abstract agent architectures," in J. P. Müller, M. P. Singh, and A. S. Rao (eds.), *Intelligent Agents V – Proceedings of the Fifth International Workshop on Agent Theories, Architectures, and Languages (ATAL-98)*, *Lecture Notes in Artificial Intelligence*, Springer-Verlag: Heidelberg, 1996.
26. M. Fisher, C. Dixon, and M. Peim, "Clausal temporal resolution," *ACM Trans. Comput. Logic*, vol. 2, no. 1, pp. 12–56, 2001.
27. M. Fisher and C. Ghidini, "Programming resource-bounded deliberative agents," in *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, 1999.
28. M. Fisher and C. Ghidini, "Agents playing with dynamic resource bounds," in *ECAI Workshop on Balancing Reactivity and Social Deliberation in Multi-Agent Systems*, Berlin, Germany, 2000a.
29. M. Fisher and C. Ghidini, "Specifying and implementing agents with dynamic resource bounds," in *Proceedings of ECAI-2000 Workshop on Cognitive Robotics*, 2000b.
30. M. Fisher and T. Kakoudakis, "Flexible agent grouping in executable temporal logic," in *Proceedings of Twelfth International Symposium on Languages for Intensional Programming (ISLIP)*, 1999.
31. B. Hirsch, M. Fisher, and C. Ghidini, "Organising logic-based agents," in *Proceedings of the Second NASA/IEEE Goddard Workshop on Formal Approaches to Agent-Based Systems (FAABS-II)*, Greenbelt, Maryland, October 2002.

32. M. Fisher and A. Kellett, "Programming dynamic multi-agent systems," in *Proceedings of the UK Intelligent Agents Workshop*, Oxford, 1997.
33. M. Fisher and R. Owens, "From the past to the future: Executing temporal logic programs," in *Proceedings of Logic Programming and Automated Reasoning (LPAR)*, St. Petersberg, Russia. (Published in *Lecture Notes in Computer Science*, Springer-Verlag), vol. 624, 1992.
34. M. Fisher and M. Wooldridge, "On the formal specification and verification of multi-agent systems," *Int. J. Coop. Inform. Syst.*, vol. 6, no. 1, pp. 37–65, 1997.
35. M. Fisher, M. Wooldridge, and C. Dixon, "A resolution-based proof method for temporal logics of knowledge and belief," in *Proceedings of the International Conference on Formal and Applied Practical Reasoning (FAPR)*, Bonn, Germany, 1996.
36. S. Franklin and A. Graesser, "Is it an agent, or just a program? A taxonomy for autonomous agents," in J. P. Müller, M. J. Wooldridge, and N. R. Jennings (eds.), *Intelligent Agents III – Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages (ATAL-96)*, *Lecture Notes in Artificial Intelligence*, Springer-Verlag: Heidelberg, 1996.
37. J. Y. Halpern and M. Y. Vardi, "Model checking vs. theorem proving: A manifesto," in V. Lifschitz (ed.), *AI and Mathematical Theory of Computation – Papers in Honor of John McCarthy*, Academic Press, 1991.
38. D. Harel, D. Kozen, and J. Tiuryn, *Dynamic Logic*, MIT Press: Cambridge, MA, USA, 2000.
39. K. Hindriks, F. de Boer, W. van der Hoek, and J.-J. Meyer, "Formal semantics for an abstract agent programming language," in *Intelligent Agents IV: Proceedings of the Fourth International Workshop on Agent Theories, Architectures and Languages*, Vol. 1365 of *Lecture Notes in Artificial Intelligence*, 1998, pp. 215–229.
40. K. Hindriks, F. de Boer, W. van der Hoek, and J.-J. Meyer, "Agent programming with declarative goals," in C. Castelfranchi and Y. Lespérance (eds.), *Intelligent Agents VII, Proceedings of the 6th Workshop on Agent Theories, Architectures, and Languages (ATAL)*, Vol. 1986 of *Lecture Notes in Artificial Intelligence*, 2001, pp. 228–243.
41. G. Holzmann, "The model checker spin," *IEEE Trans. Software Eng.*, vol. 23, no. 5, pp. 279–295, 1997. Special issue on Formal Methods in Software Practice.
42. U. Hustadt, C. Dixon, R. Schmidt, J.-J. M. M. Fisher, and W. van der Hoek, "Verification within the KARO agent theory," in *Proceedings of the First Goddard Workshop on Formal Approaches to Agent-Based Systems*, Goddard Space Flight Center, Greenbelt, Maryland, USA. *Lecture Notes in Computer Science*, Springer-Verlag, 2000a.
43. U. Hustadt, C. Dixon, R. A. Schmidt, and M. Fisher, "Normal forms and proofs in combined modal and temporal logics," in *Proceedings of the Third International Workshop on Frontiers of Combining Systems (FroCoS'2000)*, Vol. 1794 of *Lecture Notes in Artificial Intelligence*, 2000b.
44. U. Hustadt, C. Dixon, R. A. Schmidt, M. Fisher, J.-J. Meyer, and W. van der Hoek, "Reasoning about agents in the KARO framework," in C. Bettini and A. Montanari (eds.), *Proceedings of the Eighth International Symposium on Temporal Representation and Reasoning (TIME-01)*, Cividale del Friuli, Italy, pp. 206–213, 2001.
45. N. R. Jennings and M. Wooldridge, "Applications of agent technology," in *Agent Technology: Foundations, Applications, and Markets*, Springer-Verlag: Heidelberg, 1998.
46. A. Kellett, "Implementation techniques for concurrent METATEM," Ph.D. thesis, Department of Computing and Mathematics, Manchester Metropolitan University, 2000.
47. A. Kellett and M. Fisher, "Automata representations for concurrent METATEM," in *Proceedings of the Fourth International Workshop on Temporal Representation and Reasoning (TIME)*, 1997a.
48. A. Kellett and M. Fisher, "Concurrent METATEM as a coordination language," in *Coordination Languages and Models (LNCS 1282)*, 1997b.
49. Y. Kesten, Z. Manna, and A. Pnueli, "Temporal verification of simulation and refinement," *Lect. Notes Comput. Sci.*, vol. 803, pp. 273–346, 1994.
50. H. Levesque, R. Reiter, Y. Lespérance, F. Lin, and R. Scherl, "GOLOG: A logic programming language for dynamic domains," *J. Logic Prog.*, vol. 31, pp. 59–84, 1997.
51. Z. Manna and A. Pnueli, *The Temporal Logic of Reactive and Concurrent Systems: Specification*, Springer-Verlag: New York, 1992.

52. J. McCarthy and P. J. Hayes, "Some philosophical problems from the standpoint of artificial intelligence," in B. Meltzer and D. Michie (eds.), *Machine Intelligence 4*, Edinburgh University Press, 1969.
53. A. Mili, J. Desharnais, and J. R. Gagné, "Formal models of stepwise refinements of programs," *ACM Comput. Surv.*, vol. 18, no. 3, pp. 231–276, 1986.
54. N. Muscettola, P. P. Nayak, B. Pell, and B. Williams, "Remote agent: To boldly go where no AI system has gone before," *Artif. Intell.*, vol. 103, nos. 1–2, pp. 5–48, 1998.
55. H. J. Ohlbach, "Translation methods for non-classical logics – an overview," *J. IGPL*, vol. 1, no. 1, pp. 69–90, 1993.
56. A. Pnueli, "The temporal semantics of concurrent programs," *Theor. Comput. Sci.*, vol. 13, pp. 45–60, 1981.
57. A. Pnueli and R. Rosner, "On the synthesis of a reactive module," in *Proceedings of the 16th ACM Symposium on the Principles of Programming Languages*, 1989, pp. 179–190.
58. A. Rao, "AgentSpeak(L): BD1 agents speak out in a logical computable language," in W. Van de Velde and J. Perram (eds.), *Agents Breaking Away – Proceedings of the Seventh European Workshop on Modelling Autonomous Agents in a Multi-Agent World, MAAMAW-96 (LNA1 1038)*, 1996a, pp. 42–55.
59. A. S. Rao, "Decision procedures for prepositional linear-time belief-desire-intention logics," in M. Wooldridge, J. P. Müller, and M. Tambe (eds.), *Intelligent Agents II (LNAI 1037)*, Springer-Verlag: Heidelberg, Germany, 1996b, pp. 33–48.
60. A. S. Rao and M. Georgeff, "BDI agents: From theory to practice," in *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95)*, San Francisco, CA, 1995, pp. 312–319.
61. A. S. Rao and M. P. Georgeff, "Modeling agents within a BDI-architecture," in R. Fikes and E. Sandewall (eds.), *International Conference on Principles of Knowledge Representation and Reasoning (KR)*, Cambridge, Massachusetts, 1991.
62. S. Shapiro, Y. Lespérance, and H. Levesque, "Agents for expertise location," in W. Wobeke, M. Pagnucco, and C. Zhang (eds.), *Agents and Multi-Agent Systems – Formalisms, Methodologies, and Applications*, Springer-Verlag: Heidelberg, Germany, 1998.
63. B. van Linder, W. van der Hoek, and J. J. C. Meyer, "How to motivate your agents," in M. Wooldridge, J. P. Müller, and M. Tambe (eds.), *Intelligent Agents 11 (LNAI 1037)*, Springer-Verlag: Heidelberg, Germany, 1996, pp. 17–32.
64. B. van Riemsdijk, W. van der Hoek, and J.-J. Meyer, "Agent programming in dribble: From beliefs to goals with plans," in J. Rosenschein, T. Sandholm, M. Wooldridge, and M. Yokoo (eds.), *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2002, pp. 393–400.
65. M. Y. Vardi, "A temporal fixpoint calculus," in *Proceedings of the Fifteenth ACM Symposium on the Principles of Programming Languages*, San Diego, California (Extended Abstract), 1988, pp. 250–259.
66. M. Wooldridge, C. Dixon, and M. Fisher, "A tableau-based proof method for temporal logics of knowledge and belief," *J. Appl. Non-Classical Logics*, vol. 8, no. 3, pp. 225–258, 1998.
67. M. Wooldridge and N. R. Jennings, "Intelligent agents: Theory and practice," *Knowl. Eng. Rev.*, vol. 10, no. 2, pp. 115–152, 1995.
68. M. Wooldridge and A. Rao (eds.), *Foundations of Rational Agency*, *Applied Logic Series*, Kluwer Academic Publishers, 1999.
69. M. J. Wooldridge, "The logical modelling of computational multi-agent systems," Ph.D. thesis, Department of Computation, UMIST, Manchester, UK, 1992.
70. M. J. Wooldridge, "A knowledge-theoretic semantics for concurrent MetateM," in J. P. Müller, M. J. Wooldridge, and N. R. Jennings (eds.), *Intelligent Agents III – Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages (ATAL-96)*, *Lecture Notes in Artificial Intelligence*, Springer-Verlag: Heidelberg, 1996.