METATEM: The Story so Far

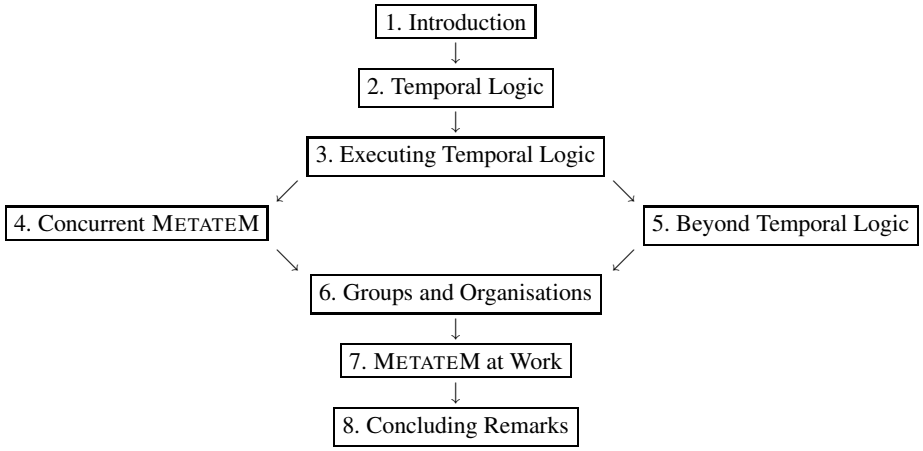
Michael Fisher

Department of Computer Science, University of Liverpool, United Kingdom
http://www.csc.liv.ac.uk/~michael

1 Introduction

METATEM is a simple programming language based on the direct execution of temporal logic statements. It was introduced through a number of papers [35,2,3] culminating in a book collecting together work on the basic temporal language [5]. However, since that time, there has been a programme of research, carried out over a number of years, extending, adapting and applying the basic approach. In particular, much of the research has concerned the development of descendents of METATEM for describing and implementing complex multi-agent systems.

Thus, while there are a number of other approaches to executing temporal statements [32,18], we will concentrate on this one particular approach and will describe the developments over the last 15 years. The structure of this article mirrors the research developments in that the path through these developments is not linear. The diagram below gives a pictorial explanation of the sections that follow.



Thus, we begin with a brief review of temporal logic itself.

2 Temporal Logic

We will begin with a review of basic temporal logic. Rather than providing an in-depth account of temporal logic, we will just provide a simple description that can be used throughout this article. For a more thorough exposition of the formal properties of this

R.H. Bordini et al. (Eds.): ProMAS 2005, LNAI 3862, pp. 3–22, 2006.

logic, see [11], while for examples of the use of temporal logic in program specification in general, see [50].

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 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 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, a 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, a semantics for the key temporal operators is given below.

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

Note that the temporal operators '$\mathcal{U}$' ("until") and '$\mathcal{W}$' ("unless") characterise intervals within the temporal sequence during which certain properties hold. Thus, '$\psi\,\mathcal{U}\,\varphi$' means that φ is 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 temporal operators relating to the *past* [49], such as '$\blacksquare$' ("always in the past"), '$\blacklozenge$' ("sometime in the past"), '$\bullet$' ("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 such operators can all be removed by translation of arbitrary temporal formulae into a specific normal form (see Section 3.2). Finally, we add a nullary operator '**start**', which is only satisfied at the "beginning of time":

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

2.1 Why Temporal Logic?

But, why do we use temporal logic? One reason is that it allows the concise expression of useful dynamic properties of individual components. For example, the formula

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

characterises a system where, once a *request* is received, a *reply* is continually sent up until the point where an *acknowledgement* is received. And, importantly, an acknowledgement is guaranteed to be received eventually. 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 components, and this relates very closely to traditional applications of temporal logic in program specification [50], the formalism is also useful for characterising properties of the overall system. For example, the formula

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

describes the message-passing behaviour within a group of components (characterised by the finite set '*Group*'). Thus, temporal logic can be used to represent both the internal behaviour of a component and the macro-level behaviour of systems.

3 Executing Temporal Logic

3.1 What Is Execution?

But, what does it mean to execute a formula, φ, of logic, L? In general, this means constructing a model, $\mathcal{M}$, for φ, i.e.

$$\mathcal{M} \models_L \varphi.$$

Typically, this construction takes place under some external constraints on φ, and many different models might satisfy φ. However, we note that:

- as φ represents a declarative statement, then producing $\mathcal{M}$ can be seen as execution in the declarative language L; and
- if φ is a specification, then constructing $\mathcal{M}$ can also be seen as prototyping an implementation of that specification.

Languages such as Prolog effectively build a form of model by attempting to refute the negation of a goal.

We execute arbitrary formulae from the temporal logic using an execution mechanism which is complete for propositional, linear temporal logic. In order to simplify the execution algorithm, arbitrary formulae are transformed into a specific normal form, called SNF (see Section 3.2). Thus, the execution algorithm works on formulae in SNF; this algorithm will be described in more detail in Section 3.3.

Although deciding propositional temporal logic formulae is complex (PSPACE-complete), deciding first-order temporal logic (FOTL) formulae is *much* worse! FOTL is incomplete (i.e. not recursively enumerable) [56,1] and so, if we wish to use arbitrary FOTL in our specifications then we are left with a few options:

1. restrict the logic and provide a 'complete' execution mechanism; or
2. execute the full logic, treating execution as simply an *attempt* to build a model for the formula.

Fragments of FOTL with 'good' properties are difficult to find and, once found [53,44], turn out to be quite restrictive. Thus, we choose (2). In addition, completeness cannot be retained in general, especially as we wish to extend the basic execution mechanism to include not only constrained backtracking, but also a dynamic model of concurrent computation and communication. Thus, in summary, execution of temporal specifications is synonymous with *attempting* to build models for such specifications.

3.2 What Is SNF?

The specification of a component's behaviour is given as a temporal formula, then transformed into a simple normal form, called Separated Normal Form (SNF) [14,19]. 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 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.

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

3.3 Execution Algorithm

The basic execution algorithm of METATEM [2,3] attempts to build a model for the formula in a simple forward-chaining fashion. The basic approach, as defined in [31,3] is described below. Assuming we are executing a set of SNF rules, R:

1. By examining the *initial* rules in R, constraints on the possible start states for the temporal model can be generated. Call these choices, C. Let 'Ω', the list of outstanding eventualities, be an empty list.
2. Make a choice from C. If there are no unexplored choices, return to a choice point in a previous state. Note that this choice mechanism takes into account a number of elements, including Ω.
   For the choice taken, generate additional eventualities, *Evs*, by checking applicability of sometime rules. Append *Evs* to Ω.
3. Generate a new state, s, from the choice made in (2) and define s as being a successor to the current state. Note that, by default, if propositions are not constrained we choose to leave them unsatisfied.
   Remove from Ω all eventualities satisfied within s.
   If s is inconsistent, or if any member of Ω has been continuously outstanding for more than $2^{5|R|}$ states, then return to (2) and select a different alternative.
4. Generate constraints on *next* states by checking applicability in s of step rules in R. Set C to be these choices. Note that C here represents all the possible choices of valuations for the next state, while Ω gives the list of eventualities that remain to be satisfied.
5. With current state, s, the set of choices on next state, C, and the list of outstanding eventualities, Ω, go to (2).

The key result here is that, under certain constraints on the choice mechanism within (2), this execution algorithm represents a decision procedure.

Theorem 1 (See [3]). *If a set of SNF rules, R, is executed using the above algorithm, with the proviso that the choice in (3) ensures that the oldest outstanding eventualities are attempted first at each step, then a model for R will be generated if, and only if, R is satisfiable.*

The above proviso ensures that, if an eventuality is outstanding for an infinite number of steps, then it will be attempted an infinite number of times. Once the choice mechanism is extended to include arbitrary ordering functions, as in [20] (see Section 5.1), then a more general version of the above theorem can be given wherein we only require a form of *fairness* on the choice mechanism. While the above proviso effectively means that we *can* potentially explore every possibility, the incorporation in the algorithm of a bound on the number of states that eventualities can remain outstanding, together with the finite model property of the logic, ensures that all of the possible states in the model can be explored if necessary.

Example. Imagine a 'car' component which can *go*, *turn* and *stop*, but can also run out of fuel (*empty*) and *overheat*.

The internal definition might be given by a temporal logic specification in SNF, for example,

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

The component's behaviour is implemented by *forward-chaining* through these formulae.

- Thus, *moving* is false at the beginning of time.
- Whenever *go* is true, a commitment to eventually make *moving* true is given.
- Whenever both *go* and *moving* are true, then either *overheat* or *empty* will be made true in the next moment in time.

4 Concurrent METATEM

4.1 From One to Many

Once we have a temporal description for a component, we can extend this to give its behaviour within an environment consisting of multiple other components. We do this simply by providing a definition of the component's interface with its environment [16]. Such an interface describes the messages that the reactive component can receive (i.e. those that come `in`) and send (i.e. those that go `out`). For example, let us consider the abstract specification of a '`car`' with the behaviour given above. This `car` can be told to *go*, *turn* and *stop*, but can also notify other components that it has run out of fuel (*empty*) or has overheated (*overheat*). A (partial) definition is given below.

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

This shows that the component recognises the messages '*go*', '*stop*' and '*turn*', and can potentially send out '*empty*' and '*overheat*' messages. Its behaviour is then specified by the temporal formula

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

Both the messages received and the messages sent are interpreted as propositions (or predicates) that can be used as part of the reactive component'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 either the component or its environment.

4.2 Communication and Concurrency

Each component executes independently. However, execution may be either *synchronous*, whereby the steps in each component occur at exactly the same time (and thus

the notion of *next* is common amongst the executing components), or *asynchronous*, whereby the steps in each component are distinct. The classification of a component's predicates as *environment* or *component* (or, indeed, *internal*) implements communication in a natural (and logical) way. Environment predicates are under the control of the component's environment, while the other categories of predicate can be made true or false by the component itself. Thus, when a component's execution mechanism makes an internal predicate true, it just records the fact in its internal memory, while when it makes a component predicate true, it also *broadcasts* a message corresponding to this predicate to all other components. Thus, in the above `car` example, when either *empty* or *overheat* is made true, then the message `empty` or `overheat`, respectively, is broadcast. If an appropriate message is received, the corresponding environment predicate set to true. In the example given above, if a `go` message is received, the proposition *go* is made true.

The use of broadcast message passing not only matches the logical view of computation but, as we will see later, also fits well with applications which concern open and dynamic systems (and where a component cannot know all the other components it might deal with).

4.3 Towards Semantics

If we consider a single METATEM component, then its semantics is effectively its temporal specification. However, once we move to a scenario with multiple components, the semantics of our system becomes more complex. If, as in [17,22], we consider '$[\![\ ]\!]$' to represent a function providing temporal semantics, then the behaviour of a system of multiple (in this case, two) components running in parallel ('$\|$'), is typically given by

$$[\![c_1 \| c_2]\!] \;=\; [\![c_1]\!] \wedge [\![c_2]\!] .$$

However, we must also be careful to distinguish the information that c_1 deals with from that which c_2 deals with. Consequently, $[\![c_1]\!]$ is *not* just the temporal rules contained within the c_1's specification and so we must enhance these rules in some way in order to capture the different components.

An obvious way to ensure that, for example, proposition p in c_1 is distinguished from p in c_2, is to rename the propositions in the component's rules ensuring that each one is 'tagged' by the name of the component in which it occurs. Thus, we would have propositions such as p_{c_1} and p_{c_2}; this is exactly the approach used in [17]. A further important aspect of the semantics given in [17] is that it allows for *either* synchronous or asynchronous models of concurrency within the framework. In the case of synchronous execution, the semantics of each component is given as a (tagged) temporal formula in a discrete, linear temporal, as described above. However, once we consider asynchronous execution, the semantics is given as a formula in the Temporal Logic of the Reals (TLR) [6], 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 component's execution.

A further complication is that, once communication is added to the synchronous case, we use

$$[\![c_1 \| c_2]\!] \;=\; [\![c_1]\!] \wedge [\![c_2]\!] \wedge \mathit{comms}(c_1, c_2) .$$

where $comms(c_1, c_2)$ is a temporal specification of the communication properties, such as broadcast message-passing. However, since once a message is broadcast from a component, then the execution of that component can no longer backtrack past such a broadcast, the semantics of communicating components becomes much more complex [60].

5 Beyond Temporal Logics

So far, we have seen how specifications given in temporal logic can be *directly executed* in order to animate the component's behaviour. Thus, this approach provides a high-level programming notation, while maintaining a close link between the program and its specification. However, when moving towards the representation and execution of *agents* [61], particularly *rational agents* [62,58], we naturally want to represent more than an agent's basic temporal behaviour. Basic agents are just autonomous components and these can be modelled, at a simple level, by Concurrent METATEM as described above [16,15,33].

Although the central aspect of an agent is *autonomy*, rational agents are agents that have *reasonable* and *explainable* courses of action that they undertake. The key concept that rational agents have brought to the forefront of software design is that, as well as describing *what* an agent does, it is vital to describe *why* it does it. Hence the need to represent the reasons for certain autonomous behaviour within agents.

In line with the BDI framework [54], and with other rational agent theories [57], it is important to represent not only an agent's temporal behaviour, but also

- its informational aspects, such as what it *believes* or *knows*,
- its motivational aspects, such as its *goals* or *intentions*, and
- its deliberative aspects, such as *why* it makes the choices it does.

Thus, inspired by the success of the BDI framework [54] in representing deliberation, the basic METATEM system was extended, in [20], with information representation, in terms of *beliefs* and explicit mechanisms for ordering goals. The representation of belief was given by extending the temporal basis with a standard modal logic having Kripke semantics [40]. Thus, during execution, modal formulae related to belief were decided again by a forward chaining process. This allowed more complex (and mixed) formulae such as

$$\begin{aligned} happy &\Rightarrow Bx \\ x &\Rightarrow \bigcirc rich\,. \end{aligned}$$

Goals, corresponding to both desires and intentions in the BDI model were, in turn, represented by temporal eventualities. This then allowed deliberation to be represented via user defined functions providing an ordering on the satisfaction of eventualities. This is best explained with an example.

5.1 Deliberation Via Goal Re-ordering

Recall that, in the basic METATEM execution above, outstanding eventualities are stored in an age-ordered (i.e. oldest-first) list that is passed on to the next execution state. For example, consider an agent that has the following eventualities it needs to satisfy

$$[\Diamond be\_famous, \Diamond sleep, \Diamond eat\_lunch, \Diamond make\_lunch]\,.$$

Now, if these were passed on to the next execution state, the standard approach would be to execute these oldest-first, i.e attempt to make *be_famous* true, then attempt to make *sleep* true, and so on. In [20], the ability to *re-order* this list before it is passed on to the next execution state was provided. Now, in the example above, imagine the agent actually re-ordered the list in terms of what it considered to be most important, e.g.

$$[\Diamond be\_famous, \Diamond eat\_lunch, \Diamond sleep, \Diamond make\_lunch] .$$

Indeed, the agent could re-order the list further if it had an idea about what it could actually achieve, e.g. what it knew *how* to make true. Thus, if the agent had no way (i.e. no plan) to make *be_famous* true, it might move it to the end of the list. If the agent knew how to make *eat_lunch* true, it might move it towards the front of the list. However, it might also be able to infer that a pre-condition of making *eat_lunch* true is to make *make_lunch* lunch true and so it might well move this to the front of the list. Thus, the list passed on to the next execution state is then

$$[\Diamond make\_lunch, \Diamond eat\_lunch, \Diamond sleep, \Diamond be\_famous] .$$

5.2 Resource-Bounded Reasoning

While the above provides a simple and concise mechanism for representing and implementing deliberative agents, it does not deal with a further important aspect of 'real' agents, namely their resource-bounded nature [9]. In particular, the representation of belief was given by extending the temporal basis with a standard modal logic having Kripke semantics [40]. As is well known, this does not match the resource-bounded nature of 'real' reasoners [39]. Indeed modal logics generally model logically omniscient agents which are forced to believe (and compute) all the logical consequences of their own beliefs.

Thus, in [23], we modified the METATEM execution framework so that it was based on *multi-context* logics of belief [10,37,36] rather than traditional modal logics. Multi-context logics [38,7] allow much finer control of the reasoning processes. In using this basis for belief in METATEM, we then have much finer control over how beliefs are explored, executed and reasoned about. In particular, we can control how much reasoning an agent carried out during its execution, and this allows us to capture an element of resource-boundedness.

This work was applied, for example to parts of the RoboCup scenario [24,25], and was extended to incorporate bounds on the temporal resourced consumed, i.e. the agent can reason about what it might do in the future, but the distance it can reason into the future about is limited [34].

A final modification brought in the concepts of *ability* and *confidence* [26], giving us the ABC framework which incorporates:

– **A**bility — captured in a very simple modal extension, where $A_i\varphi$ intuitively means "*i* is able to do φ", for example,

$$A_{me} buy\_ticket \Rightarrow \bigcirc buy\_ticket$$

– **B**elief — captured by multi-context belief with potential resource-bounds, where B_i represents the beliefs of agent i, for example

$$buy\_ticket \Rightarrow B_{me} lottery\_winner$$

– **C**onfidence — captured by the combination of B_i and $\Diamond$, where "agent i is confident of φ' is given by $B_i \Diamond \varphi$.

The idea of confidence replaces the use of just '$\Diamond$' to represent goals. Expressing confidence in terms of belief about the future not only keeps the logic simple, it allows us to express weaker motivational attitudes. This confidence can be used as an agent's reason for doing something. Importantly, confidence can come not only from the agent itself, but also from the agent's confidence *in other agents in this system*. This allows us to use the notion of confidence in a number of ways within both individual agents and multi-agent systems [26].

This also leads us on to considering more complex multi-agent systems.

6 Groups and Organisations

In Concurrent METATEM we introduced the idea of multiple, asynchronously executing, agents (let us call them this, rather than 'components' from now on) communicating through broadcast message-passing. Although this is a good model from a logical point of view it has two disadvantages: (1) in practice, broadcasting to large numbers of agents is costly, and (2) the multi-agent system is essentially flat and unstructured, and so it is difficult to represent more complex systems using this approach.

Thus, as Concurrent METATEM has developed, a range of structuring mechanisms for the agent space have been developed. All are based on the notion of *groups*, derived from both social constructs and ideas in distributed operating systems [8]. Thus all agents can occur in multiple groups [15], while groups themselves are dynamic and open, and may contain sub-groups and each agent may be a member of several groups. When an agent broadcasts a message it is restricted to being sent to members of (some of) the groups the agent occurs within. Hence, groups are useful both for restricting the extent of broadcast and structuring the agent space, effectively replacing full broadcast by a form of *multicast* message-passing.

However, a basic grouping mechanism is not enough. We often want to define different computational properties for the different groups the agent might be a member of. Thus, from initial ideas concerning additional logical properties that groups might have [21], we developed the view that agents and groups are *exactly* the same entities [30,28]. Thus, the notion of a group as essentially being a container, where agents have behaviour and groups contain agents, i.e.,

$$Agent ::= Behaviour : Spec$$
$$Group ::= Contents : \mathcal{P}(Agent)$$

became the notion that all agents have the potential to contain others, just as all have the potential to have behaviour, i.e.

$$\begin{aligned} Agent ::= & \; Behaviour : Spec \\ & Contents : \mathcal{P}(Agent) \\ & Context : \mathcal{P}(Agent) \end{aligned}$$

Thus, we can think of several varieties of agent:

- A simple agent: $Contents = \emptyset$.
- A simple group: $Behaviour = \emptyset$.
- A more complex group: $Contents \neq \emptyset$ and $Behaviour \neq \emptyset$.

Thus groups, rather than being mere *containers*, can now have behaviours, captured by their internal policies and rules. In particular, agents can control the communication policies, organisation policies, etc., within their Contents. Once agents and groups are the same entities, a number of aspects become clear. For example, since agents are opaque (i.e., their internal structure is hidden from other agents) it is not obvious whether agents have certain abilities natively, or merely make use of other (internal) agents. In addition, any agent has the potential to dynamically become a group! In particular, all agents respond to '`addToContent`' and '`addToContext`' messages (two primitive operations accessing the *Content* and *Context* aspects of an agent), and all agents can clone themselves (and perform a shallow copy of their contents), as well as terminate themselves or merge with another agent.

6.1 Building Organisations

With the structures in place to support more complex multi-agent systems, the question remains: how can we use the logical apparatus, such as the Ability, Belief and Confidence framework, in order to program agents to dynamically form complex structures. In a number of papers, mechanisms for describing and constructing such complex groups, teams and organisations have been described [27,28,29].

Let us give an outline using an example. Suppose that agent i wishes for φ to occur ($B_i \Diamond \varphi$), but does not have the corresponding ability ($\neg A_i \varphi$). Within the agent's behaviour we might have a rule such as

$$B_i \Diamond \varphi \wedge \neg A_i \varphi \Rightarrow \bigcirc send_i(A_? \varphi)$$

In this case the agent sends out a message asking for help from any agent that is able to achieve φ. Various agents, depending on their own goals, might choose to reply that they have the required ability (namely, the ability to achieve φ, i.e. $A\varphi$). Then the original (sending) agent has a choice about how to deal with these replies and, consequently, how to deal with the agents who might be of help. For example [27], the sender might

- invite relevant agents to join its *Content*,
- create a new "dedicated" agent to serve as a container for agents that share the relevant ability, or
- join a group that can help it solve its problem,

All of these can be supported through the flexible group structure and can be implemented using logical rules. Each gives a very different organisation structure.

7 METATEM at Work

Here we outline a number of examples showing how METATEM can be used. Note that we only provide an overview of each example — full details can usually be found in the cited papers.

7.1 Train Signalling

In [13], METATEM was used to model a simple railway signalling scenario. Here, the system was modelled as one large set of SNF rules. Thus there was not the separation one might expect with Concurrent METATEM. The SNF specification is then executed to animate the rail simulation. The specification consists of predicates relating to stations, trains and lines, and the key specifications concern each station's control of the trains that come into it and each trains request for entry to a station. Each of the stations 'knows' which lines it has connecting itself to other stations, what trains the station *has*, and which lines each train runs on. Consequently, there are rules such as[1]:

- $[station(S) \wedge moved(T,S)] \Rightarrow \bigcirc has(S,T)$
  i.e., the station (S) now has the train (T) if T moved to S in the last step;
- $[station(S) \wedge request(T,S)] \Rightarrow \Diamond permit(T,S)$
  i.e., if station S receives a request from train T to enter S, then the station guarantees to *permit* this move at some point in the future;
- $[station(S) \wedge has(S,T) \wedge \neg moved(T,New)] \Rightarrow \bigcirc has(S,T)$
  i.e., if station S has train T and T does not move, then S will still have T in the next step;
- $[station(S) \wedge permit(T1,S) \wedge permit(T2,S)] \Rightarrow (T1 == T2)$
  i.e., if a station permits two trains, $T1$ and $T2$, to enter, then $T1$ and $T2$ must actually be the same train.

7.2 Patient Monitoring

In [55], Reynolds applies METATEM to the modelling and animation of a patient monitoring system (PMS). Temporal formulae are used to specify, for example, under what conditions (i.e. what patient vital signs) an alarm should be sounded. Here, the specification is split into separate components, e.g.

```
nurse(alarm, display)[act, seen, req]
```

Here, for example, the message `req` is sent out by the `nurse` requesting information. SNF rules then characterise the internal behaviour of such components:

- $[wtr(P) \wedge \neg req(P)] \Rightarrow \bigcirc wtr(Q)$
  i.e., if the nurse is "waiting to request information" (*wtr*) about a patient, but does not request such information (*req*) then the nurse will continue to wait to request information in the next step;

[1] Assume, here, that all variables beginning with upper-case letters and all variables are universally quantified.

- $[wtr(P) \wedge req(P) \wedge next(P,Q)] \Rightarrow \bigcirc wtr(P)$
  i.e., if the nurse is "waiting to request information" about a patient, and does also request this information, then the nurse moves on to "waiting to request information" from the *next* patient;
- $alarm(P) \Rightarrow \bigcirc act$
  i.e., if an alarm sounds for a patient, the nurse will act.

7.3 Economic Games

We next outline a simple economic game, a simplified variation of the Nash [51] demand game. Here, two synchronous agents make bids to an arbiter who gives out rewards (in line with a specific matrix). Agents usually bid based on the best previous strategy, but sometimes (quite rarely) can choose a random value. In our case, the `arbiter` agent implements the reward matrix:

`agent1` bid :	1	2	3	1	2	3	1	2	3
`agent2` bid :	1	1	1	2	2	2	3	3	3
`agent1` reward :	1	2	3	1	2	0	1	0	0
`agent2` reward :	1	1	1	2	2	0	3	0	0

Thus, sample `arbiter` code includes (for brevity we replace $agent1$ by $a1$, and $agent2$ by $a2$, respectively)

$$[bid(a1,V1) \wedge bid(a2,V2) \wedge (4 < (V1+V2))] \Rightarrow \bigcirc reward(a1,V1,0)$$
$$[bid(a1,V1) \wedge bid(a2,V2) \wedge (4 < (V1+V2))] \Rightarrow \bigcirc reward(a2,V2,0)$$
$$[bid(a1,V1) \wedge bid(a2,V2) \wedge (4 \geq (V1+V2))] \Rightarrow \bigcirc reward(a1,V1,V1)$$
$$[bid(a1,V1) \wedge bid(a2,V2) \wedge (4 \geq (V1+V2))] \Rightarrow \bigcirc reward(a2,V2,V2)$$

The `arbiter` agent receives `bid` messages from each of the agents and then sends out `reward` messages. Note that the `reward` sent from the arbiter back to bidding agents contains the bid made in its second argument.

As we can see from the above, these rules make use of much more arithmetical operators. The rules within the bidding agents use even more. Bidding agents send out bids and receive rewards. Internally, they keep track of which bids (1, 2, or 3) have generated the most rewards. In addition, a small random element is introduced so that the bidding agent can choose a different bid, rather than just the most successful so far. Some sample rules for the bidding agents are[2]:

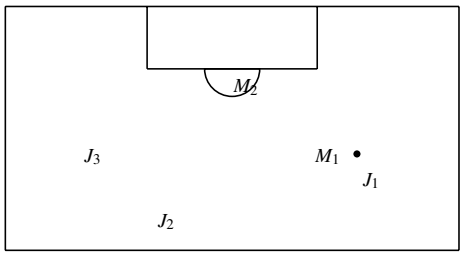
$$rand(V) \Rightarrow \bigcirc seed(V)$$
$$[seed(V) \wedge (V \leq 3072) \wedge (V1 == (V\%3+1))] \Rightarrow \bigcirc bid(agent1,V1)$$
$$[seed(V) \wedge (3072 < V) \wedge bestsofar(V1)] \Rightarrow \bigcirc bid(agent1,V1)$$

Once executing, even though small perturbations (i.e. random bids outside the optimal) are introduced, the bidding stabilises so that the agent's bid is based on the accumulated history and this isn't significantly distorted by small numbers of random bids. In the case of our reward matrix above, both bidding agent settle down to bidding '2'.

[2] Note that $rand(V)$ binds V to a random number between 0 and 2^{16} and $V\%3+1$ is V modulo 3, with 1 then added (thus giving a result of either 1, 2 or 3).

7.4 Resource-Bounded Deliberation

In this example, taken from [24], we have a simple football scenario. Pictorially, we have the following situation.



The 'J' team are attacking the goal, while the 'M' team are defending it. Agent J_1 has the ball but has two abilities: to shoot towards goal, or to pass to a team-mate. Motivated by the aim of scoring, J_1 has to decide what to do.

Now, rather than giving the program rules (which are quite complex), we outline the reasoning process that J_1 goes through.

- We again note that J_1 has two things it can do: pass or shoot. Initially, before any reasoning is carried out, J_1 has a slight (in-built) preference for shooting.
- However, J_1 begins to reason about its options, about its beliefs about its team-mates, about its beliefs about its team-mates beliefs, etc. Given sufficient reasoning time, J_1 can work out that the best approach is to pass to J_2 who, J_1 believes, will then pass on to J_3. This is based on the belief that J_3 has the best chance of scoring, and that J_2 believes this also.
  Thus, given sufficient reasoning time, J_1 will choose to pass rather than shoot.
- However, in time-constrained situations, such as near the end of the game, we can put a *bound* on the amount of reasoning concerning belief that J_1 is able to carry out. We do this by fixing a maximum depth of nested beliefs that J_1 is able to deal with.
  In such a scenario, J_1 does not have enough (reasoning) time to work out that passing is the best option.
  Consequently, in time-constrained situations, such as this, J_1 chooses to shoot.

It is important to note here that the program rules are exactly the same for both options. All that has happened is that a belief bound is changed, yet this has the significant effect of leading the agent to prefer one action over another.

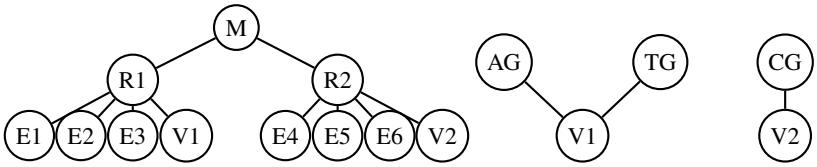
7.5 Active Museum

In [42] a particular scenario from pervasive/ubiquitous computing was examined. This is the idea of an *Active Museum* [52]. Here, a museum provides visitors with electronic guides (such as PDAs) and these guides can be programmed with the visitor's preferences. As the visitor moves through the museum, the rooms, exhibits, etc., can all interact with the PDA and, in this way, the PDA can advise the visitor what to look at next.

We model this via three aspects:

- the organisational structure of rooms, exhibits, visitors, etc.;
- the organisational structure of the agent's interests; and
- the rule(s) within the visitor agent concerning deliberation.

Thus, the group structure, combining physical aspects and museum interests can be represented as



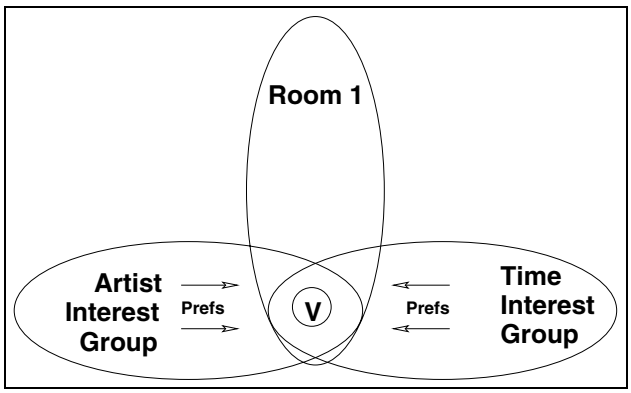
For example, visitor `V1` is within the group representing room `R1` but is also within the groups representing the two interest groups relating to a particular artist (`AG`) and to time (`TG`). Now, those groups provide the context for the visitor agent and broadcast important information to it.

Each visitor agent effectively only has one rule:

$$[canSee(Exhibit) \wedge \neg exclude(Exhibit)] \Rightarrow \Diamond lookAt(Exhibit)$$

Thus, if the visitor agent can see an exhibit and is not excluded (by one of its interest groups) from looking at it, it will eventually look at it. Note that `canSee` messages are broadcast by the agent's context, and so moving context changes what the agent does.

While visitor agents have one main rule, the important aspect concerns the deliberation of the visitor agent about what order to visit the exhibits in. In this, the agent utilises preferences to implement the deliberative re-ordering of eventualities seen earlier. Preferences are *very* simple, e.g. `prefer(E3,E1)`. Pictorially, we have:



Some preferences are internal to the agent; most are obtained by the agent from its Context. Thus, the preferences help the visitor agent decide between eventualities. Moving between rooms changes what the agent can see, but it is moving between interest groups that changes what the agent prefers to look at first.

8 Concluding Remarks

This article has provided an overview of the work that has been carried out on executable temporal and modal logics based on the METATEM approach. This has led to several practical systems developed over the years. Initially, Owens developed a basic METATEM system in Prolog, while Fisher developed a Concurrent METATEM harness in C++; early implementation techniques were described in [31]. The developments on agent representation and execution were implemented in Prolog [20] and, in parallel, more efficient mechanisms for the implementation of Concurrent METATEM were examined [47,46]. Most recently, the ABC framework has be implemented in Java [43].

But, what of the future? There are several areas actively being investigated at present, ranging from theoretical to practical and from single agent to multi-agent. These themes are outlined below.

- The broader application, particularly of the ABC approach, to the modelling and simulating of various forms of organisations. In particular, the development of virtual organisations (along the lines of the Active Museum example) and applications in pervasive and ubiquitous computing.
  This mainly involves practical application, but also involves more work on ad-hoc team formation and on probabilistic ABC, some of which is already under way [12].
- The development of a lightweight implementation, based on J2ME[3] and use of the above approaches in resource constrained environments, e.g. mobile, wearable, devices.
- Extending the theoretical work on high-level semantics of the group approach, which was recently investigated in [43] where groups were shown to match Milner's bigraphs [45] in many ways, while adapting earlier work on the use of Concurrent METATEM as a high-level coordination language [48] for use in multi-agent and pervasive applications.
- Developing a more practical (and flexible) approach to meta-level programming in METATEM. Based on [4], but extended for the ABC framework, and allowing the use of both complex meta-level adaption and simpler, preference-based, deliberation (as in [42]).

Through previous and future work our intention is to continue to develop a framework that utilises formal logic in the specification, verification and implementation of reactive [41] and multi-agent [59] systems. This framework comprises

- a logic (typically based on a combination of simpler logics) in which the high-level behaviours (of both agent and organisation) can be concisely specified, and
- a programming language providing flexible and practical concepts close to the specification notation used.

Acknowledgements

Most of the work outlined in this article was carried out in collaboration with others, and so thanks go to Howard Barringer, Nivea de Carvalho Ferreira, Marcello Finger, Dov

[3] `http://java.sun.com/j2me`

Gabbay, Chiara Ghidini, Graham Gough, Benjamin Hirsch, Wiebe van der Hoek, Tony Kakoudakis, Adam Kellett, Richard Owens, Mark Reynolds, and Mike Wooldridge.

References

1. M. Abadi. The Power of Temporal Proofs. *Theoretical Computer Science*, 64:35–84, 1989.
2. 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).
3. H. Barringer, M. Fisher, D. Gabbay, G. Gough, and R. Owens. METATEM: An Introduction. *Formal Aspects of Computing*, 7(5):533–549, 1995.
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, M. Fisher, D. Gabbay, R. Owens, and M. Reynolds, editors. *The Imperative Future: Principles of Executable Temporal Logics*. Research Studies Press, Chichester, United Kingdom, 1996.
6. 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, January 1986.
7. M. Benerecetti, A. Cimatti, E. Giunchiglia, F. Giunchiglia, and L. Serafini. Formal Specification of Beliefs in Multi-Agent Systems. In J. P. Müller, M. J. Wooldridge, and N. R. Jennings, editors, *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.
8. K. P. Birman. The Process Group Approach to Reliable Distributed Computing. Techanical Report TR91-1216, Department of Computer Science, Cornell University, July 1991.
9. M. E. Bratman, D. J. Israel, and M. E. Pollack. Plans and Resource-Bounded Practical Reasoning. *Computational Intelligence*, 4:349–355, 1988.
10. A. Cimatti and L. Serafini. Multi-Agent Reasoning with Belief Contexts: the Approach and a Case Study. In M. Wooldridge and N. R. Jennings, editors, *Intelligent Agents: Theories, Architectures, and Languages (LNAI Volume 890)*, pages 71–85. Springer-Verlag: Heidelberg, Germany, January 1995.
11. E. A. Emerson. Temporal and Modal Logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, pages 996–1072. Elsevier, 1990.
12. N. de C. Ferreira, M. Fisher, and W. van der Hoek. A Logical Implementation of Uncertain Agents. In *Workshop on Multi-Agent Systems: Theory and Applications (MASTA)*, Lecture Notes in Artificial Intelligence. Springer-Verlag, 2005.
13. M. Finger, M. Fisher, and R. Owens. METATEM at Work: Modelling Reactive Systems Using Executable Temporal Logic. In *Sixth International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems (IEA/AIE)*, Edinburgh, U.K., June 1993. Gordon and Breach Publishers.
14. M. Fisher. A Normal Form for First-Order Temporal Formulae. In *Proceedings of Eleventh International Conference on Automated Deduction (CADE)*, Saratoga Springs, New York, June 1992. (Published in *Lecture Notes in Computer Science*, volume 607, Springer-Verlag).
15. M. Fisher. A Survey of Concurrent METATEM — The Language and its Applications. In *First International Conference on Temporal Logic (ICTL)*, Bonn, Germany, July 1994. (Published in *Lecture Notes in Computer Science*, volume 827, Springer-Verlag).

16. M. Fisher. Representing and Executing Agent-Based Systems. In M. Wooldridge and N. R. Jennings, editors, *Intelligent Agents*. Springer-Verlag, 1995.
17. M. Fisher. A Temporal Semantics for Concurrent METATEM. *Journal of Symbolic Computation*, 22(5/6), November/December 1996.
18. M. Fisher. An Introduction to Executable Temporal Logics. *Knowledge Engineering Review*, 11(1):43–56, March 1996.
19. M. Fisher. A Normal Form for Temporal Logic and its Application in Theorem-Proving and Execution. *Journal of Logic and Computation*, 7(4), August 1997.
20. M. Fisher. Implementing BDI-like Systems by Direct Execution. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*. Morgan-Kaufmann, 1997.
21. M. Fisher. Representing Abstract Agent Architectures. In J. P. Müller, M. P. Singh, and A. S. Rao, editors, *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, 1999.
22. M. Fisher. Temporal Development Methods for Agent-Based Systems. *Journal of Autonomous Agents and Multi-Agent Systems*, 10(1):41–66, January 2005.
23. M. Fisher and C. Ghidini. Programming Resource-Bounded Deliberative Agents. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*. Morgan Kaufmann, 1999.
24. 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, 2000.
25. M. Fisher and C. Ghidini. Specifying and Implementing Agents with Dynamic Resource Bounds. In *Proceedings of Second International Cognitive Robotics Workshop*, Berlin, Germany, 2000.
26. M. Fisher and C. Ghidini. The ABC of Rational Agent Programming. In *Proc. First International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 849–856. ACM Press, July 2002.
27. M. Fisher, C. Ghidini, and B. Hirsch. Organising Logic-Based Agents. In M. Hinchey, J. Rash, W. Truszkowski, C. Rouff, and D. Gordon-Spears, editors, *Formal Approaches to Agent-Based Systems, Second International Workshop, FAABS 2002, Greenbelt, MD, USA, October 29-31, 2002, Revised Papers*, volume 2699 of *Lecture Notes in Computer Science*, pages 15–27. Springer, 2003.
28. M. Fisher, C. Ghidini, and B. Hirsch. Organising Computation through Dynamic Grouping. In *Objects, Agents and Features*, volume 2975 of *Lecture Notes in Computer Science*, pages 117–136. Springer-Verlag, 2004.
29. M. Fisher, C. Ghidini, and B. Hirsch. Programming Groups of Rational Agents. In *Computational Logic in Multi-Agent Systems (CLIMA-IV)*, volume 3259 of *849–856*. Springer-Verlag, November 2004.
30. M. Fisher and T. Kakoudakis. Flexible Agent Grouping in Executable Temporal Logic. In Gergatsoulis and Rondogiannis, editors, *Intensional Programming II*. World Scientific Publishing Co., March 2000.
31. 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).
32. M. Fisher and R. Owens, editors. *Executable Modal and Temporal Logics*, volume 897 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, Heidelberg, Germany, February 1995.
33. M. Fisher and M. Wooldridge. A Logical Approach to the Representation of Societies of Agents. In N. Gilbert and R. Conte, editors, *Artificial Societies*. UCL Press, 1995.

34. M. Fisher and C. Ghidini. Agents with Bounded Temporal Resources. *Lecture Notes in Computer Science*, 2403:169–??, 2002.
35. 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).
36. C. Ghidini. Modelling (Un)Bounded Beliefs. In *Proc. Second International and Interdisciplinary Conf. on Modeling and Using Context (CONTEXT)*, Trento, Italy, 1999.
37. F. Giunchiglia and C. Ghidini. Local Models Semantics, or Contextual Reasoning = Locality + Compatibility. In *Proceedings of the Sixth International Conference on Principles of Knowledge Representation and Reasoning (KR'98)*, pages 282–289, Trento, 1998. Morgan Kaufmann. Long version forthcoming in "Artificial Intelligence".
38. F. Giunchiglia and L. Serafini. Multilanguage Hierarchical Logics (or: how we can do without modal logics). *Artificial Intelligence*, 65:29–70, 1994. Also IRST-Technical Report 9110-07, IRST, Trento, Italy.
39. F. Giunchiglia, L. Serafini, E. Giunchiglia, and M. Frixione. Non-Omniscient Belief as Context-Based Reasoning. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence (IJCAI)*, pages 548–554, Chambery, France, 1993. Also IRST-Technical Report 9206-03, IRST, Trento, Italy.
40. J. Y. Halpern and Y. Moses. A Guide to Completeness and Complexity for Modal Logics of Knowledge and Belief. *Artificial Intelligence*, 54:319–379, 1992.
41. 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.
42. B. Hirsch, M. Fisher, C. Ghidini, and P. Busetta. Organising Software in Active Environments. In *Computational Logic in Multi-Agent Systems (CLIMA-V)*, volume 3487 of *Lecture Notes in Computer Science*. Springer-Verlag, 2005.
43. B. Hirsch. *Programming Rational Agents*. PhD thesis, Department of Computer Science, University of Liverpool, United Kingdom, May 2005.
44. I. Hodkinson, F. Wolter, and M. Zakharyashev. Decidable Fragments of First-Order Temporal Logics. *Annals of Pure and Applied Logic*, 2000.
45. O. H. Jensen and R. Milner. Bigraphs and Mobile Processes (revised). Technical Report UCAM-CL-TR-580, Computer Lab, Cambridge University, U.K., 2004.
46. A. Kellett. *Implementation Techniques for Concurrent* METATEM. PhD 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)*. IEEE Press, May 1997.
48. A. Kellett and M. Fisher. Coordinating Heterogeneous Components using Executable Temporal Logic. In Meyer and Treur, editors, *Agents, Reasoning and Dynamics*, Vol. 6 in Series of Handbooks in Defeasible Reasoning and Uncertainty Management Systems. Kluwer Academic publishers, 2001.
49. O. Lichtenstein, A. Pnueli, and L. Zuck. The Glory of the Past. *Lecture Notes in Computer Science*, 193:196–218, June 1985.
50. Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification*. Springer-Verlag, New York, 1992.
51. J. F. Nash. Two-person Cooperative Games. *Econometrica*, 21:128–140, 1953.
52. O. Stock and M. Zancanaro. Intelligent Interactive Information Presentation for Cultural Tourism. In *Proceedings of the International CLASS Workshop on Natural Intell igent and Effective Interaction in Multimodal Dialogue Systems*, Copenhagen, Denmark, 28-29 June 2002.

53. R. Pliuskevicius. On the Completeness and Decidability of a Restricted First Order Linear Temporal Logic. In *LNCS 1289*, pages 241–254. Springer-Verlag, 1997.
54. 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.
55. M. Reynolds. METATEM in Intensive Care. Technical Report tr-97-01, Kings College, London, 1997.
56. A. Szalas and L. Holenderski. Incompleteness of First-Order Temporal Logic with Until. *Theoretical Computer Science*, 57:317–325, 1988.
57. B. van Linder, W. van der Hoek, and J. J. Ch. Meyer. How to Motivate your Agents. In M. Wooldridge, J. P. Müller, and M. Tambe, editors, *Intelligent Agents II (LNAI 1037)*, pages 17–32. Springer-Verlag: Heidelberg, Germany, 1996.
58. M. Wooldridge. *Reasoning about Rational Agents*. MIT Press, 2000.
59. M. Wooldridge. *An Introduction to Multiagent Systems*. John Wiley & Sons, 2002.
60. M. Wooldridge, J. Bradfield, M. Fisher, and M. Pauly. Game-Theoretic Interpretations of Executable Logic. (Unpublished paper.).
61. M. Wooldridge and N. R. Jennings. Intelligent Agents: Theory and Practice. *The Knowledge Engineering Review*, 10(2):115–152, 1995.
62. M. Wooldridge and A. Rao, editors. *Foundations of Rational Agency*. Applied Logic Series. Kluwer Academic Publishers, March 1999.