Towards the Refinement of Executable Temporal Objects

Michael Fisher
Department of Computing, Manchester Metropolitan University
Manchester M1 5GD, U.K. EMAIL: M.Fisher@doc.mmu.ac.uk

Abstract
Concurrent METATEM is a high-level language in which the behaviour of an individual reactive component is represented by a temporal logic formula and is animated by direct execution. The combination of this executable temporal formalism, together with an operational model providing asynchronous concurrency and broadcast message-passing, presents a powerful and flexible framework in which to develop concurrent object-based, particularly agent-based, applications.

While Concurrent METATEM has been applied in a variety of scenarios, and techniques for the verification of properties of Concurrent METATEM systems have been developed, little work has been carried out on the basis for refining such systems. Here, we introduce simple mechanisms for the refinement both of an object's internal behaviour and interface, and of individual objects into new systems of communicating objects.

Keywords
Executable specifications, refinement, object-based systems, transformation

1 INTRODUCTION

Concurrent METATEM is a simple programming language developed for reactive systems (Fisher 1993) that has been shown to be particularly useful in representing and developing multi-agent systems (Fisher 1995*a*). It is based on the combination of two complementary elements: the direct execution of temporal logic specifications providing the behaviour of an individual object (Fisher 1996); and a concurrent operational model in which such objects execute asynchronously, communicate via broadcast message-passing, and are organised using a powerful grouping mechanism (Fisher 1994). While both the operational model and object representation technique are simple, they together provide a framework in which a variety of concurrent object-based systems can be specified and implemented. It is important to note that *object-based*, rather than *object-oriented*, systems are developed here, and so we are not directly concerned with features such as inheritance and classes; the only attributes these objects have are encapsulation and message-based communication. Note, however, that many 'standard' object-oriented features can be built on top of this basic framework, if required. In this sense, the closest related work stems from the development of the Actor paradigm (Agha 1986).

In this paper we consider the refinement of Concurrent METATEM, incorporating

1. the refinement of an individual object's behaviour, which corresponds to standard refinement of temporal specifications (Manna & Pnueli 1992);
2. the refinement of an object into a collection of new objects that together implement the original behaviour under appropriate communication constraints; and
3. the use of a fixed set of transformation rules, rather than arbitrary refinements, allowing a "pick and mix" approach to program development.

For simplicity, we consider propositional, rather than first-order, temporal specifications. While this is obviously a restriction, many of the techniques we discuss can be transferred, with a little work, to the first-order framework.

The structure of this paper is as follows. In §2, we provide a definition of the temporal logic we use, followed, in §3, by a brief review of the Concurrent METATEM language. In §4, we present the framework for refinement of Concurrent METATEM objects, and consider a range of simple examples. We also derive *fixed* transformations which are behaviour preserving, and present a larger example of system refinement. Finally, in §5, we present conclusions and identify future work.

2 TEMPORAL LOGIC

Temporal logic can be seen as classical logic extended with various modalities representing temporal aspects of logical formulae (Emerson 1990). The propositional temporal logic we use (called PTL) is based on a linear, discrete model of time (Gabbay, Pnueli, Shelah & Stavi 1980). Thus, time is modelled as an infinite sequence of discrete states, with an identified starting point, called 'the beginning of time'. Classical formulae are used to represent constraints *within* states, while temporal formulae represent constraints *between* states. This temporal logic can be seen as classical logic extended with various modalities, for example '$\Diamond$', '$\Box$', and '$\bigcirc$'. The intuitive meaning of these connectives is as follows: $\Diamond A$ is true now if A is true *sometime* in the future; $\Box A$ is true now if A is true *always* in the future; and $\bigcirc A$ is true now if A is true at the *next* moment in time. In this presentation, similar connectives are introduced to enable reasoning about the *past* (Lichtenstein, Pnueli & Zuck 1985).

2.1 Syntax

We begin with the formal syntax of the language. Formulae of PTL are constructed using the following symbols.

- A set, $\mathcal{L}_p$, of *propositional symbols* represented by strings of lower-case alphabetic characters.
- Classical connectives, $\neg$, $\vee$, $\wedge$, **true**, **false** and $\Rightarrow$.
- Future-time temporal operators, categorised as
  - nullary operators: **start**,
  - unary operators: $\bigcirc$, $\Diamond$, $\Box$,
  - binary operators: $\mathcal{U}$, and $\mathcal{W}$.
- Past-time temporal operators, categorised as
  - unary operators: ⊙, ◆, ■,
  - binary operators: $\mathcal{S}$, and $\mathcal{Z}$.

The set of *well-formed formulae* of PTL (WFF_p) is defined as follows.

- Any element of $\mathcal{L}_p$ is in WFF_p.
- If A and B are in WFF_p, then so are

$\neg A$	$A \vee B$	$A \wedge B$	$A \Rightarrow B$	**start**
$\Diamond A$	$\Box A$	$A \mathcal{U} B$	$A \mathcal{W} B$	$\bigcirc A$
◆A	■A	$A \mathcal{S} B$	$A \mathcal{Z} B$	⊙A

2.2 Semantics

Intuitively, the models for PTL formulae are based on discrete, linear structures having a finite past and infinite future, i.e., sequences such as

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

where each s_i, called a *state*, provides a propositional valuation. However, rather than representing the model structure in this way, we will define a model, σ, as

$$\sigma = \langle \mathbb{N}, \pi_p \rangle$$

where $\mathbb{N}$ is used to represent the sequence of states $s_0, s_1, s_2, s_3, \ldots$, and, π_p is a map from $\mathbb{N} \times \mathcal{L}_p$ to $\{\mathsf{T}, \mathsf{F}\}$, giving a propositional valuation for each state in the sequence.

An interpretation for this logic is defined as a pair $\langle \sigma, i \rangle$, where σ is the model and i the index of the state at which the temporal statement is to be interpreted.

A semantics for well-formed temporal formulae is a relation between interpretations and formulae, and is defined inductively as follows, with the (infix) semantic relation being represented by '$\models$'. The semantics of a proposition is defined by the valuation given to that proposition at a particular state:

$$\langle \sigma, i \rangle \models p \quad \text{iff} \quad \pi_p(i, p) = \mathsf{T} \qquad [\text{for } p \in \mathcal{L}_p].$$

The semantics of the standard propositional connectives is as in classical logic, e.g.,

$$\langle\sigma, i\rangle \models A \lor B \quad \text{iff} \quad \langle\sigma, i\rangle \models A \quad \text{or} \quad \langle\sigma, i\rangle \models B\,.$$

The semantics of the unary future-time temporal operators is defined as follows.

$$\begin{array}{lll}
\langle\sigma, i\rangle \models \bigcirc A & \text{iff} & \langle\sigma,\, i+1\rangle \models A \\
\langle\sigma, i\rangle \models \Diamond A & \text{iff} & \text{there exists } j \in \mathbb{N} \text{ such that } j \geq i \text{ and } \langle\sigma, j\rangle \models A \\
\langle\sigma, i\rangle \models \Box A & \text{iff} & \text{for all } j \in \mathbb{N}, \text{ if } j \geq i \text{ then } \langle\sigma, j\rangle \models A
\end{array}$$

Additionally, the syntax includes two binary future-time temporal operators, interpreted as follows.

$$\begin{array}{lll}
\langle\sigma, i\rangle \models A\mathcal{U}B & \text{iff} & \text{there exists } k \in \mathbb{N}, \text{ such that } k \geq i \text{ and } \langle\sigma, k\rangle \models B \\
 & & \quad \text{and for all } j \in \mathbb{N}, \text{ if } i \leq j < k \text{ then } \langle\sigma, j\rangle \models A \\
\langle\sigma, i\rangle \models A\mathcal{W}B & \text{iff} & \langle\sigma, i\rangle \models A\mathcal{U}B \quad \text{or} \quad \langle\sigma, i\rangle \models \Box A
\end{array}$$

As temporal formulae are interpreted at a particular state-index, i, then indices less than i represent states that are 'in the past' with respect to state s_i. The semantics of the unary past-time operators is given as follows.

$$\begin{array}{lll}
\langle\sigma, i\rangle \models \bullet A & \text{iff} & \langle\sigma,\, i-1\rangle \models A \quad \text{and} \quad i > 0 \\
\langle\sigma, i\rangle \models \blacklozenge A & \text{iff} & \text{there exists } j \in \mathbb{N}, \text{ s.t. } 0 \leq j < i \text{ and } \langle\sigma, j\rangle \models A \\
\langle\sigma, i\rangle \models \blacksquare A & \text{iff} & \text{for all } j \in \mathbb{N}, \text{ if } 0 \leq j < i \text{ then } \langle\sigma, j\rangle \models A
\end{array}$$

Note that, in contrast to the future-time operators, the '$\blacklozenge$' ("sometime in the past") and '$\blacksquare$' ("always in the past") operators are interpreted as being *strict*, i.e., the current index is not included in their definition. Apart from their strictness, the binary past-time operators are similar to their future-time counterparts; their semantics is defined as follows.

$$\begin{array}{lll}
\langle\sigma, i\rangle \models A\mathcal{S}B & \text{iff} & \text{there exists } k \in \mathbb{N}, \text{ s.t. } 0 \leq k < i \text{ and } \langle\sigma, k\rangle \models B \\
 & & \quad \text{and for all } j \in \mathbb{N}, \text{ if } k < j < i \text{ then } \langle\sigma, j\rangle \models A \\
\langle\sigma, i\rangle \models A\mathcal{Z}B & \text{iff} & \langle\sigma, i\rangle \models A\mathcal{S}B \quad \text{or} \quad \langle\sigma, i\rangle \models \blacksquare A\,.
\end{array}$$

Finally, the '**start**' operator is defined such that it can only be satisfied at the beginning of time, i.e. where $i = 0$.

2.3 Separated Normal Form

As an object's behaviour is represented by a temporal formula, we can transform this formula into Separated Normal Form (SNF) (Fisher 1992, Fisher 1997*a*). This not only removes the majority of the temporal operators, but also translates the formula into a set of *rules* suitable for direct execution (see §3). Each of these rules is of one of the following forms.

$$\begin{array}{rcll}
\textbf{start} & \Rightarrow & \bigvee_{j=1}^{r} m_j & \text{(an } \textit{initial } \square\text{-rule)} \\
\bigcirc \bigwedge_{i=1}^{q} k_i & \Rightarrow & \bigvee_{j=1}^{r} m_j & \text{(a } \textit{global } \square\text{-rule)} \\
\textbf{start} & \Rightarrow & \Diamond l & \text{(an } \textit{initial } \Diamond\text{-rule)} \\
\bigcirc \bigwedge_{i=1}^{q} k_i & \Rightarrow & \Diamond l & \text{(a } \textit{global } \Diamond\text{-rule)}
\end{array}$$

where each k_i, m_j or l is a literal. Note that the left-hand side of each *initial* rule is a constraint only on the *first* state, while the left-hand side of each *global* rule represents a constraint upon the previous state. The right-hand side of each $\square$-rule is simply a disjunction of literals referring to the current state, while the right-hand side of each $\Diamond$-rule is a single eventuality (i.e., '$\Diamond$' applied to a literal).

While the details of the transformation process will not be given here, it is important to note that Concurrent METATEM programs are represented as sets of rules (i.e. implications) where the left-hand side of each rule is a past-time formula, while the right-hand side of each rule is a present or future-time formula. This simple form leads naturally on to an operational model for the execution of such rules, as described in the next section.

3 CONCURRENT METATEM

The motivation for the development of Concurrent METATEM (Fisher 1993) has been provided from many areas. Being based upon executable logic, it can be utilised as part of the formal specification and prototyping of reactive systems. In addition, as it uses *temporal*, rather than classical, logic the language provides a high-level programming notation in which the dynamic attributes of individual components can be concisely represented (Barringer, Fisher, Gabbay, Gough & Owens 1995). This, together with its use of a novel model of concurrent computation, ensures that it has a range of applications in distributed and concurrent systems (Fisher 1994).

Concurrent METATEM is an object-based programming language comprising two distinct aspects:

1. the fundamental behaviour of a single object is represented as a temporal formula and animation of this behaviour is achieved through the direct execution of the formula (Fisher 1996);

2. objects are placed within an operational framework providing both asynchronous concurrency and broadcast message-passing.

While these aspects are, to a large extent, independent, the use of *broadcast* communication provides a natural link between them as it represents both a flexible communication model for concurrent objects (Birman 1991) and a natural interpretation of distributed deduction (Fisher 1997*b*). Thus, these features together provide an coherent and consistent programming model within which a variety of reactive systems can be represented and implemented.

3.1 Objects

The basic elements of Concurrent METATEM are objects. These are considered to be encapsulated entities, executing independently, and having complete control over their own internal behaviour. There are two elements to each object: its *interface definition* and its *internal definition.* The definition of which messages an object recognises, together with a definition of the messages that an object may itself produce, is provided by the interface definition for that particular object. The internal definition of each object is provided by a temporal specification.

An object's interface consists of three components, namely a unique *identifier*, which names the object, a set of symbols defining what messages will be accepted by the object (these are called *environment* propositions) and a set of symbols defining messages that the object may send (these are called *component* propositions). For example, the interface definition of a 'car' object might be:

car(go,stop,turn)[fuel,overheat]

Here, car is the identifier that names the object, {go,stop,turn} are the environment propositions, and {fuel,overheat} are the component propositions.

In order to animate the behaviour of an object, we choose to execute its temporal specification directly (Fisher 1996). Execution of a temporal formula corresponds to the construction of a model for that formula and, in order to execute a set of SNF rules representing the behaviour of a Concurrent METATEM object, we utilise the *imperative future* (Barringer, Fisher, Gabbay, Owens & Reynolds 1996) approach. This evaluates the SNF rules at every moment in time, using information about the history of the object in order to constrain future execution.

The operator used to represent the basic temporal indeterminacy within the SNF rules is the *sometime* operator, '$\Diamond$'. When $\Diamond\varphi$ is executed, the system must try to ensure that φ *eventually* becomes true. As such eventualities might not be able to be satisfied immediately, we must keep a record of the unsatisfied eventualities, retrying them as execution proceeds. It should be noted that the use of temporal logic as the basis for the computation rules gives an extra level of expressive power over the corresponding classical logics. In particular, operators such as '$\Diamond$' give the opportunity to specify future-time (temporal) indeterminacy.

As an example of a simple set of rules which form a fragment of an object's description, consider the following.

$$\begin{array}{rcl} \textbf{start} & \Rightarrow & \neg\mathsf{moving} \\ \bullet\mathsf{go} & \Rightarrow & \Diamond\mathsf{moving} \\ \bullet(\mathsf{moving} \wedge \mathsf{go}) & \Rightarrow & \mathsf{overheat} \vee \mathsf{fuel} \end{array}$$

Here, we see that moving is false at the start of execution and, whenever go is satisfied in the last moment in time, a commitment to eventually satisfy moving is made. Similarly, whenever both go and moving are satisfied in the last moment in time, then either overheat or fuel must be satisfied.

3.2 Concurrency and Communication

It is fundamental to our approach that all objects are (potentially) concurrently active. In particular, they may be asynchronously executing. Each object, in executing its temporal formula, independently constructs its own temporal model. Within Concurrent METATEM, a mechanism is provided for communication between separate objects which simply consists of partitioning each object's propositions into those controlled by the object and those controlled by its environment. As above, the former are termed either *component* or *internal* propositions while the latter are termed *environment* propositions. Within the individual object's execution, if a component proposition is satisfied, this has the side-effect of *broadcasting* the value of that proposition to all other objects. If a particular message is received, a corresponding environment proposition is satisfied in the object's execution. If an internal proposition is satisfied, this has no external effect.

To fit in with this logical view of communication, whilst also providing a flexible and powerful message-passing mechanism, *broadcast* message-passing is used to pass information between objects. Here, when an object sends a message it does not send it to a specified *destination*, it merely sends it to its environment where it can be received by *all* other objects. Although broadcast is the basic mechanism, both multicast and point-to-point message-passing can be defined on top of this (Fisher 1994). Finally, the default behaviour for a message is that if it is broadcast, then it will *eventually* be received at all possible receivers. Also note that, by default, the order of messages is not preserved, though such a constraint can be added, if required.

3.3 Applications and Implementation

The combination of executable temporal logic, asynchronous message-passing and broadcast communication provides a powerful and flexible basis for the development of reactive systems. Concurrent METATEM is being utilised in the development of a range of applications in areas from distributed artificial intelligence (Fisher & Wooldridge 1993), agent societies (Fisher & Wooldridge 1995), concurrent theorem-proving (Fisher 1997*b*), and systems simulation (Finger, Fisher & Owens 1993). A survey of some of the potential applications of the language is given in (Fisher 1994).

4 PRINCIPLES OF REFINEMENT

Given that Concurrent METATEM objects can be defined and executed, we now consider the refinement of their representations. In particular, we examine the refinement of an object's internal behaviour (via manipulation of its SNF rules), refinement of an object's interface (thus affecting how it interacts with the environment), and refinement of a single object into a *set* of objects exhibiting equivalent behaviour.

First, we will provide a definition of specifications for Concurrent METATEM objects. Rather than consisting solely of the appropriate temporal rules, an object's specification must also provide the partition of propositions into component, environment and internal sets. Note that, in the following definition, props is a function that extracts all proposition symbols from a given set of SNF rules.

Definition 1 (Specification) *A specification of a Concurrent* METATEM *object is given as a tuple* $\langle R, P_E, P_C, P_I\rangle$, *where*

- R *is the set of SNF rules comprising the object,*
- P_E *is the object's set of* environment *propositions,*
- P_C *is the object's set of* component *propositions, and*
- P_I *is the object's set of* internal *propositions,*

and where both $\mathsf{props}(R) \subseteq P_E \cup P_C \cup P_I$ *and* $P_C \cap P_I = \emptyset$.

We now consider a variety of different classes of refinement, beginning with standard refinement of temporal specifications. The notation we use for refinement of specifications is $S_1 \longrightarrow S_2$, meaning that S_2 is a refinement of S_1.

4.1 Refining an Object's Internal Behaviour

Standard refinement of temporal specifications (Manna & Pnueli 1992) can be applied to specifications of Concurrent METATEM objects. Such refinement can be carried out in the following circumstances:

$\langle R, P_E, P_C, P_I\rangle \longrightarrow \langle R', P_E, P_C, P_I\rangle$ if, and only if,

$$\vdash \left(\Box \bigwedge_{r'\in R'} r'\right) \Rightarrow \left(\Box \bigwedge_{r\in R} r\right)$$

Example 1 This first (simple) example involves the removal of eventualities. The original object defined by

$$\begin{array}{l} \mathsf{ex1(announce)[give,receive]:} \\ \quad \textbf{start} \;\Rightarrow\; \Diamond\mathsf{give} \end{array}$$

can be refined to

$$\begin{array}{l} \mathsf{ex1(announce)[give,receive]:} \\ \quad \textbf{start} \;\Rightarrow\; \mathsf{x} \\ \quad \bullet\mathsf{x} \;\Rightarrow\; \mathsf{give} \end{array}$$

showing that a give message will be produced on the second execution step of object ex1. This follows since

$$\vdash \; (\Box(\textbf{start} \Rightarrow \mathsf{x}) \wedge \Box(\bullet\mathsf{x} \Rightarrow \mathsf{give})) \;\Rightarrow\; \Box(\textbf{start} \Rightarrow \Diamond\mathsf{give})$$

Example 2 Another simple example of the reduction of non-determinism concerns the removal of disjunctions. Here, the object defined by

$$\begin{array}{l} \mathsf{ex2(ins)[outs]:} \\ \quad \bullet\mathsf{p} \;\Rightarrow\; \mathsf{q} \vee \mathsf{r} \end{array}$$

can be refined to

$$\begin{array}{l} \mathsf{ex2(ins)[outs]:} \\ \quad \bullet\mathsf{p} \;\Rightarrow\; \mathsf{q} \end{array}$$

Thus, as in standard formal development, non-determinism within the temporal specification is reduced by such refinement steps.

4.2 Interface Refinement

In addition to the refinement of an object's rule set, as above, we can apply refinement to the object's interface. However, certain restrictions on this are enforced. Thus,

$$\langle R, P_E, P_C, P_I\rangle \longrightarrow \langle R, P'_E, P'_C, P'_I\rangle$$

just as long as the following constraints are observed.

1. Any increase in component, environment or internal sets involves propositions not already present in *any* of those sets, e.g. for environment propositions

$$(P'_E - P_E) \cap (P_E \cup P_C \cup P_I) = \emptyset$$

2. Any decrease in component, environment or internal sets does not involve propositions which appear in the original rule set, e.g., again for environment propositions

$$(P_E - P'_E) \cap \mathsf{props}(R) = \emptyset$$

Example 3 To provide a simple example showing why such restrictions are necessary, we give an *illegal* refinment below which may produce unwelcome behaviour. Thus, if we refine the interface of

$$\begin{array}{l} \mathsf{ex3(ins)[outs,p]:} \\ \quad \mathbf{\bigcirc true} \Rightarrow \Diamond \mathsf{p} \end{array}$$

by removing p from its component set, we produce

$$\begin{array}{l} \mathsf{ex3(ins)[outs]:} \\ \quad \mathbf{\bigcirc true} \Rightarrow \Diamond \mathsf{p} \end{array}$$

However, while p messages are broadcast from the original object infinitely often, no p messages are ever broadcast from the refined object. If any other object is dependent upon these messages, then this can obviously lead to radically different behaviour across the system.

4.3 Object Decomposition

Next, we consider refining a single object into a set of objects that together implement the required behaviour. For example, in Fig. 1, an object A is refined into three objects B1, B2 and B3, which communicate together to provide A's behaviour.

This decomposition can take place as long as

$$\vdash \left(\bigwedge_{i=1}^{n} [\![\langle R^i, P_E^i, P_C^i, P_I^i \rangle]\!] \right) \Rightarrow [\![\langle R, P_E, P_C, P_I \rangle]\!]$$

where $[\![\ ]\!]$ provides the temporal semantics of each object (Pnueli 1981), under appropriate communication constraints (Fisher 1995*b*). Note that this validation also checks that the sets of environment, internal and component predicates are consistent across the distributed system.

Unfortunately, simple (yet non-trivial) examples of this type of transformation are difficult to find. The problem is that, even for relatively small specifications, the temporal formula representing the semantics of an object tends to be large. Lack of space precludes the inclusion of such examples.

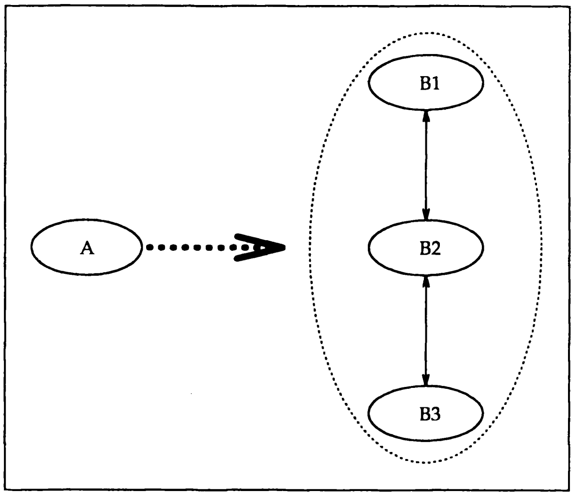

Figure 1 Decomposing an Object

4.4 Fixed Transformations

In an effort to simplify the refinement procedure for the system developer, particularly in the case where an object is decomposed into a new set of objects, we introduce fixed transformations which preserve the behaviour of the systems. While many of the simpler transformations concern the single object refinements described above (in §4.1 and §4.2), the ones we describe here relate to object decomposition (as in §4.3) and have been found to be very useful in simplifying the derivation of refinement.

We label the first transformation, $T1$; the second one, called $T2$ is essentially a generalisation of the first.

$\underline{T1:}$ Given a specification containing the rules

$$\begin{array}{lcl} \bullet\mathsf{p} & \Rightarrow & \mathsf{q} \\ \bullet\mathsf{q} & \Rightarrow & \mathsf{r} \end{array}$$

where r and q do not occur on the left-hand side of any other rules, apart from the above, then the specification of the new object is $\langle\{\bullet\mathsf{q} \Rightarrow \mathsf{r}\}, \{\mathsf{q}\}, \{\mathsf{r}\}, \emptyset\rangle$ where $\bullet\mathsf{q} \Rightarrow \mathsf{r}$ is removed from the original object's ruleset and q is added to its set of component propositions. Note that, if r does not occur in *any* other rule in the original object, this proposition can be removed from its set of component propositions.

$\underline{T2:}$ This is similar to $T1$, except that the r message is recognised by the original object, hence providing a mechanism for passing information between the new and original object. Thus, given a specification containing the rules

$$\begin{array}{lcl} \bullet\mathsf{p} & \Rightarrow & \mathsf{q} \\ \bullet\mathsf{q} & \Rightarrow & \mathsf{r} \\ \bullet\mathsf{r} & \Rightarrow & \mathsf{s} \end{array}$$

where r and q do not occur in any other rules, then the specification of the new object is again $\langle\{\bullet\mathsf{q} \Rightarrow \mathsf{r}\}, \{\mathsf{q}\}, \{\mathsf{r}\}, \emptyset\rangle$. As before, $\bullet\mathsf{q} \Rightarrow \mathsf{r}$ is removed from the original object's ruleset and q is added to its set of component propositions, but now r is added to its set of environment propositions. While this transformation produces the same new object as in $T1$, the difference is that, using $T2$, the r message will effectively be passed back to the original object.

Other transformations follow this pattern. The important property of these transformations is as follows.

Theorem 1 *If a fixed transformation is applied to specification S to give S', then $[\![S]\!] \Leftrightarrow [\![S']\!]$ under the standard communication constraints.*

The standard communication constraints are that if a message is broadcast it will eventually arrive at all other objects. Given this, the above theorem can be established by appealing to the temporal semantics of Concurrent METATEM (Fisher 1995*b*).

Example 4 As a simple example of the use of $T2$, we can transform the object

```
ex4(a)[s,b]:
 ●a  ⇒  p
 ●p  ⇒  q
 ●q  ⇒  r
 ●q  ⇒  b
 ●r  ⇒  s
 ●p  ⇒  s
```

to a system comprising two objects, i.e.

```
ex4(a,r)[s,q]:
 ●a  ⇒  p
 ●p  ⇒  q
 ●r  ⇒  s
 ●p  ⇒  s

new4(q)[r,b]:
 ●q  ⇒  r
 ●q  ⇒  b
```

Thus, the sub-computation concerning the q proposition (message) can be isolated within the newly spawned object.

4.5 Development Example

In this section, we provide a simple example exhibiting the varieties of refinment steps described above, particularly with respect to decomposing an object into a new set of communicating objects. This example, which consists of a very basic planning system, not only has obvious relevance to the domain of multi-agent systems but also shows how specifications incorporating general liveness constraints may be refined. The original object is specified simply by

$$
\begin{array}{rcl}
\multicolumn{3}{l}{\mathsf{planner(goal)[plan]:}} \\
\boldsymbol{\bigcirc}\mathsf{goal} & \Rightarrow & \Diamond\mathsf{plan} \\
\neg\mathsf{subplan1}\;\mathcal{S}\;\mathsf{goal} & \Rightarrow & \neg\mathsf{plan} \\
\neg\mathsf{subplan2}\;\mathcal{S}\;\mathsf{goal} & \Rightarrow & \neg\mathsf{plan} \\
\neg\mathsf{subplan3}\;\mathcal{S}\;\mathsf{goal} & \Rightarrow & \neg\mathsf{plan} \\
\boldsymbol{\bigcirc}\mathsf{goal} & \Rightarrow & \Diamond\mathsf{subplan1} \\
\boldsymbol{\bigcirc}\mathsf{goal} & \Rightarrow & \Diamond\mathsf{subplan2} \\
\boldsymbol{\bigcirc}\mathsf{goal} & \Rightarrow & \Diamond\mathsf{subplan3}
\end{array}
$$

Here, once a goal message is received by the object, it guarantees to eventually produce a plan that achieves that goal. However, the plan can not be produced until all of its three subplans have been completed. Thus, the object also undertakes to generate these subplans.

An obvious structural refinement for this object is to decompose it into four objects, one effectively coordinating the planning activity, the other three producing the three subplans. This refined system can be represented as

$$
\begin{array}{rcl}
\multicolumn{3}{l}{\mathsf{planner(goal,subplan1,subplan2,subplan3)[plan,goal1,goal2,goal3]:}} \\
\boldsymbol{\bigcirc}\mathsf{goal} & \Rightarrow & \Diamond\mathsf{plan} \\
(\neg\mathsf{subplan1} \vee \neg\mathsf{subplan2} \vee \neg\mathsf{subplan3})\;\mathcal{S}\;\mathsf{goal} & \Rightarrow & \neg\mathsf{plan} \\
\boldsymbol{\bigcirc}\mathsf{goal} & \Rightarrow & \mathsf{goal1} \\
\boldsymbol{\bigcirc}\mathsf{goal} & \Rightarrow & \mathsf{goal2} \\
\boldsymbol{\bigcirc}\mathsf{goal} & \Rightarrow & \mathsf{goal3}
\end{array}
$$

$$
\begin{array}{l}
\mathsf{p1(goal1)[subplan1]:} \\
\quad \boldsymbol{\bigcirc}\mathsf{goal1} \Rightarrow \Diamond\mathsf{subplan1}
\end{array}
$$

$$
\begin{array}{l}
\mathsf{p2(goal2)[subplan2]:} \\
\quad \boldsymbol{\bigcirc}\mathsf{goal2} \Rightarrow \Diamond\mathsf{subplan2}
\end{array}
$$

$$
\begin{array}{l}
\mathsf{p3(goal3)[subplan3]:} \\
\quad \boldsymbol{\bigcirc}\mathsf{goal3} \Rightarrow \Diamond\mathsf{subplan3}
\end{array}
$$

To see how the original specification is transformed into this, we will now consider the refinement steps used in a little more detail.

1. First, we can merge the three rules utilising the '$\mathcal{S}$' operator into one, i.e.

$$(\neg\mathsf{subplan1} \vee \neg\mathsf{subplan2} \vee \neg\mathsf{subplan3})\,\mathcal{S}\,\mathsf{goal} \Rightarrow \neg\mathsf{plan}$$

   using standard temporal refinement.
2. Next, we refine each of the '$\bullet\mathsf{goal} \Rightarrow \Diamond\mathsf{subplan}_i$' rules in order to introduce intermediate steps characterised by new variables. Thus, '$\bullet\mathsf{goal} \Rightarrow \Diamond\mathsf{subplan1}$' becomes

$$\begin{array}{rcl}\bullet\mathsf{goal} & \Rightarrow & \mathsf{goal1}\\ \bullet\mathsf{goal1} & \Rightarrow & \Diamond\mathsf{subplan1}\end{array}$$

   again using standard temporal refinement. N.B., goal1 will be used as the coordinating message between the planner and p1.
3. Finally, this specification is then distributed amongst four objects, with the planner object retaining all, and only, those SNF rules that contain the goal proposition. This is achieved by fixed transformations based upon $T2$ being applied to rules of the form $\bullet\mathsf{goal}_i \Rightarrow \mathsf{subplan}_i$ generated in step (2) above.

While this example is relatively simple, similar refinements occur in many systems. Since all temporal specifications are translated into SNF, wherein the main temporal operator is '$\Diamond$', and since the global structures within the application are often provided by grouping, which itself is based upon the decomposition of objects into appropriate sets of objects, it is perhaps not surprising that many applications require transformation steps of this form.

5 CONCLUSIONS AND FUTURE WORK

We have provided a basic framework for refinement in Concurrent METATEM, thus allowing the principled development of a range of concurrent object-based systems. While refinement proofs remain, in general, difficult, the use of fixed transformations provides the system developer with a toolbox of fast (no verification required), simple and safe (behaviour preserving) refinements.

These refinement techniques are powerful, yet relatively simple, primarily because of the fit between the computational model and the temporal execution mechanism. It is important to note that the simplicity of both the refinement conditions and of the fixed transformations derived is a consequence of the easy match between temporal specification, temporal execution and the particular model of computation used. Specifically, utilising objects makes the general refinement of temporal specifications simpler (Barringer, Kuiper & Pnueli 1984), using broadcast messages obviates the need to keep track of senders and receivers of messages, and using a simple communication constraint, such as a message broadcast will eventually arrive at all objects, avoids tight coupling of objects. Although simple, this model of object-based computation can be used to represent applications in a number of areas. In particular, in the multi-agent systems area agents are often quite simply specified, yet it is the patterns of communication and grouping between them that gives these systems power.

While there is much work in the areas of refinement of temporal specifications (Manna & Pnueli 1992), formal methods for object-oriented systems (Buchs & Guelfi 1993), and the development of concurrent object-based systems (Agha 1986), there is little research directly relevant to that presented here. This is mainly because the object model we use is much simpler than most formal models of object-oriented systems, the operational model is unusual compared with the majority of work on concurrent object-based systems, and previous work on refinement of temporal specifications has neither consider the refinement of executable temporal specifications, nor the refinement of specifications under such a model of computation.

There are three directions for future work.

1. The extension of these refinements to first-order temporal logics.
2. The derivation of slightly more complex fixed transformations concerning cases where the objects produced need to communicate together in order to arrive at a common view and where true *grouping* (Birman 1991) is employed (again these are useful for the multi-agent systems area).
3. The development of a toolbox of fixed transformations has led us to consider producing a visual environment for object decomposition. Here, users select the rule(s) they wish to move to new objects, the system checks that this can be achieved and then generates a template for the new objects.

REFERENCES

Agha, G. (1986), *Actors - A Model for Concurrent Computation in Distributed Systems*, MIT Press.

Barringer, H., Fisher, M., Gabbay, D., Gough, G. & Owens, R. (1995), 'METATEM: An Introduction', *Formal Aspects of Computing* 7(5), 533–549.

Barringer, H., Fisher, M., Gabbay, D., Owens, R. & Reynolds, M., eds (1996), *The Imperative Future: Principles of Executable Temporal Logics*, Research Studies Press, Chichester, United Kingdom.

Barringer, H., Kuiper, R. & Pnueli, A. (1984), Now You May Compose Temporal Logic Specifications, *in* 'Proceedings of the Sixteenth ACM Symposium on the Theory of Computing'.

Birman, K. P. (1991), The Process Group Approach to Reliable Distributed Computing, Techanical Report TR91-1216, Department of Computer Science, Cornell University.

Buchs, D. & Guelfi, N. (1993), Formal Development of Actor Programs using Structured Algebraic Petri Nets, *in* 'Parallel Architectures and Languages, Europe (PARLE)', Munich, Germany. (Published in *Lecture Notes in Computer Science*, volume 694, Springer-Verlag).

Emerson, E. A. (1990), Temporal and Modal Logic, *in* J. van Leeuwen, ed., 'Handbook of Theoretical Computer Science', Elsevier, pp. 996–1072.

Finger, M., Fisher, M. & Owens, R. (1993), 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', Gordon and Breach Publishers, Edinburgh, U.K.

Fisher, M. (1992), A Normal Form for First-Order Temporal Formulae, *in* 'Proceedings of Eleventh International Conference on Automated Deduction (CADE)', Saratoga Springs, New York. (Published in *Lecture Notes in Computer Science*, volume 607, Springer-Verlag).

Fisher, M. (1993), Concurrent METATEM — A Language for Modeling Reactive Systems, *in* 'Parallel Architectures and Languages, Europe (PARLE)', Munich, Germany. (Published in *Lecture Notes in Computer Science*, volume 694, Springer-Verlag).

Fisher, M. (1994), A Survey of Concurrent METATEM — The Language and its Applications, *in* 'First International Conference on Temporal Logic (ICTL)', Bonn, Germany. (Published in *Lecture Notes in Computer Science*, volume 827, Springer-Verlag).

Fisher, M. (1995*a*), Representing and Executing Agent-Based Systems, *in* M. Wooldridge & N. R. Jennings, eds, 'Intelligent Agents', Springer-Verlag.

Fisher, M. (1995*b*), Towards a Semantics for Concurrent METATEM, *in* M. Fisher & R. Owens, eds, 'Executable Modal and Temporal Logics', Springer-Verlag.

Fisher, M. (1996), 'An Introduction to Executable Temporal Logics', *Knowledge Engineering Review* **11**(1), 43–56.

Fisher, M. (1997*a*), 'A Normal Form for Temporal Logic and its Application in Theorem-Proving and Execution', *Journal of Logic and Computation* **7**(4).

Fisher, M. (1997*b*), An Open Approach to Concurrent Theorem-Proving, *in* 'Parallel Processing for Artificial Intelligence III', Elsevier Science B.V.

Fisher, M. & Wooldridge, M. (1993), Executable Temporal Logic for Distributed A.I., *in* 'Twelfth International Workshop on Distributed A.I.', Hidden Valley Resort, Pennsylvania.

Fisher, M. & Wooldridge, M. (1995), A Logical Approach to the Representation of Societies of Agents, *in* N. Gilbert & R. Conte, eds, 'Artificial Societies', UCL Press.

Gabbay, D., Pnueli, A., Shelah, S. & Stavi, J. (1980), The Temporal Analysis of Fairness, *in* 'Proceedings of the Seventh ACM Symposium on the Principles of Programming Languages', Las Vegas, Nevada, pp. 163–173.

Lichtenstein, O., Pnueli, A. & Zuck, L. (1985), 'The Glory of the Past', *Lecture Notes in Computer Science* **193**, 196–218.

Manna, Z. & Pnueli, A. (1992), *The Temporal Logic of Reactive and Concurrent Systems: Specification*, Springer-Verlag, New York.

Pnueli, A. (1981), 'The Temporal Semantics of Concurrent Programs', *Theoretical Computer Science* **13**, 45–60.