Specifying and Verifying Distributed Intelligent Systems

Michael Fisher and Michael Wooldridge

Department of Computing
Manchester Metropolitan University
Chester Street
Manchester M1 5GD
United Kingdom

Abstract. This paper describes first steps towards the formal specification and verification of Distributed Artificial Intelligence (DAI) systems, through the use of temporal belief logics. The paper first describes Concurrent METATEM, a programming language for DAI, and then develops a logic that may be used to reason about Concurrent METATEM systems. The utility of this logic for specifying and verifying Concurrent METATEM systems is demonstrated through a number of examples. The paper concludes with a brief discussion of the wider implications of the work, and in particular on the use of similar logics for reasoning about DAI systems in general.

1 Introduction

In the past decade, the discipline of DAI has moved from being a somewhat obscure relation of mainstream AI to being a major research area in its own right. DAI techniques have been applied to domains as diverse as archaeology and economics, as well as more mundane problems such as distributed sensing and manufacturing control [7]. Many testbeds for building and experimenting with DAI systems have been reported [2, 19, 21, 10, 26, 9, 24, 12]. And yet almost no research has considered the important problems of *specifying* and *verifying* DAI systems. In short, the purpose of this paper is to address these issues: we present preliminary results on specifying and verifying systems implemented using Concurrent METATEM, a novel new programming language for DAI [14, 13, 16]. A Concurrent METATEM system contains a number of concurrently executing *objects*, which are able to communicate via message passing; each object executes a temporal logic specification representing its desired behaviour. In this paper, we describe Concurrent METATEM in more detail, and then develop a *temporal belief logic* that can be used to reason about Concurrent METATEM systems. We show how this logic may be used to specify and verify the properties of Concurrent METATEM systems. Additionally, the paper discusses the wider issues involved in reasoning about DAI systems.

The remainder of the paper is structured as follows. In the following section, we outline the background to, and motivation for, our work. In §2, we describe Concurrent METATEM in more detail. In §3, we develop a Temporal Belief Logic (TBL), which is used, in §4, to axiomatize the properties of Concurrent METATEM systems. Examples of the use of the logic for specifying and verifying Concurrent METATEM systems are presented in §5. Some comments and conclusions are presented in §6; in particular, we discuss the implications of our work for reasoning about DAI systems in general.

1.1 Background and Motivation

Distributed AI is a subfield of AI whose loose focus is the study of cooperative activity in systems composed of multiple intelligent computational objects (or *agents*, as they are often called). Over the past decade, many frameworks for building and experimenting with DAI systems have been reported. Probably the best-known is MACE, a LISP-based fully instrumented testbed for building DAI systems at a range of sizes and complexity [19]. Other testbeds include Georgeff and Lanksy's procedural reasoning system [21], [20] (aspects of which have been formalised [25]), and the MCS/IPEM platform described by Doran *at al.*, (in which each agent has virtual access to a sophisticated non-linear planner) [10]. More recently, Shoham has described AGENT0, an interpreted programming language for DAI which represents a first step towards the ideal of an 'agent-oriented programming' paradigm [26]. Several actor-style languages for DAI have been developed [2]: Bouron *et al.* describe MAGES, a system based on the ACTALK actor language [9]; Ferber and Carle describe MERING IV, a reflexive concurrent object language [12]; Maruichi *et al.* describe an 'autonomous agent' model (similar in some respects to the computational model underlying Concurrent METATEM), in which autonomous, continually executing actor-like objects communicate through asynchronous message passing [24].

One aspect of DAI missing from all the above accounts is the notion of objects/agents as *reactive systems*, in the following sense: a reactive system is one which cannot adequately be described in terms of 'initial' and 'final' states. Any non-terminating system is therefore a reactive system. The purpose of a reactive system is to *maintain an interaction* with an environment [22]. Any concurrent system is a reactive system, since the objects/agents in a concurrent system must maintain an interaction with other objects/agents: DAI systems are therefore reactive. (The term 'reactive system' has recently been used in AI to describe systems that respond rapidly to the world, without reasoning explicitly about it: we do *not* use the term in this sense; we use it only in the sense we have just described.)

How is one to reason about reactive systems? *Temporal logic* has long been considered a viable tool for this purpose (see, for example, [11] for a good introduction to the extensive literature in this area). This is because temporal logic allows one to describe the ongoing behaviour of a system, which cannot be easily expressed in other formalisms (such as those based on pre- and post-conditions). We take it as axiomatic that the notion of a reactive system is a valuable one for DAI, and that temporal logic is appropriate for reasoning about DAI systems (these issues are examined in more detail in [16]).

Concurrent METATEM is a programming language for DAI in which the notion of reactivity is central. A Concurrent METATEM system contains a number of concurrently executing objects (a.k.a. agents), which are able to communicate through message passing. Each object executes a temporal logic specification of its desired behaviour. The move from specification to implementation in Concurrent METATEM is therefore a small one (since the specification and implementation languages have much in common). However, 'standard' temporal logic cannot simply be used to describe Concurrent MetateM systems (although see [13] for a semantics of Concurrent MetateM based on *dense* temporal logic [5]). This is because each object in a Concurrent MetateM system is a symbolic AI system in its own right: it contains a set of explicitly represented (temporal logic) formulae which it manipulates in order to decide what to do. Thus, to reason about the behaviour of Concurrent MetateM systems, we require some description of the formulae that each agent is manipulating at each moment in time. One way of doing this would be to use a first-order temporal meta-language (as in [4]). However, meta-languages are notationally cumbersome, and can be confusing. What we propose instead is to use a multi-modal language, which contain both temporal connectives and an indexed set of modal *belief* operators, one for each object. These belief operators will be used to describe the formulae that each agent manipulates (see [23] for a detailed exposition on the use of belief logics without a temporal component for describing AI systems).

We have now set the scene for the remainder of the paper. In the next section, we describe Concurrent MetateM in more detail. We then develop a temporal belief logic, and show how it can be used to reason about Concurrent MetateM systems.

2 Concurrent MetateM

In this section we introduce Concurrent MetateM, and present a short example to illustrate its use; in §2.3 we describe the temporal logic that is used for Concurrent MetateM program rules. Although first-order temporal logic is used to represent an object's behaviour in MetateM [15], we will, for the sake of simplicity, restrict our examples to the propositional case.

2.1 Objects and Object Execution

A Concurrent MetateM system contains a number of concurrently executing *objects*, which are able to communicate through asynchronous broadcast message passing. Each object directly executes a temporal logic specification, given to it as a set of 'rules'. In this section, we describe objects and their execution in more detail. Each object has two main components:

- an *interface*, which defines how the object may interact with its environment (i.e., other objects);
- a *computational engine*, which defines how the object may act.

An object interface consists of three components:

- a unique *object identifier* (or just object id), which names the object;
- a set of symbols defining what messages will be accepted by the object — these are called *environment propositions*;
- a set of symbols defining messages that the object may send — these are called *component propositions*.

For example, the interface definition of a 'stack' object might be [13]:

$$stack(pop, push)[popped, stackfull]$$

Here, *stack* is the object id that names the object, $\{pop, push\}$ are the environment propositions, and $\{popped, stackfull\}$ are the component propositions. Whenever a message headed by the symbol *pop* is broadcast, the *stack* object will *accept* the message; we describe what this means below. If a message is broadcast which is not declared in the *stack* object's interface, then *stack* ignores it. Similarly, the only messages which can be sent by the *stack* object are headed by the symbols *popped* and *stackfull*. (All other propositions used within the object are *internal* propositions, which have no external correspondence.)

The computational engine of an object is based on the METATEM paradigm of executable temporal logics. The idea which informs this approach is that of directly executing a declarative object specification, where this specification is given as a set of *program rules*, which are temporal logic formulae of the form:

$$\text{antecedent about past} \Rightarrow \text{consequent about future.}$$

The past-time antecedent is a temporal logic formula referring strictly to the past, whereas the future time consequent is a temporal logic formula referring either to the present or future. The intuitive interpretation of such a rule is 'on the basis of the past, do the future', which gives rise to the name of the paradigm: *declarative past and imperative future* [17]. The actual execution of an object is, superficially at least, very simple to understand. Each object obeys a cycle of trying to match the past time antecedents of its rules against a *history*, and executing the consequents of those rules that 'fire'[1].

To make the above discussion more concrete, we will now informally introduce a propositional temporal logic, called Propositional METATEM Logic (PML), in which the individual METATEM rules will be given. (A complete definition of PML is given in §2.3.)

PML is essentially classical propositional logic augmented by a set of modal connectives for referring to the *temporal ordering* of actions. PML is based on a model of time that is *linear* (i.e., each moment in time has a unique successor), *bounded in the past* (i.e., there was a moment that was the 'beginning of time'), and *infinite in the future* (i.e., there are an infinite number of moments in the future). The temporal connectives of PML can be divided into two categories, as follows.

1. Strict past time connectives: '●' (weak last), '◎' (strong last), '◈' (was), '■' (heretofore), '$\mathcal{S}$' (since) and '$\mathcal{Z}$' (zince, or weak since).
2. Present and future time connectives: '$\bigcirc$' (next), '$\Diamond$' (sometime), '$\Box$' (always), '$\mathcal{U}$' (until) and '$\mathcal{W}$' (unless).

The connectives {◎, ●, ◈, ■, $\bigcirc$, $\Diamond$, $\Box$} are unary; the rest are binary. In addition to these temporal connectives, PML contains the usual classical logic connectives.

The meaning of the temporal connectives is quite straightforward, with formulae being interpreted at a particular moment in time. Let ϕ and ψ be formulae of PML. Then $\bigcirc\phi$ is true (satisfied), at the current moment in time if ϕ is true at the next moment in time; $\Diamond\phi$ is true now if ϕ is true now or at some future moment in time;

[1] There are obvious similarities between the execution cycle of an object and production systems — but there are also significant differences. The reader is cautioned against taking the analogy too seriously.

$\square\phi$ is true now if ϕ is true now and at all future moments; $\phi\,\mathcal{U}\,\psi$ is true now if ψ is true at some future moment, and ϕ is true until then — $\mathcal{W}$ is a binary connective similar to $\mathcal{U}$, allowing for the possibility that the second argument never becomes true.

The past-time connectives are similar: $\circledcirc$ and $\bullet$ are true now if their arguments were true at the previous moment in time — the difference between them is that, since the model of time underlying the logic is bounded in the past, the beginning of time is a special case: $\circledcirc\phi$ will always be false when interpreted at the beginning of time, whereas $\bullet\phi$ will always be true at the beginning of time; $\blacklozenge\,\phi$ will be true now if ϕ was true at some previous moment in time; $\blacksquare\phi$ will be true now if ϕ was true at all previous moments in time; $\phi\,\mathcal{S}\,\psi$ will be true now if ψ was true at some previous moment in time, and ϕ has been true since then; $\mathcal{Z}$ is the same, but allowing for the possibility that the second argument was never true. Finally, a temporal operator that takes no arguments can be defined which is true only at the first moment in time: this useful operator is called '**start**'.

Any formula of PML which refers strictly to the past is called a *history formula*; any formula referring to the present or future is called a *commitment formula*; any formula of the form

$$\textit{history formula} \Rightarrow \textit{commitment formula}$$

is called a *rule*; an object specification is a set of such rules.

A more precise definition of object execution will now be given. Objects continually execute the following cycle:

1. Update the *history* of the object by receiving messages from other objects and adding them to their history. (This process is described in more detail below.)
2. Check which rules *fire*, by comparing past-time antecedents of each rule against the current history to see which are satisfied.
3. *Jointly execute* the fired rules together with any commitments carried over from previous cycles. This is done by first collecting consequents of newly fired rules and old commitments, which become *commitments*. It may not be possible to satisfy *all* the commitments on the current cycle, in which case unsatisfied commitments are carried over to the next cycle. An object will then have to choose between a number of execution possibilities.
4. Goto (1).

Clearly, step (3) is the heart of the execution process. Making a bad choice at this step may mean that the object specification cannot subsequently be satisfied (see [3, 15]).

A natural question to ask is: how do objects do things? How do they send messages and perform actions? When a proposition in an object becomes *true*, it is compared against that object's interface (see above); if it is one of the object's *component* propositions, then that proposition is broadcast as a message to all other objects. On receipt of a message, each object attempts to match the proposition against the environment propositions in their interface. If there is a match then they add the proposition to their history, prefixed by a '$\circledcirc$' operator, indicating that the message has just been received.

The reader should note that although the use of only broadcast message-passing may seem restrictive, standard point-to-point message-passing can easily be simulated by adding an extra 'destination' argument to each message; the use of broadcast message-passing as the communication mechanism gives the language the ability to define more adaptable and flexible systems. We will not develop this argument further; the interested reader is urged to either consult our earlier work on Concurrent MetateM [13, 16], or relevant work showing the utility of broadcast and multicast mechanisms [8, 6, 24].

2.2 A Simple Concurrent MetateM System

To illustrate Concurrent MetateM in more detail, we present in Fig. 1 an example system (outlined originally in [3] and extended in [13])[2]. The system contains three objects: rp, $rc1$ and $rc2$. The object rp is a 'resource producer': it can '*give*' to only one object at a time (rule 3), and will commit to eventually *give* to any object that '*ask*'s' (rules 1 and 2). Object rp will only accept messages $ask1$ and $ask2$, and can only send $give1$ and $give2$ messages.

$rp(ask1, ask2)[give1, give2]$:
1. $\odot ask1 \Rightarrow \Diamond give1$;
2. $\odot ask2 \Rightarrow \Diamond give2$;
3. $\mathbf{start} \Rightarrow \Box\neg(give1 \wedge give2)$.

$rc1(give1)[ask1]$:
1. $\mathbf{start} \Rightarrow ask1$;
2. $\odot ask1 \Rightarrow ask1$.

$rc2(ask1, give2)[ask2]$:
1. $\odot(ask1 \wedge \neg ask2) \Rightarrow ask2$.

Fig. 1. A Simple Concurrent MetateM System

The object $rc1$ will send an $ask1$ message on every cycle: this is because **start** is satisfied at the beginning of time, thus firing rule 1, while $\odot ask1$ will then be true on the next cycle, thus firing rule 2, and so on. Thus $rc1$ asks for the message on every cycle, using an $ask1$ message. Object $rc1$ will only accept a $give1$ message, and can only send an $ask1$ message.

The object $rc2$ will send an $ask2$ message on every cycle where, on its previous cycle, it did not send an $ask2$ message, but $rc1$ sent an $ask1$ message. Object $rc2$ will only accept messages $ask1$ and $give2$, and can only send an $ask2$ message.

[2] Note that in the interests of readability, this program has been 'pretty printed' using the symbolic form of temporal connectives, rather than the plain-text form that the implementation actually requires; additionally, rule numbers have been introduced to facilitate easy reference — these are *not* part of the language!

To conclude, the system in Fig. 1 has the following properties:

1. Objects $rc1$ and $rc2$ will ask rp for the resource infinitely often;
2. Every time rp is 'asked', it must eventually 'give' to the corresponding asker.

¿From which we can informally deduce that:

3. Object rp will give the resource to both objects infinitely often.

In §5, we will show formally that the system does indeed have this behaviour.

2.3 A Propositional Temporal Logic (PML)

We now give a complete definition of the propositional temporal logic used for program rules (PML); for a fuller account of propositional temporal logic see, for example, [11].

Syntax

Definition 1 *The language of* PML *contains the following symbols:*

1. *a countable set,* Prop, *of* proposition symbols*;*
2. *the symbol* **true***;*
3. *the* unary propositional connective '$\neg$' *and* binary propositional connective '$\vee$'*;*
4. *the* unary temporal connectives $\{\circledcirc, \bigcirc\}$*;*
5. *the* binary temporal connectives $\{\mathcal{U}, \mathcal{S}\}$*;*
6. *the punctuation symbols* $\{), (\}$*.*

All the remaining propositional and temporal connectives are introduced as abbreviations (see below). The syntax of PML is defined as follows.

Definition 2 *Well-formed formulae of* PML*,* WFF(PML)*, are defined by the following rules.*

1. *All proposition symbols are* PML *formulae;*
2. *If ϕ is a* PML *formula then so are $\neg\phi$, $\bigcirc\phi$ and $\circledcirc\phi$;*
3. *If ϕ and ψ are* PML *formulae then so are $\phi \vee \psi$, $\phi\,\mathcal{U}\,\psi$, and $\phi\,\mathcal{S}\,\psi$;*
4. *If ϕ is a* PML *formula, then (ϕ) is a* PML *formula.*

Semantics The semantics of well-formed formulae of PML is given in the obvious way, with formulae being interpreted in a model, at a particular moment in time.

Definition 3 *A* model*, M, for* PML *is a structure $\langle \sigma, \pi_p \rangle$ where*

- *σ is the ordered set of states s_0, s_1, s_2, ... representing 'moments' in time, and*
- *$\pi_p : \mathbb{N} \times \mathrm{Prop} \to \{T, F\}$ is a function assigning T or F to each atomic proposition at each moment in time.*

$$
\begin{array}{lll}
\langle M, u\rangle \models \mathbf{true} & & \\
\langle M, u\rangle \models p & \text{iff } \pi_p(u, p) = T & \\
\langle M, u\rangle \models \neg\phi & \text{iff } \langle M, u\rangle \not\models \phi & \\
\langle M, u\rangle \models \phi \vee \psi & \text{iff } \langle M, u\rangle \models \phi \ \text{ or } \ \langle M, u\rangle \models \psi & \\
\langle M, u\rangle \models \bigcirc\phi & \text{iff } \langle M, u+1\rangle \models \phi & \\
\langle M, u\rangle \models \circledcirc\phi & \text{iff } u > 0 \ \text{ and } \ \langle M, u-1\rangle \models \phi & \\
\langle M, u\rangle \models \phi\,\mathcal{U}\,\psi & \text{iff } \exists v \in \mathbb{N} \cdot (u \leq v) \ \text{ and } \ \langle M, v\rangle \models \psi & \\
 & \qquad\qquad \text{and } \ \forall w \in \{u, \ldots, v-1\} \cdot \langle M, w\rangle \models \phi & \\
\langle M, u\rangle \models \phi\,\mathcal{S}\,\psi & \text{iff } \exists v \in \mathbb{N} \cdot (v < u) \ \text{ and } \ \langle M, v\rangle \models \psi & \\
 & \qquad\qquad \text{and } \ \forall w \in \{v+1, \ldots, u-1\} \cdot \langle M, w\rangle \models \phi &
\end{array}
$$

Fig. 2. Semantics of PML

As usual, we use the relation '$\models$' to give the truth value of a formula in a model M, at a particular moment in time u. This relation is defined for formulae of PML in Fig. 2. Note that these rules only define the semantics of the basic propositional and temporal connectives; the remainder are introduced as abbreviations (we omit the propositional connectives, as these are standard):

$$
\begin{array}{rlrl}
\bullet\phi & \stackrel{\text{def}}{=} \neg \circledcirc \neg\phi & \phi\,\mathcal{W}\,\psi & \stackrel{\text{def}}{=} \Box\phi \vee \phi\,\mathcal{U}\,\psi \\
\mathbf{start} & \stackrel{\text{def}}{=} \bullet\mathbf{false} & \blacklozenge\phi & \stackrel{\text{def}}{=} \mathbf{true}\,\mathcal{S}\,\phi \\
\Diamond\phi & \stackrel{\text{def}}{=} \mathbf{true}\,\mathcal{U}\,\phi & \blacksquare\phi & \stackrel{\text{def}}{=} \neg\blacklozenge\neg\phi \\
\Box\phi & \stackrel{\text{def}}{=} \neg\Diamond\neg\phi & \phi\,\mathcal{Z}\,\psi & \stackrel{\text{def}}{=} \blacksquare\phi \vee \phi\,\mathcal{S}\,\psi
\end{array}
$$

Proof Theory The proof theory of PML has been examined exhaustively elsewhere (see, for example, [18, 27, 1]). Here, we identify some axioms and inference rules that are later used in our proofs.

$$\vdash \Box(\phi \Rightarrow \psi) \Rightarrow (\Box\phi \Rightarrow \Box\psi) \tag{1}$$
$$\vdash \Box(\phi \Rightarrow \psi) \Rightarrow (\Diamond\phi \Rightarrow \Diamond\psi) \tag{2}$$
$$\vdash \bigcirc(\phi \Rightarrow \psi) \Rightarrow (\bigcirc\phi \Rightarrow \bigcirc\psi) \tag{3}$$
$$\vdash \Diamond\Diamond\phi \Leftrightarrow \Diamond\phi \tag{4}$$
$$\vdash (\mathbf{start} \Rightarrow \Box\phi) \Rightarrow \Box\phi \tag{5}$$
$$\vdash \Box(\phi \Rightarrow \bigcirc\phi) \Rightarrow (\phi \Rightarrow \Box\phi) \tag{6}$$
$$\vdash \Box\phi \ \Rightarrow \ \phi \tag{7}$$
$$\vdash (\bigcirc\circledcirc\phi \ \Rightarrow \ \psi) \ \Leftrightarrow \ (\phi \Rightarrow \psi) \tag{8}$$
$$\vdash (\circledcirc\bigcirc\phi \ \Rightarrow \ \psi) \ \Rightarrow \ (\phi \Rightarrow \psi) \tag{9}$$
$$\text{From } \vdash \phi \text{ infer } \vdash \Box\phi \tag{10}$$

$$\text{From } \vdash \phi \text{ infer } \vdash \Diamond\phi \tag{11}$$
$$\text{From } \vdash \phi \text{ infer } \vdash \bigcirc\phi \tag{12}$$

3 A Temporal Belief Logic

This section introduces a Temporal Belief Logic (TBL), that can be used to reason about Concurrent MetateM systems. Like PML, it is a linear discrete temporal logic, with bounded past and infinite future. TBL is similar to the logic L^{B} that Konolige developed for his deduction model of belief [23]; it was derived from Wooldridge's work on reasoning about multi-agent systems [28, 29].

Syntax Syntactically, TBL can be thought of as PML augmented by an indexed set of unary modal operators $[i]$, (applied to formulae of PML), and closed under the propositional and temporal connectives of PML. The 'i' in $[i]$ is an object id. A TBL formula of the form $[i]\phi$ should be read "ϕ is in i's current state." Thus if ϕ is a history formula, this would say that ϕ was in i's history; if ϕ was a rule, this would say that ϕ was one of i's rules, and if ϕ were a commitment, it would say that i was committed to ϕ.

Definition 4 *The language of* TBL *contains all the symbols of* PML*, and in addition the square brackets* $\{], [\}$ *and a countable set* obj *of object ids.*

Definition 5 *Well-formed formulae of* TBL *are defined by the following rules:*

1. TBL *contains all formulae of* PML*;*
2. *If* ϕ *is a formula of* PML *and* $i \in obj$ *then* $[i]\phi$ *is a formula of* TBL*;*
3. TBL *is closed under the propositional and temporal connectives of* PML*.*

Semantics One obtains a model for TBL by taking a PML model and adding a function mapping each object id and each moment in time to a set of PML formulae representing the object's state at that moment in time.

Definition 6 *A* model $\mathcal{M}$ *for* TBL *is a structure* $\langle \sigma, \pi_p, \mathcal{O} \rangle$ *where*

$$\mathcal{O} : obj \times \mathbb{N} \rightarrow \text{powerset WFF(PML)}$$

and σ *and* π_p *are as in* PML*.*

The formula $[i]\phi$ will be satisfied if ϕ is in i's state at the appropriate time. This gives the following additional semantic rule, for formulae of the form $[i]\phi$.

$$\langle \mathcal{M}, u \rangle \models [i]\phi \quad \text{iff} \quad \phi \in \mathcal{O}(i, u)$$

Other formulae are interpreted using the same rules as before.

3.1 Proof Theory

TBL inherits all the axioms and inference rules of PML. Further, since there is no direct interaction between the temporal connectives and the '$[i]$' operators in the basic TBL system, then the only extra axiom that we add is the following.

$$\vdash \; [i](P \Rightarrow F) \;\Rightarrow\; (([i]P) \Rightarrow ([i]F)) \tag{13}$$

This characterises the fact that the '$\Rightarrow$' operator in Concurrent METATEM follows the same logical rules as standard implication.

4 Axiomatizing Concurrent METATEM

In this section, we use TBL to *axiomatize* the properties of Concurrent METATEM systems. This involves extending the basic TBL proof system outlined above to account for the particular properties of the Concurrent METATEM systems that we intend to verify properties of.

The first axiom we add describes the conditions under which an object will send a message: if a proposition becomes 'true' inside an object, and the proposition symbol appears in the object's component propositions, then the proposition is broadcast. (Note the use of ordinary PML propositions to describe messages.) So if P is one of i's component propositions, then the following axiom holds.

$$\vdash ([i]P) \Rightarrow \Diamond P \tag{14}$$

The second axiom deals with how objects *receive* messages: if a message is broadcast, and the proposition symbol of that message appears in an object's environment proposition list, then that message is accepted by the object, which subsequently 'believes' that the proposition was true. So if P is one of i's environment propositions, then the following axiom holds.

$$\vdash \; P \Rightarrow \Diamond[i] \odot P \tag{15}$$

The third axiom states that objects maintain accurate histories.

$$\vdash \; ([i]\phi) \Rightarrow [i] \bigcirc \odot \phi \tag{16}$$

To simplify the proofs, we will assume that all objects in a Concurrent METATEM system execute synchronously, i.e., the 'execution steps' of each object match. This simplification allows us to add the following *synchronisation axioms*. (Note that, without this simplification, each object would be executing under a distinct local clock, and so proving properties of such a system becomes *much* more difficult, though possible [13].)

$$\vdash (\bigcirc[i]\phi \;\Rightarrow\; \psi) \;\Leftrightarrow\; ([i]\bigcirc\phi \;\Rightarrow\; \psi) \tag{17}$$

$$\vdash (\odot[i]\phi \;\Rightarrow\; \psi) \;\Leftrightarrow\; ([i]\odot\phi \;\Rightarrow\; \psi) \tag{18}$$

Now, for every rule, R, in an object, i, we add the following axiom showing that once the object has started executing, the rule is always applicable.

$$\vdash \; [i]\mathbf{start} \;\Rightarrow\; \Box[i]R \tag{19}$$

Finally, to simplify the proofs still further, we will assume that all objects commence execution at the same moment, denoted by the global '**start**' operator. Thus, for every object, i, we add the following axiom.

$$\vdash \ \mathbf{start} \ \Rightarrow \ [i]\mathbf{start} \tag{20}$$

5 Examples

In this section, we show how TBL can be used to reason about Concurrent MetateM systems. We begin by proving some properties of the system presented earlier. In the proofs that follow, we will use the notation $\{\mathrm{S}\} \vdash \phi$ to represent the statement '*system S satisfies property* ϕ'. Also, as the majority of the proof steps involve applications of the *Modus Ponens* inference rule, we will omit reference to the particular rule used at each step. As we refer to the axioms and theorems used, the rule used will be clear at each step.

5.1 Resource Controller

Let the system given in Fig. 1 be called S1. This system contains three objects: a resource producer (rp), and two resource consumers ($rc1$ and $rc2$). The first property we prove is that the object $rc1$, once it has commenced execution, satisfies the commitment $ask1$ on every cyle.

Lemma 1. $\{\mathrm{S1}\} \vdash \ \mathbf{start} \ \Rightarrow \ \Box[rc1]ask1.$

Proof See Fig. 3.

Using this result, it is not difficult to establish that the message $ask1$ is then sent infinitely often.

Lemma 2. $\{\mathrm{S1}\} \vdash \ \Box\Diamond ask1.$

Proof This, and all remaining proofs, are omitted due to space restrictions.

Similarly, we can show that any object that is *listening* for $ask1$ messages, in particular rp, will receive them infinitely often.

Lemma 3. $\{\mathrm{S1}\} \vdash \ \mathbf{start} \ \Rightarrow \ \Box\Diamond[rp] \circledcirc ask1.$

Now, since we know that $ask1$ is one of rp's environment propositions, then we can show that once both rp and $rc1$ have started, the resource will be given to $rc1$ infinitely often.

Lemma 4. $\{\mathrm{S1}\} \vdash \ \mathbf{start} \ \Rightarrow \ \Box\Diamond give1.$

Similar properties can be shown for $rc2$. Note, however, that we require knowledge about $rc1$'s behaviour in order to reason about $rc2$'s behaviour.

Lemma 5. $\{\mathrm{S1}\} \vdash \ \mathbf{start} \ \Rightarrow \ \Box\Diamond[rp] \circledcirc ask2.$

1. $[rc1]\mathbf{start} \;\Rightarrow\; \Box[rc1](\mathbf{start} \Rightarrow ask1)$	(rule 1 in $rc1$)
2. $\mathbf{start} \;\Rightarrow\; \Box[rc1](\mathbf{start} \Rightarrow ask1)$	(axiom 19, 1)
3. $\Box[rc1](\mathbf{start} \Rightarrow ask1)$	(axiom 5, 2)
4. $[rc1](\mathbf{start} \Rightarrow ask1)$	(axiom 7, 3)
5. $[rc1]\mathbf{start} \;\Rightarrow\; [rc1]ask1$	(axiom 13, 4)
6. $\mathbf{start} \;\Rightarrow\; [rc1]ask1$	(axiom 19, 5)
7. $[rc1]\mathbf{start} \;\Rightarrow\; \Box[rc1](\bullet ask1 \Rightarrow ask1)$	(rule 2 in $rc1$)
8. $\mathbf{start} \;\Rightarrow\; \Box[rc1](\bullet ask1 \Rightarrow ask1)$	(axiom 19, 7)
9. $\Box[rc1](\bullet ask1 \Rightarrow ask1)$	(axiom 5, 8)
10. $[rc1](\bullet ask1 \Rightarrow ask1)$	(axiom 7, 9)
11. $[rc1]\bullet ask1 \;\Rightarrow\; [rc1]ask1$	(axiom 13, 10)
12. $\bigcirc([rc1]\bullet ask1 \;\Rightarrow\; [rc1]ask1)$	(inf. rule 12, 11)
13. $\bigcirc[rc1]\bullet ask1 \;\Rightarrow\; \bigcirc[rc1]ask1$	(axiom 3, 12)
14. $\bigcirc\bullet[rc1]ask1 \;\Rightarrow\; \bigcirc[rc1]ask1$	(axiom 18, 13)
15. $[rc1]ask1 \;\Rightarrow\; \bigcirc[rc1]ask1$	(axiom 8, 14)
16. $\Box([rc1]ask1 \;\Rightarrow\; \bigcirc[rc1]ask1)$	(inf. rule 10, 15)
17. $[rc1]ask1 \;\Rightarrow\; \Box[rc1]ask1$	(axiom 6, 16)
18. $\mathbf{start} \;\Rightarrow\; \Box[rc1]ask1$	(6, 17)

Fig. 3. Proof of Lemma 1

Given this, we can derive the following result.

Lemma 6. $\{S1\} \vdash \mathbf{start} \;\Rightarrow\; \Box\Diamond give2.$

Finally, we can show the desired behaviour of the system; compare this to result (3) that we informally deduced in §2.2.

Theorem 7. $\{S1\} \vdash \mathbf{start} \;\Rightarrow\; (\Box\Diamond give1 \;\wedge\; \Box\Diamond give2).$

5.2 Distributed Problem Solving

We now consider a distributed problem solving system. Here, a single object, called *executive*, broadcasts a problem to a group of problem solvers. Some of these problem solvers can solve the particular problem completely, and some will reply with a solution.

We define such a Concurrent MetateM system in Fig. 4. Here, *solvera* can solve a different problem from the one *executive* poses, while *solverb* can solve the desired problem, but doesn't announce the fact (as $solution1$ is not a component proposition

$executive(solution1)[problem1, solved1]:$
1. $\mathbf{start} \Rightarrow \Diamond problem1;$
2. $\odot solution1 \Rightarrow solved1.$

$solvera(problem2)[solution2]:$
1. $\odot problem2 \Rightarrow solution2.$

$solverb(problem1)[solution2]:$
1. $\odot problem1 \Rightarrow \Diamond solution1.$

$solverc(problem1)[solution1]:$
1. $\odot problem1 \Rightarrow \Diamond solution1.$

Fig. 4. A Distributed Problem Solving System

$solverd(problem1, solution1.2)[solution1]:$
1. $(\odot solution1.2 \ \wedge \ \blacklozenge\ problem1) \Rightarrow \Diamond solution1.$

$solvere(problem1)[solution1.2]:$
1. $\odot problem1 \Rightarrow \Diamond solution1.2.$

Fig. 5. Additional Problem Solving Agents

for *solverb*); *solverc* can solve the problem posed by *executive*, and will *eventually* reply with the solution.

If we call this system S2, then we can prove the following.

Theorem 8. $\{\mathrm{S2}\} \vdash \ \mathbf{start} \ \Rightarrow \ \Diamond solved1.$

We now remove *solverc* and replace it by two objects who together can solve $problem1$, but can not manage this individually. These objects, called *solverd* and *solvere* are defined in Fig. 5.

Thus, when *solverd* receives the problem it cannot do anything until it has heard from *solvere*. When *solvere* receives the problem, it broadcasts the fact that it can some of the problem (i.e., it broadcasts $solution1.2$). When *solverd* sees this, it knows it can solve the other part of the problem and broadcasts the whole solution.

Thus, given these new objects we can prove the following (the system is now called S3).

Theorem 9. $\{\mathrm{S3}\} \vdash \ \mathbf{start} \ \Rightarrow \ \Diamond solved1.$

6 Concluding Remarks

In this paper, we have described Concurrent MetateM, a programming language for DAI, and developed a Temporal Belief Logic for reasoning about Concurrent MetateM systems. In effect, we used the logic to develop a crude semantics for the language; this approach did not leave us with a complete proof system for Concurrent MetateM, (since we made the assumption of synchronous action). However, it has the advantage of simplicity when compared to other methods for defining the semantics of the language (such as those based on dense temporal logic [13], or first-order temporal meta-languages [4]).

More generally, logics similar to that developed herein can be used to reason about a wise class of DAI systems: those in which agents/objects have a classic 'symbolic AI' architecture. Such systems typically employ explicit symbolic representations, which are manipulated in order to plan and execute actions. A belief logic such as that described in this paper seems appropriate for describing these representations (see also [23]). A temporal component to the logic seems to be suitable for describing reactive systems, of which DAI systems are an example. In other work, we have developed a family of temporal belief logics, augmented by modalities for describing the actions and messages of individual agents, and demonstrated how these logics can be used to specify a wide range of cooperative structures [28, 29].

Future work will concentrate on refining the axiomatisation of Concurrent MetateM, and on developing (semi-)mechanical proof procedures for logics such as TBL, based upon either multi-modal tableaux or multi-modal resolution.

Acknowledgements Michael Fisher was partially supported by the SERC under research grant GR/H/18449. Michael Wooldridge was partially supported by an SERC PhD award.

References

1. M. Abadi. *Temporal-Logic Theorem Proving.* PhD thesis, Department of Computer Science, Stanford University, March 1987.
2. G. Agha. *Actors - A Model for Concurrent Computation in Distributed Systems.* MIT Press, 1986.
3. H. Barringer, M. Fisher, D. Gabbay, G. Gough, and R. Owens. MetateM: A Framework for Programming in Temporal Logic. In *Proceedings of REX Workshop on Stepwise Refinement of Distributed Systems: Models, Formalisms, Correctness*, Mook, Netherlands, June 1989. (Published in *Lecture Notes in Computer Science*, volume 430, Springer Verlag).
4. H. Barringer, M. Fisher, D. Gabbay, and A. Hunter. Meta-Reasoning in Executable Temporal Logic. In J. Allen, R. Fikes, and E. Sandewall, editors, *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning (KR)*, Cambridge, Massachusetts, April 1991. Morgan Kaufmann.
5. 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 (POPL)*, St. Petersberg Beach, Florida, January 1986.

6. K. Birman. The Process Group Approach to Reliable Distributed Computing. Techanical Report TR91-1216, Department of Computer Science, Cornell University, USA, July 1991.
7. A. H. Bond and L. Gasser, editors. *Readings in Distributed Artificial Intelligence*. Morgan Kaufmann, 1988.
8. A. Borg, J. Baumbach, and S. Glazer. A Message System Supporting Fault Tolerance. In *Proceedings of the Ninth ACM Symposium on Operating System Principles*, pages 90–99, New Hampshire, October 1983. ACM. (In ACM Operating Systems Review, vol. 17, no. 5).
9. T. Bouron, J. Ferber, and F. Samuel. MAGES: A Multi-Agent Testbed for Heterogeneous Agents. In Y. Demazeau and J. P. Muller, editors, *Decentralized AI 2 – Proceedings of the Second European Workshop on Modelling Autonomous Agents and Multi-Agent Worlds (MAAMAW)*. Elsevier/North Holland, 1991.
10. J. Doran, H. Carvajal, Y. J. Choo, and Y. Li. The MCS Multi Agent Testbed: Developments and Experiments. In S. M. Deen, editor, *Proceedings of the International Working Conference on Cooperating Knowledge Based Systems (CKBS)*. Springer-Verlag, 1991.
11. E. A. Emerson. Temporal and Modal Logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, pages 996–1072. Elsevier, 1990.
12. J. Ferber and P. Carle. Actors and Agents as Reflective Concurrent Objects: a MERING IV Perspective. *IEEE Transactions on Systems, Man and Cybernetics*, December 1991.
13. M. Fisher. Concurrent METATEM — A Language for Modeling Reactive Systems. In *Parallel Architectures and Languages, Europe (PARLE)*, Munich, Germany, June 1993. Springer-Verlag.
14. M. Fisher and H. Barringer. Concurrent METATEM Processes — A Language for Distributed AI. In *Proceedings of the European Simulation Multiconference*, Copenhagen, Denmark, June 1991.
15. 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, July 1992. (Published in *Lecture Notes in Computer Science*, volume 624, Springer Verlag).
16. M. Fisher and M. Wooldridge. Temporal Logic Programming for Distributed A.I. In *Twelfth International Workshop on Distributed A.I.*, Hidden Valley Resort, Pennsylvania, May 1993.
17. D. Gabbay. Declarative Past and Imperative Future: Executable Temporal Logic for Interactive Systems. In B. Banieqbal, H. Barringer, and A. Pnueli, editors, *Proceedings of Colloquium on Temporal Logic in Specification*, pages 402–450, Altrincham, U.K., 1987. (Published in *Lecture Notes in Computer Science*, volume 398, Springer Verlag).
18. D. Gabbay, A. Pnueli, S. Shelah, and J. Stavi. The Temporal Analysis of Fairness. In *Proceedings of the Seventh ACM Symposium on the Principles of Programming Languages (POPL)*, pages 163–173, Las Vegas, Nevada, January 1980.
19. L. Gasser, C. Braganza, and N. Hermann. MACE: A Flexible Testbed for Distributed AI Research. In M. Huhns, editor, *Distributed Artificial Intelligence*. Pitman/Morgan Kaufmann, 1987.
20. M. P. Georgeff and F. F. Ingrand. Decision-Making in an Embedded Reasoning System. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence (IJCAI)*, Detroit, USA, 1989. Morgan Kaufmann.
21. M. P. Georgeff and A. L. Lansky. Reactive Reasoning and Planning. In *Proceedings of the American Association for Artificial Intelligence (AAAI)*. Morgan Kaufmann, 1987.
22. D. Harel and A. Pnueli. On the Development of Reactive Systems. Technical Report CS85-02, Department of Applied Mathematics, The Weizmann Institute of Science,

Revohot, Israel, January 1985.
23. K. Konolige. *A Deduction Model of Belief.* Pitman/Morgan Kaufmann, 1986.
24. T. Maruichi, M. Ichikawa, and M. Tokoro. Modelling Autonomous Agents and their Groups. In Y. Demazeau and J. P. Muller, editors, *Decentralized AI – Proceedings of the First European Workshop on Modelling Autonomous Agents and Multi-Agent Worlds (MAAMAW)*. Elsevier/North Holland, 1990.
25. A. S. Rao and M. P. Georgeff. Modeling Agents within a BDI-Architecture. In R. Fikes and E. Sandewall, editors, *International Conference on Principles of Knowledge Representation and Reasoning (KR)*, Cambridge, Massachusetts, April 1991. Morgan Kaufmann.
26. Y. Shoham. Agent Oriented Programming. Technical Report STAN–CS–1335–90, Department of Computer Science, Stanford University, California, USA, 1990.
27. P. Wolper. The Tableau Method for Temporal Logic: An overview. *Logique et Analyse*, 110–111:119–136, June-Sept 1985.
28. M. Wooldridge. *The Logical Modelling of Computational Multi-Agent Systems.* PhD thesis, Department of Computation, UMIST, Manchester, UK, 1992.
29. M. Wooldridge and M. Fisher. A First-Order Branching Time Logic of Multi-Agent Systems. In *Proceedings of the Tenth European Conference on Artificial Intelligence (ECAI '92)*, Vienna, Austria, August 1992. Wiley and Sons.

This article was processed using the LaTeX macro package with LLNCS style