Concurrent METATEM as a Coordination Language

Adam Kellett and Michael Fisher

Department of Computing, Manchester Metropolitan University
Manchester, United Kingdom, M1 5GD
EMAIL: {A.Kellett,M.Fisher}@doc.mmu.ac.uk

Abstract. In the area of concurrent and reactive system design, the use of temporal logic as a formal notation has become widespread. Concurrent METATEM is a language designed to support such systems by allowing the direct execution of temporal specifications. Programs in this language consist of asynchronous, concurrent objects which communicate via broadcast message passing. Each object executes its own temporal specification representing a required behaviour. In this paper we present work on the development of Concurrent METATEM as a coordination language. By using the temporal specifications as a high-level mechanism whereby properties required of coordinated applications can be concisely defined, we show how Concurrent METATEM can be extended to utilize the functionality of an underlying language.

1 Introduction

In this paper we consider the extension of Concurrent METATEM to act as a coordination language. Our aim is to extend the functionality of the language to provide a consistent link between formal specification using temporal logic and implemented systems. Temporal logic can be seen as classical logic extended with various modalities representing temporal aspects of logical formulae [2]. The propositional and first-order temporal logics we use (called PTL and FTL) are based on a linear, discrete model of time. Thus, time is modeled 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. As formulae are interpreted at particular states in a sequence, operators which refer to both the past and future are required. Examples of such operators and criteria for their satisfaction at a specific moment in time are given below.

\[ \Diamond \varphi \] is satisfied if \( \varphi \) is satisfied sometime in the future.
\[ \Box \varphi \] is satisfied if \( \varphi \) is satisfied always in the future.
\[ \bigcirc \varphi \] is satisfied if \( \varphi \) is satisfied in the next moment.
\[ \bullet \varphi \] is satisfied if in the previous moment in time \( \varphi \) was satisfied.
\( \varphi U \psi \) is satisfied if \( \psi \) is satisfied until a future moment when \( \varphi \) is satisfied.

The past-time operators \( \bullet \) (sometime in the past), \( \Box \) (always in the past), and \( S \) (since) mirror those of the future given above. The operator \textit{start} is introduced to represent the beginning to time.

Concurrent METATEM programs are, in effect, executable specifications. Programs express the properties required of an execution. Using the Imperative Future paradigm [1],
program execution is a forward chaining process which dynamically reflects changes over time in a temporal model. This imperative approach means that an execution not only assesses the validity of events against the properties required, but actively takes steps to ensure they are satisfied. In this paper, we present an overview of Concurrent METATEM and our approach to coordination.

2 Concurrent METATEM

Concurrent METATEM is a programming language for reactive systems [7] that has been shown to be particularly useful in representing and developing multi-agent systems [4]. It is based on the combination of two complementary elements: the direct execution of temporal logic specifications providing the behaviour of an individual object [5]; and a concurrent operational model in which such objects execute asynchronously, communicate via broadcast message-passing, and are organized using a grouping mechanism [3].

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. An interface definition specifies messages an object may receive and produce, while the internal definition of each object is provided by a temporal formula. Execution corresponds to the construction of a model for an object's formula. At each moment in time, the formula is evaluated using information about the history of the object in order to constrain its future execution. As an example, the following forms a fragment of an object's description.

\[
\begin{align*}
\text{start} & \Rightarrow \neg \text{moving} \\
\bullet \text{go} & \Rightarrow \Diamond \text{moving} \\
\bullet (\text{moving} \land \text{go}) & \Rightarrow \text{overheat} \lor \text{fuel}
\end{align*}
\]

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

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 sequence. Within Concurrent METATEM, communication between separate objects consists of a partition of each object's predicates into those controlled by the object and those controlled by its environment. 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.

3 Coordination

Concurrent METATEM as a coordination language provides a formal mechanisms for defining the interaction between independent software modules. As a coordination language, the use of temporal logic, utilizing the declarative mechanisms of rules and constraints, provides a highly expressive formal mechanism for the abstract definition of