Overview of the Semantics of TCOZ

Brendan Mahony
Information Technology Division
Defence Science and Technology Organisation of Australia
Brendan.Mahony@dsto.defence.gov.au

Jin Song Dong
School of Computing,
National University of Singapore,
dongjs@comp.nus.edu.sg

Abstract

Object-Z is an extension to the Z language designed to facilitate specification in an object-oriented style. It is an excellent tool for modelling data and operations, but its object semantics are single threaded, operations are atomic, and object control logic is defined implicitly. This makes it difficult to use Object-Z to capture the behaviour of concurrent real-time reactive systems. On the other hand, Timed CSP is good at modelling real-time concurrent behaviour, but has little support for modelling the state of a complex system. This paper describes the semantics of TCOZ, a language blended from Object-Z and Timed CSP. The semantic model adopted is the infinite timed failures model of Timed CSP, extended to include initial state and update events for modelling operations on internal state.

1 Introduction

> *Convergence: The choice of a formal method or tool is no longer controversial: formal methods are chosen in relation to their purpose and they are increasingly used in effective combination.* (C. A. R. Hoare [10])

A recent research focus in formal methods is the integration of Z/Object-Z and CSP/CCS [4, 6, 9, 16, 17, 19, 12]. This paper gives an overview of the semantics of one such approach, the blending of Object-Z [3] and Timed CSP [14], called Timed Communicating Object-Z (TCOZ) [12]. TCOZ builds on the respective strengths of both notations in order to provide a single elegant notation for modelling both state and process aspects of complex systems. The notion of blending Object-Z with CSP has been suggested independently by Fischer [4] and Smith [16]. TCOZ is novel in that it includes timing primitives, it supports the modelling of true multi-threaded concurrency, it clearly separates the design of control logic from algorithm design, it clearly separates the communications interface of classes from their internal structure, and it integrates the notions of operational refinement and process refinement into a single notion of class refinement.

This paper presents only the essential process semantics aspects of TCOZ, ignoring other aspects of TCOZ semantics. In particular, the Object-Z reference semantics by Griffith [7] are not discussed. In this sense, our presentation is similar to Smith's abstract semantic model for Object-Z [15].

The support of timing primitives in TCOZ is made possible through the adoption of Reed's timed-failures semantics for Timed CSP [14]. The timed-failures semantics models CSP processes in terms of timed event-traces and timed event-failures. This semantic model allows CSP to be extended with time related primitives such as delays, timeouts, and clock-interrupts. In order to support objects with encapsulated state this model is extended to include an initial state and state update events. Object-Z operations are modelled as terminating sequences of timed state-update events. State-update events are similar to *signal* events, as described by Davies [1], in that their occurrence is determined solely by the process, there is no requirement to model state-failures. Since Z allows unbounded nondeterminism, it is also necessary to adopt the infinite trace variant of the timed failures model, as described by Mislove *et al* [13].

The outline of the paper is as follows. Section 2 gives an overview of TCOZ notation. Section 3 presents the semantic toolkit used to describe the semantics of TCOZ. Section 4 presents the TCOZ semantics. Section 5 concludes the paper.

2 Overview of the TCOZ language

A detailed introduction to TCOZ and its Timed CSP and Object-Z features may be found elsewhere [12, 2]. For comparison of TCOZ to other approaches see Mahony and Dong [11] or Fischer [5]. Here we concentrate on the semantic implications of the language design.

2.1 Process/state duality

Object-Z and Timed CSP seem in many ways to complement each other in their capabilities. Object-Z has strong data and algorithm modelling capabilities. The Z mathematical toolkit is extended with object oriented structuring techniques. Timed CSP has strong process control modelling capabilities. The multi-threading and synchronisation primitives of CSP are extended with timing primitives. Moreover both formalisms are already strongly influenced by the other in their areas of weakness. Object-Z supports a number of primitives which have been inspired by CSP notions such as external choice and synchronisation. CSP practitioners tend to make use of notation inspired by the Z mathematical toolkit in the specification of processes with internal state.

Even at the semantic level there is considerable overlap. The standard behavioural semantics of Object-Z classes is as transition systems [15]. The transition system begins in a legal initial state and then evolves through a series of state transitions each effected by one of the class operations. There is a well-known duality between this state transition behavioural model and event based behavioural models [8] which makes it a simple matter to develop complementary semantics for the two languages.

Given these factors it is natural to consider the possibility of blending the two notations into a more complete approach to modelling real-time and/or concurrent systems. Fischer [4] and Smith [16] have independently suggested CSP-style semantics for Object-Z classes in which operation calls become CSP events. Operation names also take on the role of CSP channels, with the values of both the input and output parameters of an operation call being exchanged on the operation channel by an *atomic* communication event. The sequencing of operation events is determined implicitly by the preconditions of the individual events, at each time the object participates in any event which is currently enabled.

TCOZ takes a different approach. TCOZ uses a fine grain of atomicity in modelling operations. Rather than treating operations as atomic events, TCOZ treats them as sequences of abstract state-update events. We say abstract because these state-update events do not appear at the concrete level in TCOZ. The Z operation schema is the only way to describe a state change in TCOZ. In addition, the responsibility of communicating inputs and outputs is removed from the Z schema. Instead, inputs and outputs are passed as explicit communications on CSP-style channels or else, in the default case, between computations in the same control thread. This has the following benefits.

The temporal properties of an operation may be described by considering the elapsed time to the final state-update in a calculation. Since the granularity of the atomic state-updates remains unspecified, there are no problems in considering arbitrary composition, decomposition, and refinement of operations. Under this model it is permissible to combine simple operations with CSP operators, making it possible to represent true multi-threaded computation even at the operation level.

2.2 Abstract syntax

We take the approach of describing TCOZ in terms of an abstract syntax.

The abstract syntax of TCOZ is essentially the abstract syntax of Timed CSP with the addition of operations and state guards, which are essentially Z schema expressions. The syntax and semantics of Z schemas and expressions has been treated adequately by Spivey [18], so we simply represent the abstract syntax of schemas and expressions by the given sets $[ZS, ZE]$.

The abstract syntax of TCOZ process expressions is given in the form of a free type definition, in a style similar to that adopted by Schneider and Davies [14]. The abstract syntax includes two state related primitives novel to Timed CSP. The *op* constructor allows a Z schema to appear in place of a process expression and the guard operator $(\_ \bullet \_)$ blocks a process when the initial state does not satisfy a guard condition expressed as a schema. For example, $[x > 0] \bullet P$ is the process that acts as P if $x > 0$ initially, and halts otherwise.

As much as possible, the names of the various constructors of TCOZ abstract syntax constructors are identical to their concrete syntax keywords. In the following let $[\Sigma]$ represent the set of all possible communications channels.

$$
\begin{aligned}
TZE ::= {} & ref\langle\!\langle NAME\rangle\!\rangle \mid \textsc{Stop} \mid \textsc{Chaos} \mid \textsc{Skip} \mid \textsc{Wait}\langle\!\langle ZE\rangle\!\rangle \mid \\
& op\langle\!\langle ZS\rangle\!\rangle \mid (\_ \bullet \_)\langle\!\langle ZS \times TZE\rangle\!\rangle \mid \\
& (\_ \to \_)\langle\!\langle \Sigma \times TZE\rangle\!\rangle \mid (\_\_ \to \_)\langle\!\langle \Sigma \times ZE \times TZE\rangle\!\rangle \mid \\
& (\_ \sqcap \_)\langle\!\langle TZE \times TZE\rangle\!\rangle \mid (\_ \,\Box\, \_)\langle\!\langle TZE \times TZE\rangle\!\rangle \mid \\
& (\_ \parallel \_)\langle\!\langle TZE \times TZE\rangle\!\rangle \mid (\_|[\_]|\_)\langle\!\langle TZE \times \mathbb{P}\,\Sigma \times TZE\rangle\!\rangle \mid \\
& (\_ \,|||\, \_)\langle\!\langle TZE \times TZE\rangle\!\rangle \mid (\_;\ \_)\langle\!\langle TZE \times TZE\rangle\!\rangle \mid \\
& (\_[\_/\_])\langle\!\langle TZE \times \Sigma \times \Sigma\rangle\!\rangle \mid (\_ \setminus \_)\langle\!\langle TZE \times \mathbb{P}\,\Sigma \mid \\
& (\mu\,\_ \bullet \_)\langle\!\langle NAME \times TZE\rangle\!\rangle
\end{aligned}
$$

An expression which does not contain any references is called a *ground* term.

The body of TCOZ class is essentially a system of simultaneous equations defining (possible recursively) a collection of (stateful) operations and processes. Each equation consists of a name $[NAME]$ and a TCOZ expression. The collection of names referenced in an expression, tze is called the process signature of tze and written $\mathrm{sig}\, tze$. The process signature of an expression may be determined by a straightforward recursive search of the expression structure.

The body of a TCOZ class definition is modelled abstractly as a mapping from names to TCOZ expressions, such that every name in the collective signature of the expressions has a defining equation in the class.

$$
\begin{aligned}
& TZB == \\
& \quad \{\mathcal{C} : NAME \twoheadrightarrow\!\!\!\!\!\!\shortmid\;\; TZE \mid \\
& \qquad \forall\, tze : \mathrm{ran}\,\mathcal{C};\ N : NAME \bullet \\
& \qquad\quad N \in \mathrm{sig}\, tze \Rightarrow N \in \mathrm{dom}\,\mathcal{C}\}
\end{aligned}
$$

In TCOZ, classes fall into two categories, passive and active. Passive classes consist of an initialisation and a class body.

$$TZC_p == [init : ZS;\ \mathcal{C} : TZB]$$

An active TCOZ class definition consists of an initialisation schema, a definition body, and main process expression, which determines the overall behaviour of the class.

$$TZC_a == [init : ZS;\ \mathcal{C} : TZB;\ main : TZE \mid \mathrm{sig}\, main \subseteq \mathrm{dom}\, TZB]$$

2.3 Semantic models

The state of an active TCOZ object is fully encapsulated, the only way to interact with an active object is via its communications interface. Active objects are modelled as pure (non-terminating) CSP processes, using the basic infinite timed failures semantics.

Passive classes are treated essentially as specification libraries. They have no process semantics and all of their internal components (attributes and operations) may be accessed directly. The operations defined in classes (both active and passive) have explicit state. This is modelled by extending the infinite timed failures model with an initial state component and by introducing a special class of events which update system state.

Thus we utilise two semantic models, the basic infinite timed failures models and the novel infinite timed state model.

3 Semantic toolkit

In this section the mathematical models used in describing the TCOZ semantics are developed. The data-related aspects of the language are modelled using state bindings and the process-related aspects of the language are modelled using event traces and refusals.

The notation of the basic Z mathematical toolkit [18] is used to develop the various models, with the exception that an enhanced convention for treating schemas and schema bindings as first class objects is adopted. This builds on the enhancements to the Z toolkit suggested by Valentine [20] and currently under consideration for inclusion in the forthcoming Z Standard. This convention explicitly recognises schema-types, such as $[a, b : \mathbb{N}]$, as sets of variable bindings. Variable bindings are expressed as terms of the form $\langle\!| a \mathrel{\widehat{=}} e_a, b \mathrel{\widehat{=}} e_b |\!\rangle$. The θ operator is explicitly defined so that, for example, $\theta[a, b : \mathbb{N}] = \langle\!| a \mathrel{\widehat{=}} a, b \mathrel{\widehat{=}} b |\!\rangle$. The notions of bindings and tuples are unified, for example the tuple (a, b, c) is identified with the binding $\langle\!| \#1 \mathrel{\widehat{=}} a, \#2 \mathrel{\widehat{=}} b, \#3 \mathrel{\widehat{=}} c |\!\rangle$.

3.1 Schemas and State bindings

The standard paradigm for system specification in Object-Z is *state oriented*. That is to say that system behaviour is described in terms of *operations* performed on the *system state*. An important feature of Object-Z (and Z) is that system state is described in terms of its identified observable quantities or *variables*. An *observation* of a system is then an assignment or *binding* of observed values to the variables of the system. Properties of systems are described by truth valued expressions in the system variables. The *schema* construct is the notational embodiment this approach. It consists of a *declaration part* which describes the variables of the system under consideration and a *predicate part* which is a truth valued expression describing the properties of the system. In this section, the notions of state variables, bindings, and schemas are modelled and the interpretation of schemas as state guards, initialisations, and operations described. These notions have been treated in considerable detail by other authors (for example Spivey [18] has produced a detailed description of the semantics of Z) so the development presented here is deliberately abstract, describing only the facets of Z's state semantics which are of direct relevance to the novel aspects of the semantics of TCOZ.

3.1.1 Variables

Let the abstract set $[\Pi]$ denote the collection of all legal Object-Z state attribute names.

In the definition of operations it is useful to distinguish names representing initial state, final state, inputs, and outputs. This is achieve through the technique of *decoration* At the lexical level decorations are achieved through the use of distinguished characters, but in this paper it is more convenient to consider decorations as abstract operations on variable names. The trivial decoration allows attribute names to appears as (undecorated) variables. There are three non-trivial decoration operations, read *prime*, *query*, and *shriek* respectively, and denoting the final, input, and output variables respectively. There is also a distinguished variable δ used to describe operation timing.

$$\mathsf{VAR} ::= (\_)\langle\!\langle \Pi \rangle\!\rangle \mid (\_')\langle\!\langle \Pi \rangle\!\rangle \mid (\_?)\langle\!\langle \Pi \rangle\!\rangle \mid (\_!)\langle\!\langle \Pi \rangle\!\rangle \mid \delta$$

The notational convenience of writing, V for $(\_)(\!| V |\!)$, V' for $(\_')(\!| V |\!)$, etc., is adopted throughout this paper.

A collection of variables is called a *signature*. It is useful to define a function for stripping the decorations off variables.

$$\mathcal{V} == \{v : \mathsf{VAR};\ u : \Pi \mid v \in \{u, u', u?, u!\}\}$$

Note that, in TCOZ, the final, input, and output decoration conventions are used solely in operation schemas to help determine the initial and final state signatures. In all other contexts, these decorations are disallowed.

The collection of value constants is represented by the abstract set $[\mathsf{VAL}]$. The syntax of Z (and hence of TCOZ) is designed so as to ensure that VAL may always be represented by a proper set (Spivey [18]), built up inductively from the given sets of a Z document.

A state binding is a partial function from variables to values.

$$Bind \mathrel{\widehat{=}} \mathsf{VAR} \nrightarrow \mathsf{VAL}$$

The domain of a binding is called its signature.

3.1.2 Z expressions

The syntax and semantics of Z expressions has been treated by Spivey [18]. The details of the language definition, such as variable scoping, typing etc., are not novel and may safely be abstracted in discussing the semantics of the TCOZ language. In treating Z expressions it is assumed that *ZE* represents some abstract syntax for Z expression terms. The semantics of Z expressions is modelled using a record consisting of a signature and a binding from states to values.

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{Exp}} \\
\hline
\Gamma : \mathbb{P}\,\mathsf{VAR} \\
\nu : \mathsf{STATE} \pfun \mathsf{VAL} \\
\hline
\operatorname{dom} \nu = \{\gamma : \mathsf{STATE} \mid \operatorname{dom} \gamma \supseteq \Gamma\} \\
\hline
\end{array}
$$

The semantic function on Z expressions is represented by $\mathcal{F}_Z [\![\_]\!]$, that is the meaning of the expression ze is $\mathcal{F}_Z [\![ze]\!]$.

3.1.3 Schemas

The schema is the basic modularisation mechanism in the Z predicate language. It consists of a declaration part and a predicate part. As for Z expressions, the details of the syntax and semantics of Z schemas are abstracted. The abstract syntax of schemas is represented by a given set *ZS*. The semantic space for schemas consists of a signature Γ and a collection of states ϕ, each with signature Γ.

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{Schema}} \\
\hline
\Gamma : \mathbb{P}\,\mathsf{VAR} \\
\phi : \mathbb{P}\,\mathit{Bind} \\
\hline
\forall \gamma : \phi \bullet \operatorname{dom} \gamma = \Gamma \\
\hline
\end{array}
$$

The semantic mapping on Z schemas is represented by $\mathcal{F}_Z [\![\_]\!]$.

3.2 Operations

A *state transition* is a pair of Γ states, the *initial state* and the *final* state, and a duration.

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{Act}} \\
\hline
i : \Pi \pfun \mathsf{VAL} \\
f : \Pi \pfun \mathsf{VAL} \\
d : \mathit{Time} \\
\hline
\end{array}
$$

An operation is represented by an initial-state signature Γ_i, a final-state signature Γ_f, and a collection of transitions (known as the *transition relation* of the operation).

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{Opr}} \\
\hline
\Gamma_i : \mathbb{P}\,\Pi \\
\Gamma_f : \mathbb{P}\,\Pi \\
X : \mathbb{P}\,\mathit{Act} \\
\hline
\forall \chi : X \bullet \\
\qquad \operatorname{dom} \chi.i = \Gamma_i \land \\
\qquad \operatorname{dom} \chi.f = \Gamma_f \\
\hline
\end{array}
$$

The *precondition* of the operation is the collection of initial states which are associated with a final state.

$$\mathrm{pre}\ \_ == \lambda\, \mathrm{O} : \mathrm{Opr} \bullet \mathrm{i}(\!|\mathrm{O.X}|\!)$$

An operation is said to be *non-deterministic* when there is some initial state which is related to more than one final state. An operation O_2 is said to be a *refinement* of another operation O_1, written $O_1 \sqsubseteq O_2$, when they have the same signature but O_2 has a weaker precondition and is less non-deterministic than O_1.

$$\begin{array}{l}
(_ \sqsubseteq _) == \\
\quad \{O_1, O_2 : \mathit{Opr} \mid \\
\qquad O_1.\Gamma_i = O_2.\Gamma_i \wedge O_1.\Gamma_f = O_2.\Gamma_f \wedge \mathrm{pre}\ \mathrm{O}_1 \subseteq \mathrm{pre}\ \mathrm{O}_2 \wedge \\
\qquad \forall \chi : O_2.X \bullet \chi.i \in \mathrm{pre}\ \mathrm{O}_1 \Rightarrow \chi \in \mathrm{O}_1.\mathrm{X}\}
\end{array}$$

Operations are expressed using Z schemas, under the following interpretation. In order to describe a TCOZ operation, the timing variable δ must be in the the signature of a schema, the final variables must be a subset of the initial variables and the input variables and output variables must be distinct from the state variables. This is more restrictive than for Object-Z or Z, but is required if the input and output variables are to be used sensibly outside the schema, where decorations are not allowed.

$$\begin{array}{|l}
\textit{Opr_Schema} \\
\hline
\mathit{Schema} \\
\hline
\delta \in \Gamma \\
\mathcal{V}(\!|\Gamma \cap \Pi'|\!) \subseteq (\Gamma \cap \Pi) \\
\mathcal{V}(\!|\Gamma \cap \Pi?|\!) \cap (\Gamma \cap \Pi) = \varnothing \\
\mathcal{V}(\!|\Gamma \cap \Pi!|\!) \cap (\Gamma \cap \Pi) = \varnothing \\
\hline
\end{array}$$

These requirements are enforced at the syntactic level, for example δ and the class state variables are implicitly included in the signature of every operation schema. We may safely define the semantics of operations only on such schemas. The semantic function on operation schemas is *mk_Opr*, which, for each schema of the correct form, determines the corresponding operation.

$$\begin{array}{l}
\mathcal{B} == \lambda\, \gamma : \mathit{Bind} \bullet \{v : \Pi \cap \mathrm{dom}\, \gamma \bullet \mathcal{V}(v) \mapsto \gamma(v)\} \\
\mathcal{B}' == \lambda\, \gamma : \mathit{Bind} \bullet \{v : \Pi' \cap \mathrm{dom}\, \gamma \bullet \mathcal{V}(v) \mapsto \gamma(v)\} \\
\mathcal{B}? == \lambda\, \gamma : \mathit{Bind} \bullet \{v : \Pi? \cap \mathrm{dom}\, \gamma \bullet \mathcal{V}(v) \mapsto \gamma(v)\} \\
\mathcal{B}! == \lambda\, \gamma : \mathit{Bind} \bullet \{v : \Pi! \cap \mathrm{dom}\, \gamma \bullet \mathcal{V}(v) \mapsto \gamma(v)\} \\
\mathit{mk_Opr} == \\
\quad \{\mathit{Opr_Schema};\ \mathit{Opr} \mid \\
\qquad \Gamma_i = \mathcal{V}(\!|\Gamma \cap (\Pi \cup \Pi?)|\!) \wedge \\
\qquad \Gamma_f = \mathcal{V}(\!|\Gamma \cap (\Pi' \cup \Pi!)|\!) \wedge \\
\qquad X = [\mathit{Act} \mid \exists\, \gamma : \phi \bullet i = \mathcal{B}(\gamma) \cup \mathcal{B}?(\gamma) \wedge \\
\qquad \qquad \qquad \qquad \qquad f = \mathcal{B}'(\gamma) \cup \mathcal{B}?(\gamma) \wedge \\
\qquad \qquad \qquad \qquad \qquad d = \gamma(\delta)] \\
\qquad \bullet\ \theta \mathit{Schema} \mapsto \theta \mathit{Opr}\}
\end{array}$$

3.2.1 State Guards

A state guard is essentially a predicate on the state variables. It consists of a signature and a collection of states.

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{Grd}} \\
\hline
\Gamma : \mathbb{P}\,\Pi \\
\phi : \mathbb{P}\,\mathit{Bind} \\
\hline
\forall \gamma : \phi \bullet \mathrm{dom}\,\gamma = \Gamma \\
\hline
\end{array}$$

Again a state guard is expressed using a schema. The semantic function on guards, *mk_Grd*, is a simple injection.

$$\mathit{mk\_Grd} == \{\mathit{Schema};\ \mathit{Grd} \bullet \theta \mathit{Schema} \mapsto \theta \mathit{Grd}\}$$

3.2.2 State initialisation

A state initialisation is also a predicate on state variables.

$$\begin{array}{|l}
\multicolumn{1}{l}{\textsc{Init}} \\
\hline
\Gamma : \mathbb{P}\,\Pi \\
\phi : \mathbb{P}\,\mathit{Bind} \\
\hline
\forall \gamma : \phi \bullet \mathrm{dom}\,\gamma = \Gamma \\
\hline
\end{array}$$

Again the semantic function for state initialisations is a simple injection.

$$\mathit{mk\_}\textsc{Init} == \{\mathit{Schema};\ \textsc{Init} \bullet \theta \mathit{Schema} \mapsto \theta \textsc{Init}\}$$

3.3 Traces and refusals

In this section a toolkit for describing infinite timed failures is developed. The general notational approach of Mislove *et al* [13] is followed fairly closely, though the Z syntactic conventions are exploited fully.

3.3.1 Events

A TCOZ event may be either an update event, a simple synchronisation, a channel communication, or a termination event.

$$\begin{array}{l}
U == [v : \Pi;\ d : \mathsf{VAL}] \\
S == [c : \Sigma] \\
C == [c : \Sigma;\ d : \mathsf{VAL}] \\
\mathit{Event} == \mathit{update}\langle\!\langle U \rangle\!\rangle \mid \mathit{sync}\langle\!\langle S \rangle\!\rangle \mid \mathit{com}\langle\!\langle C \rangle\!\rangle \mid \checkmark
\end{array}$$

Updates record the variable changed and the new value. Synchronisations simply record the channel used, while channel communications also record a data value.

$$\mathbb{U} == update(\!| U |\!)$$
$$\mathbb{S} == syn(\!| S (\!|$$
$$\mathbb{C} == com(\!| C |\!)$$

The update events are distinguished from the other events in that they do not require the cooperation of the environment. The events which do require the cooperation of the environment are called environment events.

$$\mathbb{E} == \mathbb{S} \cup \mathbb{C} \cup \{\checkmark\}$$

Finally we overload the field selectors of the schema types U, S, C to act on events in the obvious way. For example, $update(\langle\!| v \mathrel{\widehat=} \nu, d \mathrel{\widehat=} \alpha |\!\rangle).v = \nu$.

3.3.2 Traces

A timed trace is a (possibly infinite) sequence of time stamped events. The time stamps on the records in a timed trace must be non-decreasing.

$$\begin{array}{l} Stamp == [\tau : Time;\ \sigma : Event] \\ Trace == \{t : \mathrm{seq}_\infty\ Stamp \mid \\ \qquad\qquad \forall\, n, m : \mathrm{dom}\, t \bullet n \leq m \Rightarrow t(n).\tau \leq t(m).\tau\} \end{array}$$

There is no requirement on traces, even infinite traces, that time should progress, merely that it should not go backwards.

A finite trace is a trace with finite elements.

$$Fin\_Trace == \{t : Trace \mid \exists\, n : \mathbb{N} \bullet \mathrm{dom}\, t = 0 \mathrel{..} n\}$$

The *begin* time of a (non-empty) trace is the time stamp of its first element.

$$\mathrm{begin} == \lambda\, t : Trace \bullet (head\ t).\tau$$

The *end* time of a (non-empty) finite trace is the time stamp of its last element.

$$\mathrm{end} == \lambda\, t : Fin\_Trace \bullet (last\ t).\tau$$

Filtering of traces is specialised form of sequence filtering. A projection function on stamped records is specified and the trace is restricted to elements whose projection falls in a given set.

$$\begin{array}{|l} [Y] \\ \hline \_ \upharpoonright\_\,\_ : Trace \times (Stamp \rightarrow\!\!\!\!\!\!+\; Y) \times \mathbb{P}\, Y \rightarrow Trace \\ \hline \forall\, t : Trace;\ \pi : Stamp \rightarrow\!\!\!\!\!\!+\; Y;\ F : \mathbb{P}\, Y \bullet \\ \qquad t \upharpoonright_\pi F = squash\{n : \mathbb{N};\ s : Stamp \mid s \in \pi^{-1}(\!| F |\!)\} \end{array}$$

For example, $t \upharpoonright_\tau [0 \ldots D]$ is the sequence of events occurring up to time D.

A trace may be delayed in time using the delay operator.

$$del == \lambda\, t : Trace;\ \delta : \mathbb{R} \bullet \lambda\, i : \mathrm{dom}(t) \bullet \langle\!| \tau \mathrel{\widehat=} t(i).\tau + \delta, \sigma \mathrel{\widehat=} t(i).\sigma |\!\rangle$$

A trace may be shifted forward in time by composing it with a negative delay and restricting to positive times.

$$(\_ \dot{-} \_) == \lambda\, t : Trace;\ D : Time \bullet del(t, -D) \restriction_\tau Time$$

The final state constructed by a finite trace is the final value assigned to each of the variables updated by the trace.

$$\begin{array}{l} final == \lambda\, t : Fin\_Trace \bullet \\ \quad \{v : \Pi;\ d : \mathsf{VAL} \mid \\ \quad\quad \exists\, n : \mathrm{dom}\, t \bullet t(n).\sigma = update(\theta U) \wedge \\ \quad\quad\quad \forall\, m : \mathrm{dom}\, t \mid m > n \bullet t(m).\sigma.v \neq v\} \end{array}$$

Two traces are *sequenced* by delaying the second until the end of the first, provided the first is finite. If the first is infinite then it is the result.

$$\begin{array}{l} (\_;\ \_) == \\ \quad \lambda\, t_1, t_2 : Trace \bullet \\ \quad\quad \mathbf{if}\ t_1 \in Fin\_Trace\ \mathbf{then}\ t_1 \frown del(t_2, \mathrm{end}\, t_1) \\ \quad\quad \mathbf{else}\ t_1 \end{array}$$

3.3.3 Refusals

Refusals represent the failure of a process to engage in events offered by the environment. We define them in the usual way [14, 13]. The basic timed refusal set or *refusal token* ($RTOK$) consists of a collection of (environment) events refused for some (half-open) interval of time ($RINT$). Update events do not appear in refusals, since they do not require the cooperation of the environment. A *finite* refusal ($RSET$) consists of a finite number of refusal tokens. An infinite refusal ($IRSET$) consists of any collection of refusal tokens which is finite up to any finite time.

$$\begin{array}{l} RINT == \{\alpha, \omega : Time \bullet [\tau : Time \mid \tau \in [\alpha \ldots \omega)]\} \\ RTOK == \{I : RINT \bullet I \times [\sigma : \mathbb{E}]\} \\ RSET == \{R : \mathbb{F}\, RTOK \bullet \bigcup R\} \\ IRSET == \{R : \mathbb{P}\, RTOK \mid (\forall\, D : Time \bullet (\bigcup R \restriction_\tau [0 \ldots D)) \in RSET) \\ \quad \bullet \bigcup R\} \end{array}$$

The filter, begin and end operators are overloaded to act on refusals.

3.4 The infinite timed states model for TCOZ

TCOZ operations and processes are interpreted in the *infinite timed states model.* This model extends the infinite timed failures model for Timed CSP [13], to allow treatment of local state. On the other hand, since the local state of active objects in encapsulated, they are interpreted in the basic infinite timed failures model. The use of an infinite trace model is mandated by the presence of unbounded non-determinism in Z schemas, but it does introduce some complications in the treatment of recursion which we discuss below.

3.4.1 Behaviours

The denotation of each TCOZ process consists of a collection of behaviours. Each such behaviour consists of an initial state, an event trace, and a timed refusal.

$$
\begin{array}{|l}
\hline
\mathcal{B}_{TIS} \\
\iota : \Pi \rightarrow \mathsf{VAL} \\
s : \mathit{Trace} \\
\aleph : \mathit{RSET} \\
\hline
\end{array}
$$

A basic infinite timed failure does not contain an initial state and restricts the trace to environment events.

$$
\begin{array}{|l}
\hline
\mathcal{B}_{TIF} \\
s : \mathit{Trace} \\
\aleph : \mathit{RSET} \\
\hline
s = s \upharpoonright_{\sigma} \mathbb{E} \\
\hline
\end{array}
$$

The filter, begin, and end operators are extended to behaviours in the obvious manner.

The timed infinite failures model consists of a collections of timed infinite failures behaviours which satisfy the five healthiness criteria, $\mathcal{H}$, described by Mislove *et al* [13].

$$
\begin{array}{ll}
\mathcal{M}_{TIF} == \{S : \mathbb{P}\, \mathcal{B}_{TIF} \mid & \\
\quad \mathcal{H}(S) & \text{[Healthiness criteria]} \\
\quad \} &
\end{array}
$$

Processes in the timed infinite state model consist of $\mathcal{B}_{TIS}$ behaviours which satisfy the $\mathcal{M}_{TIF}$ healthiness criteria for each possible initial state.

$$\mathcal{M}_{TIS} == \{S : \mathbb{P}\, \mathcal{B}_{TIS} \mid \forall\, i : \Pi \rightarrow \mathsf{VAL} \bullet \mathcal{H}(\{\mathcal{B}_{TIS} \mid \iota = i \bullet \theta\mathcal{B}_{TIF}\})\}$$

The filter operator is extended to act on $\mathcal{M}_{TIF}$ and $\mathcal{M}_{TIS}$ in the obvious way.

We define refinement in the timed infinite states model to be reverse subset inclusion on processes.

The signature of a timed infinite states process P, written $\operatorname{sig} P$, is the collection of all variables updated by any behaviour of P.

3.4.2 Recursion

In the case of infinite trace models, it is not known how to construct either a complete metric space (the usual Timed CSP recursion model) or a complete partial order (cpo) (the usual untimed CSP recursion model). Instead, Mislove *et al* [13] use Reed's complete metric [14]

$$\mathcal{D}_{TIF}(S, T) == \inf\{t : \mathit{Time} \mid S \upharpoonright_\tau [0 \ldots t] = T \upharpoonright_\tau [0 \ldots t] \bullet 2^{-t}\},$$

only to find fixed points for deterministic processes. The deterministic processes are used to 'dominate' the local cpo induced by process refinement in the infinite timed failures model. This 'domination' is used to lift the metric recursion semantics on the space of deterministic process to a semantics for *time-guarded* recursive equations in the local cpo of the infinite timed failures model. A recursion $\mu P \bullet F(P)$ is time guarded if and only if there is some (uniform) lower bound on the time consumed between recursive calls, that is

$$\begin{array}{l} \exists \delta : \mathit{Time} \bullet \forall P, Q, t : \mathcal{M}_{TIS} \bullet \\ \quad P \upharpoonright_\tau [0 \ldots t] = Q \upharpoonright_\tau [0 \ldots t] \Rightarrow \\ \qquad F(P) \upharpoonright_\tau [0 \ldots t + \delta] = F(Q) \upharpoonright_\tau [0 \ldots t + \delta]. \end{array}$$

Although our model differs from the basic infinite time failures model, these differences do not affect the Mislove *et al* [13] construction. For the sake of brevity we do not give detailed consideration of these matters. Briefly, the construction begins by defining a Reed-metric on the infinite timed states model.

$$\mathcal{D}_{TIS}(S, T) == \inf\{t : \mathit{Time} \mid S \upharpoonright_\tau [0 \ldots t] = T \upharpoonright_\tau [0 \ldots t] \bullet 2^{-t}\},$$

We then observe that processes in the infinite timed state model may be viewed as collections of infinite timed failures processes, indexed by the set of initial states. Similarly, recursions in $\mathcal{M}_{TIS}$ may be viewed as indexed collections of $\mathcal{M}_{TIF}$ recursions. If a stateful recursion is time-guarded, its corresponding stateless recursions must also be time-guarded and therefore have least fixed points in the Mislove order [13]. The fixed-point of the stateful recursion may then be made up of from fixed-points of its corresponding stateless recursions. Thus there is a well-defined operator *fix* which determines the least fixed point (with respect to the refinement order outlined above) of any time-guarded recursion in the infinite timed state model.

In order to ensure time-guardedness, it is sufficient to guard each recursive call by a definite WAIT. This may also be achieved using an operation schema, provided the operation has a minimum execution time. It is conventional in TCOZ to assume some uniform lower bound on all the operations of a given specification document, but this is not enforced by the semantics.

4 The timed infinite states semantics for TCOZ

In this section a semantics is presented for the TCOZ abstract syntax presented in Section 2.2. Since this is an overview paper, we concentrate our attention to those aspects of TCOZ which are novel with respect to Timed CSP semantics. The semantic operator for the infinite timed states model is represented by $\mathcal{F}_{TIS}\llbracket\_\rrbracket$.

4.1 The semantics of ground terms

The first step in developing the semantics is to give a meaning to each of the ground constructors of the TCOZ abstract syntax. Due to the space limitation, we omit some details.

The STOP process deadlocks immediately.

$$\mathcal{F}_{TIS}\llbracket \text{STOP} \rrbracket == \{\mathcal{B}_{TIS} \mid s = \langle\,\rangle \bullet \theta\mathcal{B}_{TIS}\}$$

The CHAOS process diverges immediately, allowing any behaviour.

$$\mathcal{F}_{TIS}\llbracket \text{CHAOS} \rrbracket == \mathcal{B}_{TIS}$$

The SKIP process terminates immediately.

$$\begin{aligned}
&\mathcal{F}_{TIS}\llbracket \text{SKIP} \rrbracket == \\
&\quad \{\mathcal{B}_{TIS} \mid s = \langle\,\rangle \wedge \checkmark \notin \sigma(\!|\aleph|\!) \bullet \theta\mathcal{B}_{TIS}\} \cup \\
&\quad \{\mathcal{B}_{TIS};\ t : \mathit{Time} \mid s = \langle(\!|\tau \mathrel{\widehat{=}} t, \sigma \mathrel{\widehat{=}} \checkmark|\!)\rangle \wedge \checkmark \notin \sigma(\!|\aleph \restriction_\tau [\![0 \dots t]\!]|\!) \\
&\quad \bullet \theta\mathcal{B}_{TIS}\}
\end{aligned}$$

The WAIT process terminates after a set time. For $t : \mathit{Time}$,

$$\begin{aligned}
&\mathcal{F}_{TIS}\llbracket \text{WAIT}\,_ \rrbracket(t) == \\
&\quad \{\mathcal{B}_{TIS} \mid s = \langle\,\rangle \wedge \checkmark \notin \sigma(\!|\aleph \restriction_\tau [\![t \dots \infty)\!]|\!) \bullet \theta\mathcal{B}_{TIS}\} \cup \\
&\quad \{\mathcal{B}_{TIS};\ t' : \mathit{Time} \mid t' \geq t \wedge s = \langle(\!|\tau \mathrel{\widehat{=}} t', \sigma \mathrel{\widehat{=}} \checkmark|\!)\rangle \wedge \\
&\qquad \checkmark \notin \sigma(\!|\aleph \restriction_\tau [\![t \dots t')\!]|\!) \bullet \theta\mathcal{B}_{TIS}\}
\end{aligned}$$

An operation performs a sequence of state updates so as to ensure a desired relationship between initial and final states, then terminates. If the initial state does not satisfy the precondition, the process diverges. For $O : \mathit{Opr}$,

$$\begin{aligned}
&\mathcal{F}_{TIS}\llbracket \mathit{op} \rrbracket(O) == \\
&\quad \{\mathcal{B}_{TIS};\ \mathit{Act} \mid \theta\mathit{Act} \in O.X \wedge \\
&\qquad O.\Gamma_i \lhd \iota \in \mathrm{pre}\ \mathrm{O} \wedge \mathrm{s} = \mathrm{s} \restriction_{\sigma;\,\mathrm{v}} \mathrm{O}.\Gamma_\mathrm{f} \wedge \mathrm{end}\,\aleph \leq \mathrm{d} \wedge \mathrm{end}\,\mathrm{s} \leq \mathrm{d} \\
&\qquad \bullet \theta\mathcal{B}_{TIS}\} \cup \\
&\quad \{\mathcal{B}_{TIS};\ \mathit{Act} \mid \theta\mathit{Act} \in O.X \wedge \\
&\qquad \Gamma_i \lhd \iota \in \mathrm{pre}\ \mathrm{O} \wedge \mathrm{s} = \mathrm{s} \restriction_{\sigma;\,\mathrm{v}} \mathrm{O}.\Gamma_\mathrm{f} \wedge \mathrm{end}\,\aleph > \mathrm{d} \wedge \mathrm{end}\,\mathrm{s} = \mathrm{d} \wedge \\
&\qquad f = \mathit{final}(s) \wedge \checkmark \notin \sigma(\!|\aleph \restriction_\tau [\![d \dots \infty)\!]|\!) \bullet \theta\mathcal{B}_{TIS}\} \cup \\
&\quad \{\mathcal{B}_{TIS};\ s_u : \mathit{Trace};\ t : \mathit{Time};\ \mathit{Act} \mid \theta\mathit{Act} \in O.X \wedge \\
&\qquad \Gamma_i \lhd \iota \in \mathrm{pre}\ \mathrm{O} \wedge \mathrm{f} = \mathrm{final}(\mathrm{s_u}) \wedge \mathrm{s_u} = \mathrm{s_u} \restriction_{\sigma;\,\mathrm{v}} \mathrm{O}.\Gamma_\mathrm{f} \wedge
\end{aligned}$$

$$
\begin{aligned}
&\quad \text{end } s_u = d \wedge t \geq d \wedge \\
&\quad s = s_u \frown \langle (\!| \tau \mathrel{\widehat{=}} t, \sigma \mathrel{\widehat{=}} \checkmark |\!) \rangle \wedge \checkmark \notin \sigma (\!| \aleph \upharpoonright_\tau [d \ldots t) |\!) \\
&\quad \bullet\ \theta \mathcal{B}_{TIS} \} \cup \\
&\{ \mathcal{B}_{TIS} \mid \Gamma_i \lhd \iota \notin \mathrm{pre\ O} \bullet \theta \mathcal{B}_{\mathrm{TIS}} \}
\end{aligned}
$$

Theorem 4.1 *The Z operation $O_1 : Opr$ is refined by $O_2 : Opr$ if and only if $\mathcal{F}_{TIS}[\![op]\!](O_1)$ is refined by $\mathcal{F}_{TIS}[\![op]\!](O_2)$.*

Proof: Assume that $O_1 \sqsubseteq O_2$ and let $b : \mathcal{B}_{TIS}$ be such that $b \in \mathcal{F}_{TIS}[\![op]\!](O_2)$ and let $i == O_2.\Gamma_i \lhd b.\iota$.

If $i \notin \mathrm{pre\ O_2}$, then also $i \notin \mathrm{pre\ O_1}$ and hence $b \in \mathcal{F}_{TIS}[\![op]\!](O_1)$.

If $i \in \mathrm{pre\ O_2} \setminus \mathrm{pre\ O_1}$, then $b \in \mathcal{F}_{TIS}[\![op]\!](O_1)$ trivially.

Now suppose that $i \in \mathrm{pre\ O_1}$.

If $b.s$ is non-terminating and there exists $\chi : O_2.X$ such that $\text{end } b.\aleph \leq \chi.d$, then since $\chi.d$ is also an execution time allowed by O_1, $b \in \mathcal{F}_{TIS}[\![op]\!](O_1)$.

Finally, suppose there exists $\chi : O_2.X$ such that $\text{end } b.\aleph \geq \chi.d$, $\chi.i = O_2.\Gamma_i \lhd \iota$, and $\mathit{finalb.s} = \chi.f$. If $b.s$ is non-terminating and $\checkmark$ is not refused in $b.\aleph$ from time $\chi.d$, then $b \in \mathcal{F}_{TIS}[\![op]\!](O_1)$. If $b.s$ is terminating and $\checkmark$ is not refused in $b.\aleph$ from time $\chi.d$ before it occurs, then again $b \in \mathcal{F}_{TIS}[\![op]\!](O_1)$.

The reverse implication is similar. $\Box$

State guarding causes a process to deadlock if its initial state fails the guard. For $G : Grd$, $P : \mathcal{M}_{TIS}$,

$$
\begin{aligned}
&\mathcal{F}_{TIS}[\![(\_ \bullet \_)]\!](G, P) = \\
&\quad \{ \mathcal{B}_{TIS} \mid G.\Gamma \lhd \iota \in G.\phi \bullet \theta \mathcal{B}_{TIS} \} \cup \\
&\quad \{ \mathcal{B}_{TIS} \mid G.\Gamma \lhd \iota \notin G.\phi \wedge s = \langle\, \rangle \bullet \theta \mathcal{B}_{TIS} \}
\end{aligned}
$$

Event sequencing guards the commencement of a process by a synchronisation or a channel communication. We consider only the channel communication case, since this involves a Z expression term. For $c : \Sigma$, $e : Exp$, $P : \mathcal{M}_{TIS}$,

$$
\begin{aligned}
&\mathcal{F}_{TIS}[\![(\_.\_ \rightarrow \_)]\!](c, e, P) == \\
&\quad \{ \mathcal{B}_{TIS};\ d : \mathsf{VAL} \mid \\
&\qquad d = e.\nu(\iota) \wedge s = \langle\, \rangle \wedge com(\theta Com) \notin \sigma (\!| \aleph |\!) \bullet \theta \mathcal{B}_{TIS} \} \cup \\
&\quad \{ \mathcal{B}_{TIS};\ (\mathcal{B}_{TIS})_P;\ d : \mathsf{VAL};\ t : \mathit{Time} \mid \\
&\qquad \theta (\mathcal{B}_{TIS})_P \in P \wedge \iota = \iota_P \wedge d = e.\nu(\iota) \wedge \\
&\qquad s = \langle (\!| \tau \mathrel{\widehat{=}} t, \sigma \mathrel{\widehat{=}} com(\theta Com) |\!) \rangle;\ s_P \wedge \\
&\qquad com(\theta Com) \notin \sigma (\!| \aleph \upharpoonright_\tau [0 \ldots t) |\!) \wedge \aleph \mathbin{\dot{-}} t = \aleph_P \\
&\qquad \bullet\ \theta \mathcal{B}_{TIS} \}
\end{aligned}
$$

The internal choice operator combines the behaviours of its arguments. For $P, Q : \mathcal{M}_{TIS}$,

$$
\mathcal{F}_{TIS}[\![(\_ \sqcap \_)]\!](P, Q) = P \cup Q
$$

The external choice operator also combines the behaviours of its arguments, but in this case the process chosen is the first to become active.

$$\begin{aligned}
&\mathcal{F}_{TIS}\,[\![(_ \Box _)]\!](P, Q) = \\
&\quad \{\mathcal{B}_{TIS};\ t : \mathit{Time} \mid t = \mathrm{begin}\ s \land \\
&\qquad \theta\mathcal{B}_{TIS} \upharpoonright_{\tau} [0 \ldots t) \in (P.S \cap Q.S) \upharpoonright_{\tau} [0 \ldots t) \\
&\qquad \theta\mathcal{B}_{TIS} \in P.S \cup Q.S \bullet \theta\mathcal{B}_{TIS}\}
\end{aligned}$$

The simple parallel operator forces the behaviours of processes to synchronise on all environment events and all updated events from their shared signature. Update events not in the shared signature are interleaved. For $P, Q : \mathcal{M}_{TIS}$

$$\begin{aligned}
&\mathcal{F}_{TIS}\,[\![(_ \parallel _)]\!](P, Q).S = \\
&\quad \{\mathcal{B}_{TIS};\ (\mathcal{B}_{TIS})_P;\ (\mathcal{B}_{TIS})_Q;\ A, B : \mathbb{P}\,\mathit{Event} \mid \\
&\qquad A = \mathbb{E} \cup \mathit{update}(\![U \mid v \in \mathrm{sig}\,P]\!) \land \\
&\qquad B = \mathbb{E} \cup \mathit{update}(\![U \mid v \in \mathrm{sig}\,Q]\!) \land \\
&\qquad \iota = \iota_P = \iota_Q \land \theta(\mathcal{B}_{TIS})_P \in P \land \theta(\mathcal{B}_{TIS})_Q \in Q \land \\
&\qquad s = s \upharpoonright_{\sigma} (A \cup B) \land s_P = s \upharpoonright_{\sigma} A \land s_Q = s \upharpoonright_{\sigma} B \land \\
&\qquad \aleph \upharpoonright_{\sigma} (A \cup B) == (\aleph_P \upharpoonright_{\sigma} A) \cup (\aleph_Q \upharpoonright_{\sigma} B) \bullet \theta\mathcal{B}_{TIS}\}
\end{aligned}$$

The details of restricted synchronisation and interleaving are unsurprising and we omit them.

Sequential composition passes the final state of the first process as the initial state of the next. For $P, Q : \mathcal{M}_{TIS}$,

$$\begin{aligned}
&\mathcal{F}_{TIS}\,[\![(_;\ _)]\!](P, Q) == \\
&\quad \{\mathcal{B}_{TIS} \mid \theta\mathcal{B}_{TIS} \in P \land \checkmark \notin \sigma(\!\mathrm{ran}\,s\!) \land\bullet\ \theta\mathcal{B}_{TIS}\} \cup \\
&\quad \{\mathcal{B}_{TIS};\ (\mathcal{B}_{TIS})_P;\ (\mathcal{B}_{TIS})_Q;\ t : \mathit{Time} \mid \theta(\mathcal{B}_{TIS})_P \in P \land \\
&\qquad \theta(\mathcal{B}_{TIS})_Q \in Q \land \iota = \iota_P \land \iota_P \oplus \mathit{final}(s_P) = \iota_Q \land \\
&\qquad (\exists\, r : \mathit{Trace} \bullet \\
&\qquad\quad \checkmark \notin \sigma(\!\mathrm{ran}\,r\!) \land \\
&\qquad\quad s_P = r \frown \langle\!|\tau \mathrel{\widehat{=}} t, \sigma \mathrel{\widehat{=}} \checkmark|\!\rangle \land \\
&\qquad\quad s = r;\ s_Q \land \\
&\qquad \aleph_P = \aleph \upharpoonright_{\tau} [0 \ldots t) \cup [\mathit{Stamp} \mid \tau < t \land \sigma = \checkmark] \land \aleph_Q = \aleph \dot{-} t \land \\
&\qquad \bullet\ \theta\mathcal{B}_{TIS}\}
\end{aligned}$$

The renaming operator and hiding operators have unsurprising semantics and are omitted.

4.2 The semantics of passive classes

The semantics of a passive class is modelled as a binding from names to infinite timed state processes, referred to as *process bindings*. Were only ground terms to appear in defining equations, this binding could be determined directly from the ground term semantics. However, if name references are to appear, it is necessary to adopt a fixed point approach to determining the semantics.

The first step in this process is to interpret a TCOZ expression relative to a given binding of its signature. That is, expressions are modelled as mappings from process bindings to processes. Suppose that $tze : TZE$ is a TCOZ expression and that $\beta : NAME \twoheadrightarrow \mathcal{M}_{TIS}$ is a process binding such that $\operatorname{dom} tze \subseteq \operatorname{sig}\beta$. The semantics of tze relative to β, written $\mathcal{F}_{TIS}\,[\![tze]\!]_\beta$ is defined by recursion on the structure of tze.

$$
\begin{aligned}
&\mathcal{F}_{TIS}\,[\![ref(N)]\!]_\beta = \beta(N)\\
&\mathcal{F}_{TIS}\,[\![\text{STOP}]\!]_\beta = \mathcal{F}_{TIS}\,[\![\text{STOP}]\!]\\
&\mathcal{F}_{TIS}\,[\![\text{DIV}]\!]_\beta = \mathcal{F}_{TIS}\,[\![\text{DIV}]\!]\\
&\mathcal{F}_{TIS}\,[\![\text{SKIP}]\!]_\beta = \mathcal{F}_{TIS}\,[\![\text{SKIP}]\!]\\
&\mathcal{F}_{TIS}\,[\![\text{WAIT}\ te]\!]_\beta = \mathcal{F}_{TIS}\,[\![\text{WAIT}\ \_]\!](\mathcal{F}_Z\,[\![te]\!])\\
&\mathcal{F}_{TIS}\,[\![op(OS)]\!]_\beta = \mathcal{F}_{TIS}\,[\![op]\!](mk\_Opr(\mathcal{F}_Z\,[\![OS]\!]))\\
&\mathcal{F}_{TIS}\,[\![GS \bullet P]\!]_\beta = \mathcal{F}_{TIS}\,[\![\_ \bullet \_]\!](mk\_Grd(\mathcal{F}_Z\,[\![GS]\!]), \mathcal{F}_{TIS}\,[\![P]\!]_\beta)\\
&\mathcal{F}_{TIS}\,[\![c \rightarrow P]\!]_\beta = \mathcal{F}_{TIS}\,[\![\_ \rightarrow \_]\!](c, \mathcal{F}_{TIS}\,[\![P]\!]_\beta)\\
&\mathcal{F}_{TIS}\,[\![c.VE \rightarrow P]\!]_\beta = \mathcal{F}_{TIS}\,[\![\_.\_ \rightarrow \_]\!](c, \mathcal{F}_Z\,[\![VE]\!], \mathcal{F}_{TIS}\,[\![P]\!]_\beta)\\
&\mathcal{F}_{TIS}\,[\![P \sqcap Q]\!]_\beta = \mathcal{F}_{TIS}\,[\![\_ \sqcap \_]\!](\mathcal{F}_{TIS}\,[\![P]\!]_\beta, \mathcal{F}_{TIS}\,[\![Q]\!]_\beta)\\
&\mathcal{F}_{TIS}\,[\![P \,\Box\, Q]\!]_\beta = \mathcal{F}_{TIS}\,[\![\_ \,\Box\, \_]\!](\mathcal{F}_{TIS}\,[\![P]\!]_\beta, \mathcal{F}_{TIS}\,[\![Q]\!]_\beta)\\
&\mathcal{F}_{TIS}\,[\![P \parallel Q]\!]_\beta = \mathcal{F}_{TIS}\,[\![\_ \parallel \_]\!](\mathcal{F}_{TIS}\,[\![P]\!]_\beta, \mathcal{F}_{TIS}\,[\![Q]\!]_\beta)\\
&\mathcal{F}_{TIS}\,[\![P \,|[\, C \,]|\, Q]\!]_\beta = \mathcal{F}_{TIS}\,[\![\_ |[\_]| \_]\!](\mathcal{F}_{TIS}\,[\![P]\!]_\beta, C, \mathcal{F}_{TIS}\,[\![Q]\!]_\beta)\\
&\mathcal{F}_{TIS}\,[\![P \,|||\, Q]\!]_\beta = \mathcal{F}_{TIS}\,[\![\_ \,|||\, \_]\!](\mathcal{F}_{TIS}\,[\![P]\!]_\beta, \mathcal{F}_{TIS}\,[\![Q]\!]_\beta)\\
&\mathcal{F}_{TIS}\,[\![P;\ Q]\!]_\beta = \mathcal{F}_{TIS}\,[\![\_;\ \_]\!](\mathcal{F}_{TIS}\,[\![P]\!]_\beta, \mathcal{F}_{TIS}\,[\![Q]\!]_\beta)\\
&\mathcal{F}_{TIS}\,[\![P[c1/c2]]\!]_\beta = \mathcal{F}_{TIS}\,[\![\_[\_/\_]]\!](c1, c2, \mathcal{F}_{TIS}\,[\![P]\!]_\beta)\\
&\mathcal{F}_{TIS}\,[\![P \setminus C]\!]_\beta = \mathcal{F}_{TIS}\,[\![\_ \setminus \_]\!](\mathcal{F}_{TIS}\,[\![P]\!]_\beta, C)\\
&\mathcal{F}_{TIS}\,[\![\mu\, N \bullet P]\!]_\beta = \mathit{fix}(\lambda\, Q : \mathcal{M}_{TIS} \bullet \mathcal{F}_{TIS}\,[\![P]\!]_{\beta \oplus \{N \mapsto Q\}})
\end{aligned}
$$

This binding semantics is used to determine the semantics of a collection of process definitions.

The same technique used to define fixed points for individual processes is used to define fixed points on bindings. The distance metric on process bindings with signature Γ is defined to be the maximum of the distances between respective process values. That is for $\mathcal{C}_1, \mathcal{C}_2 : \Gamma \rightarrow \mathcal{M}_{TIS}$,

$$\mathcal{D}_{TISB}(\mathcal{C}_1, \mathcal{C}_2) == max\{N : \Gamma \mid \mathcal{D}_{TIS}(\mathcal{C}_1(N), \mathcal{C}_1(N))\}.$$

The refinement relation on process bindings is the pointwise lift of the refinement relation on processes.

$$\mathcal{C}_1 \sqsubseteq \mathcal{C}_2 == \forall\, N : \Gamma \bullet \mathcal{C}_1(N) \sqsubseteq \mathcal{C}_2(N)$$

As in the simple process case the metric can be used to provide a 'domination' of the refinement ordering and thus define a fixed point operator on time-guarded recursion operators.

The semantics of a collection of process definitions is then the fixed point of the corresponding mapping from process bindings to process bindings.

$$\mathcal{F}_{TIS}\,[\![\mathcal{C}]\!] = \mathit{fix}(\mathcal{D}_{TISB}, \lambda\,\beta : \mathrm{sig}\,\mathcal{C} \to \mathcal{M}_{TIS} \bullet \lambda\,N : \mathrm{sig}\,\mathcal{C} \bullet \mathcal{F}_{TIS}\,[\![\mathcal{C}(N)]\!]_{\beta})$$

4.3 The semantics of active classes

Before defining the semantics of active classes, we introduce the state initialisation operator. The state initialisation operator converts a stateful process into a pure process by hiding the initial state and the state update events. The stateful process is also restricted to those behaviours with initial states that satisfy the initialisation predicate.

For $I : \textsc{Init}$ and $P : \mathcal{M}_{TIS}$,

$$\begin{array}{l} \mathcal{F}_{TIS}\,[\![\mathit{init}]\!](I, P) == \\ \quad \{\mathcal{B}_{TIF};\ (\mathcal{B}_{TIS})_P \mid \\ \qquad \theta(\mathcal{B}_{TIS})_P \in P \wedge I.\Gamma \lhd \iota_P \in I.\phi \wedge s = s_P \upharpoonright_{\sigma} \mathbb{E} \wedge \aleph = \aleph_P \\ \qquad \bullet\ \theta\mathcal{B}_{TIF}\} \end{array}$$

The semantics of a class is the main process evaluated in the binding defined by the class definitions, with the local state initialised by the class initialisation.

$$\mathcal{F}_{TIF}\,[\![C]\!] = \mathcal{F}_{TIS}\,[\![\mathit{init}(C.\mathit{init})]\!](\mathcal{F}_{TIS}\,[\![C.\mathit{main}]\!]_{\mathcal{F}_{TIS}\,[\![C.\mathcal{C}]\!]})$$

5 Conclusion

This paper presents the novel aspects of a blended state/event process semantic model for TCOZ. Other aspects of the TCOZ semantics, such as the object-reference semantics and data refinement, are not discussed and are the subject of further work.

The process model used by TCOZ consists of sets of tuples consisting of: an initial state; a (possibly infinite) trace (a sequence of time stamped events, including update-events), and a (possibly infinite) refusal (a record what and when events are refused by the process). This represents a conservative extension to the basic infinite timed failures model of Mislove *et al* [13] which allows us to retain the same basic model of recursion. An infinite trace model has been used so as to ensure proper account is taken of the potentially unbounded nondeterminism allowed by Z schemas.

TCOZ differs from other approaches to blending Object-Z with a process algebra in that it does not identify operations with events. Instead a fine-grained collection of (abstract) state-update events is hypothesised. Put simply, operation schemas are modelled by the collection of those sequences of update events that achieve the state change described by the schema. This means that there is no semantic difference between a Z operation schema and a (terminating) CSP process. It therefore makes sense to also identify their syntactic classes. TCOZ allows operations to be defined either using the schema calculus or the process calculus and it allows operation schema to appear as process in CSP expressions.

Finally we note that Theorem 4.1 means that the TCOZ semantics of operations ensures Timed CSP process refinement agrees with Z operation refinement. This means that the operation refinement and structured specification techniques of standard Z are more applicable to TCOZ than to either Object-Z or any blended notation of which we are aware.

References

[1] J. Davies. *Specification and Proof in Real-Time Systems.* PhD thesis, Oxford University Computing Laboratory, Programming Research Group, 1991.

[2] J. S. Dong and B. Mahony. Active Objects in TCOZ. In M. Hinchey, S. Liu, and J. Staples, editors, *the 2nd IEEE International Conference on Formal Engineering Methods (ICFEM'98).* IEEE Computer Society Press, December 1998. (to appear).

[3] R. Duke, G. Rose, and G. Smith. Object-Z: a Specification Language Advocated for the Description of Standards. *Computer Standards and Interfaces*, 17:511–533, 1995.

[4] C. Fischer. CSP-OZ: A combination of Object-Z and CSP. In H. Bowmann and J. Derrick, editors, *Formal Methods for Open Object-Based Distributed Systems (FMOODS '97)*, volume 2, pages 423–438. Chapman & Hall, 1997.

[5] C. Fischer. How to combine Z with a process algebra. In J. P. Bowen, A. Fett, and M. G. Hinchey, editors, *ZUM'98: The Z Formal Specification Notation*, volume 1493 of *Lect. Notes in Comput. Sci.*, pages 5–23. Springer-Verlag, 1998.

[6] A. J. Galloway and W. J. Stoddart. An operational semantics for ZCCS. In M. Hinchey and S. Liu, editors, *the IEEE International Conference on Formal Engineering Methods (ICFEM'97)*, pages 272–282, Hiroshima, Japan, November 1997. IEEE Computer Society Press.

[7] A. Griffiths and G. Rose. A Semantic Foundation for Object Identity in Formal Specification. *Object-Oriented Systems*, 2:195–215, Chapman & Hall 1995.

[8] J. He. Process simulation and refinement. *Formal Aspects of Computing*, 1(3):229–241, 1989.

[9] M. Heisel and C. Suhl. Formal Specification of Safety-critical Software in Z and Real-Time CSP . In *15th International Conference on Computer Safety, Reliability and Security*, 1996.

[10] C. A. R. Hoare. Formal methods observations. Conference overview, FM'99, http://www.cert.fr/fm99/conferen.htm, 1997.

[11] B. Mahony and J. S. Dong. Network Topology and a Case Study in TCOZ. In J. Bowen, A. Fett, and M. Hinchey, editors, *The 11th International Conference of Z Users*, volume 1493 of *Lecture Notes in Computer Science*, pages 308–327, Berlin, Germany, September 1998. Springer-Verlag.

[12] B. P. Mahony and J. S. Dong. Blending Object-Z and Timed CSP: An introduction to TCOZ. In K. Futatsugi, R. Kemmerer, and K. Torii, editors, *The 20th International Conference on Software Engineering (ICSE'98)*, pages 95–104, Kyoto, Japan, April 1998. IEEE Computer Society Press.

[13] M. Mislove, A. Roscoe, and S. Schneider. Fixed Points Without Completeness. *Theoretical Computer Science*, 138:273–314, 1995.

[14] S. Schneider and J. Davies. A brief history of Timed CSP. *Theoretical Computer Science*, 138, 1995.

[15] G. Smith. A fully abstract semantics of classes for Object-Z. *Formal Aspects of Computing*, 7(3):289–313, 1995.

[16] G. Smith. A semantic integration of Object-Z and CSP for the specification of concurrent systems. In *Proceedings of FME'97: Industrial Benefit of Formal Methods*, Graz, Austria, September 1997. Springer-Verlag.

[17] G. Smith and J. Derick. Refinement and verification of concurrent systems specified in Object-Z and CSP. In M. Hinchey and S. Liu, editors, *the IEEE International Conference on Formal Engineering Methods (ICFEM'97)*, pages 293–302, Hiroshima, Japan, November 1997. IEEE Computer Society Press.

[18] J. M. Spivey. *Understanding Z: A specification language and its formal semantics*, volume 3 of *Cambridge Tracts in Theoretical Comput. Sci.* Cambridge University Press, UK, 1988.

[19] K. Taguchi and K. Araki. The State-Based CCS Semantics for Concurrent Z Specification. In M. Hinchey and S. Liu, editors, *the IEEE International Conference on Formal Engineering Methods (ICFEM'97)*, pages 283–292, Hiroshima, Japan, November 1997. IEEE Computer Society Press.

[20] S. H. Valentine. Equal rights for schemas in Z. In J. P. Bowen and M. G. Hinchey, editors, *ZUM'95: The Z Formal Specification Notation*, number 967 in Lecture Notes in Computer Science, pages 203–223, Limerick, Ireland, September 1995. Springer-Verlag.