Formal Aspects of Computing (2002) 13: 142-160

An Introduction to Real-Time Object-Z

Graeme Smith[1] and Ian Hayes[2]

[1]Software Verification Research Centre
[2]School of Computer Science and Electrical Engineering
University of Queensland, Australia

Abstract. This paper presents Real-Time Object-Z: an integration of the object-oriented, state-based specification language Object-Z with the timed trace notation of the timed refinement calculus. This integration provides a method of formally specifying and refining systems involving continuous variables and real-time constraints. The basis of the integration is a mapping of the existing Object-Z history semantics to timed traces.

Keywords: real-time specification; real-time refinement; Object-Z; timed refinement calculus

1. Introduction

Object-Z [Smi00] is an extension of Z [Spi92] to facilitate specification in an object-oriented style. The enhanced structuring provided by object-oriented constructs, such as classes, and techniques, such as inheritance, significantly improve the clarity of large specifications. While Object-Z has found application in a number of areas including telecommunications [RD93], user interfaces [HC98], multi-media presentations [DDtHR95], and metamodelling of programming and description languages [DDR97, Atk95, GKP98, KC99], its utility is limited by its inability to easily specify continuous variables and real-time constraints. Indeed, extensions of Object-Z have been proposed for specifying continuous [Fri95, MD99] and real-time systems [DCZ96, PA98, MD00].

Semantically, classes in Object-Z are represented by sets of *histories* [Smi95]. Each history describes a possible sequence of states an object of the class can pass through together with a corresponding sequence of operations the object undergoes. This has allowed Object-Z to be readily integrated with the process algebra CSP in order to more easily specify concurrent systems [Smi97, Fis97]. Similarly, we contend that it facilitates the integration of Object-Z with a *timed trace* notation in order to specify continuous and real-time systems. Timed trace notations model systems by the way their observable properties change over time. They include the duration calculus [Zho91] and the timed refinement calculus [MH92, Mah92].

In this paper we present Real-Time Object-Z: an integration of Object-Z with a specification notation

Correspondence and offprint requests to: Graeme Smith, Software Verification Research Centre, University of Queensland 4072, Australia. e-mail: smith@svrc.uq.edu.au

based on a simplified subset of the timed refinement calculus [FHMW98]. In Section 2, we introduce the Object-Z notation and discuss its limitations with respect to modelling continuous and real-time systems. In Section 3, we present the specification notation of the timed refinement calculus and show how it can be integrated with Object-Z in Section 4. In Section 5, we present the semantic basis for our integration and, in Section 6, present a definition of refinement before concluding in Section 7.

The paper summarises and extends our previous work [SH99]. In particular, it adds a definition of refinement and two new features to the language. The first allows a discrete output from a class to be declared as a state variable to indicate that it can only change when an operation occurs. The second allows the time intervals during which an operation is occurring to be explicitly specified. The semantics of Real-Time Object-Z has also been reformulated; in particular, to distinguish input, output and local variables of a class. This has been shown to be useful for defining a parallel composition operator for classes [SH00].

2. Object-Z

The main object-oriented construct in Object-Z is the class. A class may be used to define one or more objects of a system, or to specify the interactions between referenced objects of (other) classes. The focus of this paper is on integrating the timed refinement calculus with classes. Using these classes to construct more complex systems is an area of ongoing work [SH00].

A class in Object-Z is represented syntactically by a named box possibly with generic parameters. In this box there may be local type and constant definitions, at most one state schema and associated initial state schema, and zero or more operation schemas. Each operation schema has a Δ-list of state variables which it may change; all other state variables are implicitly unchanged.

As an example, consider the following specification of a simple digital thermometer [MD99]. We let $\mathbb{R}$ be the set of real numbers [OB97], and let $x \in y \pm z$ denote $y - z \leqslant x \leqslant y + z$.

DigitalThermometer

$Screen ::= Temp\langle\langle \mathbb{Z} \rangle\rangle \mid nil$

$approx : \mathbb{R} \rightarrow (Screen \leftrightarrow \mathbb{R})$

$\forall s : Screen;\ t, e : \mathbb{R} \bullet$
$(s, t) \in approx(e) \Leftrightarrow s \neq nil \land Temp^{\sim}(s) * 0.1 \in t \pm e$

$on : \mathbb{B}$
$screen : Screen$

$\neg\ on \Rightarrow screen = nil$

INIT

$\neg\ on$

On

$\Delta(on)$

$\neg\ on \land on'$

Off

$\Delta(on, screen)$

$on \land \neg\ on'$

SetScreen

$\Delta(screen)$
$t? : \mathbb{R}$

$on \land (screen', t?) \in approx(0.06)$

The class has two state variables: *on*, a Boolean variable which is true when the thermometer is switched on, and *screen*, a variable of the local type *Screen* denoting the value displayed on the screen (in tenths of a degree Celsius) or *nil* when the screen is blank. The interpretation of the value on a non-blank screen is provided by the relation $approx(0.06)$ which relates an integer representation with the set of real numbers representing the temperatures it approximates ($Temp^{\sim}$ denotes the inverse of the function *Temp*). The state invariant of the class captures the property that whenever the thermometer is off, the screen is blank.

Initially the thermometer is off and the operations *On* and *Off* enable it to be switched on and off respectively. Once switched on, the screen remains blank until the *SetScreen* operation occurs. This operation models the screen being set to an approximation of the current temperature represented by the input variable $t?$. The operation remains enabled while the thermometer is on and can occur repeatedly.

In an implementation of the above specification, the operation *SetScreen* would necessarily take some time to occur. During this time, we cannot be certain of the value of *screen*: for example, it may be blank or it may display some arbitrary value. We are only certain that, after the operation, it displays the approximation to the input temperature. Therefore, it would be desirable to specify that either the display is meaningful during the operation occurrence or that the duration of the operation is short enough that any arbitrary value of *screen* is not perceived by an observer.

Similarly, we would like to specify that the time between *SetScreen* operations is long enough that the value on the screen can be observed before it changes. If, for example, the temperature was between 3.9 and 4.0, and the screen was rapidly alternating between 3.9 and 4.0, this might be perceived as 9.8 (see Figure 1).

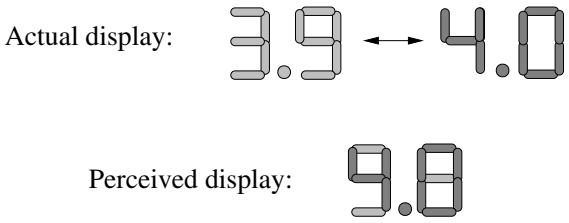

Figure 1: Problem with rapid repetition of *SetScreen*.

At the same time, however, we want the displayed temperature to be within a certain acceptable error of the actual temperature and therefore *SetScreen* operations cannot be too far apart.

All of these conditions are difficult to specify in Object-Z. There is no mechanism for placing constraints on the timing of operations nor for the specification of continuous variables such as the actual temperature. Such conditions are readily captured, however, by a timed trace notation such as that of the timed refinement calculus.

3. Timed refinement calculus

The timed refinement calculus [MH92, Mah92] is a Z-based notation for the specification and refinement of real-time systems. It has been extended with a simple set-theoretic notation for concisely expressing time intervals [Mah94] and operators for accessing interval endpoints. In this section, we present a simplified subset of the notation based on that of Fidge, et al. [FHMW98] which provides a minimal set of operators in addition to those of standard set theory.

Absolute time, $\mathbb{T}$, is modelled by real numbers.

$$\mathbb{T} == \mathbb{R}$$

For the purposes of this paper, we assume the units of $\mathbb{T}$ is seconds.

Observable variables of a system are modelled as total functions from the time domain to a type representing the set of all values the variable may assume. For example, a variable indicating that a digital thermometer is on can be specified as

$$on : \mathbb{T} \rightarrow \mathbb{B}.$$

Similarly, given the definition of *Screen* from Section 2, a variable modelling the screen of the thermometer can be specified as

$screen : \mathbb{T} \rightarrow Screen.$

Functions modelling physical quantities generally map from the time domain to some contiguous subset of the real numbers. In most cases, such functions are *differentiable* (i.e., continuous and smooth). To facilitate specifying this, whenever X is a contiguous set of real numbers, the notation $\mathbb{T} \rightsquigarrow X$ is used to represent the set of all differentiable functions from the time domain to X [FHM98]. For example, the temperature may be declared, using a type $Celsius == \mathbb{R}$, as

$temp : \mathbb{T} \rightsquigarrow Celsius.$

Given this declaration, the derivative of *temp* is denoted by $\underline{d}\ temp$ [FHM98].

A system is specified by constraints on the time intervals over which properties hold. Sets of such intervals can be specified using the interval brackets $\langle\ \rangle$. For example, the set of all intervals where the thermometer is on for the whole interval is specified as

$\langle on \rangle.$

An interval I is in the set of intervals $\langle on \rangle$ if, for all times t in I, $on(t)$ is true.

In general, the property in the brackets is any first-order predicate in which total functions from the time domain to some type X may be treated as values of type X. The elision of explicit references to the time domain of these functions results in specifications which are more concise and readable.

For example, the set of all intervals where the thermometer is on and the screen is not blank for the whole interval is specified as

$\langle on \wedge screen \neq nil \rangle.$

The set of all intervals of duration 1 second can be specified as

$\langle \delta = 1 \rangle.$

The symbol δ is a reserved symbol representing the duration of an interval. Other reserved symbols are α and ω representing the start time and end time of an interval respectively, and ϕ representing the interval itself [FHMW98].

Predicates are formed by combining sets of intervals using operators from set theory such as $\cap$, $\cup$ and $\subseteq$. For example, the property that, when the thermometer is on, the screen is within 0.5 degrees of the actual temperature except for periods less than 0.01 seconds can be specified as follows.

$\langle on \wedge (screen = nil \vee (screen, temp) \notin approx(0.5)) \rangle \subseteq \langle \delta < 0.01 \rangle$

That is, the set of all intervals where the thermometer is on and the screen is not within 0.5 degrees of *temp* is a subset of the set of all intervals of duration less than 0.01 seconds.

More complex properties can also be specified using the interval concatenation operator ';'. This operator forms a set of intervals by joining intervals from one set to those of another whenever their end points meet. (One endpoint must be closed and the other open [FHMW98]). For example, the set of intervals in which the above property is true provided that the thermometer has been on for at least 1 second is specified as

$\langle on \wedge \delta = 1 \rangle\ ;\ \langle on \wedge (screen = nil \vee (screen, temp) \notin approx(0.5)) \rangle \subseteq \langle \delta < 1.01 \rangle.$

Specifications in the timed refinement calculus are constructed from two such predicates. The first predicate is the *assumption* the specification makes about the environment. The second predicate is the *effect* of the specified system under this assumption. A proof obligation exists that the effect does not constrain any variables regarded as inputs to the specified system [Mah92].

4. Combining the notations

In this section, we describe the approach to specifying continuous and real-time systems using Real-Time Object-Z. We also compare our approach with others for specifying continuous and real-time systems using Object-Z.

Classes in Real-Time Object-Z comprise two parts separated by a horizontal line. The part above the line is essentially the standard Object-Z local definitions and schemas. The part below the line contains further constraints on the class specified in the timed refinement calculus notation. As in the timed refinement

calculus, the latter is divided into an assumption and effect part. All state variables $x : X$ in the Object-Z part above the line are interpreted as timed trace variables $x : \mathbb{T} \to X$ in the timed trace part below the line. Furthermore, operation names may appear as variables of type $\mathbb{T} \to \mathbb{B}$ in the timed trace part of the class. The variable representing an operation is true in all intervals in which the operation is occurring. An example of this is given in Section 6.

4.1. Continuous variables

A system specified by a class in Real-Time Object-Z may interact with continuously changing variables in its environment. These variables are specified using the real-time notation for modelling physical quantities. For example, given the definition of *Celsius* from Section 3, the actual temperature of the environment in which a digital thermometer is placed can be specified as follows.

$$| \; temp? : \mathbb{T} \rightsquigarrow Celsius$$

Since this definition gives the values of the temperature over all time, it need not be treated as a modifiable state component and can appear as a local constant in the class. The "?" decoration on the name indicates that it is an environmental variable and, as we will see in the specification of the digital thermometer below, can be used as an "input" to operations. Such operations can be thought of as sampling the environmental variable. Similarly, environmental variables decorated with "!" can be used as "outputs" of operations. When such outputs are discrete, as opposed to continuous, they may be declared as state variables (rather than constants) to indicate that they only change value when an operation, whose Δ-list they appear in, occurs.

Other approaches which support continuous variables in Object-Z [Fri95, MD99] introduce additional notation to distinguish continuous and discrete variables. Our approach, however, is to use standard notation ($\rightsquigarrow$ can be specified in Z [FHM98]) and to be explicit about the types of continuous variables. This makes it more accessible to specifiers already familiar with Object-Z and more amenable to analysis by tools such as Object-Z's type checker *wizard* [Joh96].

4.2. Real-time constraints

All real-time properties of a class could be specified in its timed trace part. However, to make specifying real-time constraints more flexible, and specifications clearer, state variables and local constants of type $\mathbb{T}$ are allowed in the Object-Z part of the class. These variables and constants are generally for specification purposes only and would not be found in an implementation of the class.

In addition, there is an implicit state variable τ of type $\mathbb{T}$ which denotes the current time. This is captured by an implicit constraint $\forall\, t : \mathbb{T} \bullet \tau(t) = t$ in the timed trace part of the class. This constraint formalises the notion that the current time progresses, without the necessity of an explicit *Tick* operation as in the approach of Dong, Colton and Zucconi [DCZ96].

Our approach again has been to use standard Object-Z notation in the Object-Z part of the class (with the exception of τ which is common to all classes) and make real-time constraints explicit. This is in contrast to the TCOZ language [MD98] where additional notation is introduced into both the state schemas and operations of classes. We have also maintained standard Object-Z specification style, in contrast to the approach of Periyasamy and Alagar [PA98] where each object is described by two classes: one specifying its functionality and one specifying its real-time properties.

4.3. Digital thermometer example

As an example of our approach, consider extending the digital thermometer of Section 2 to include the necessary real-time constraints discussed at the end of that section.

DigitalThermometer

$Screen ::= Temp\langle\langle \mathbb{Z} \rangle\rangle \mid nil$

$temp? : \mathbb{T} \rightsquigarrow Celsius$
$approx : \mathbb{R} \rightarrow (Screen \leftrightarrow \mathbb{R})$

$\forall s : Screen;\ t, e : \mathbb{R} \bullet$
$\quad (s, t) \in approx(e) \Leftrightarrow s \neq nil \land Temp^{\sim}(s) * 0.1 \in t \pm e$

$on : \mathbb{B}$
$screen! : Screen$
$last\_set : \mathbb{T}$

$\neg\ on \Rightarrow screen! = nil$

INIT

$\neg\ on$

On

$\Delta(on, last\_set)$

$\neg\ on \land on'$
$last\_set' + 2 < \tau'$

Off

$\Delta(on, screen!)$

$on \land \neg\ on'$

SetScreen

$\Delta(screen!, last\_set)$

$on \land last\_set + 2 < \tau$
$\exists\, t : \mathbb{T} \bullet \tau \leqslant t \leqslant \tau' \land (screen!', temp?(t)) \in approx(0.06)$
$\tau' < \tau + 0.01 \land last\_set' = \tau'$

assumption
$\forall\, t : \mathbb{T} \bullet |(\underline{d}\ temp?)(t)| \leqslant 0.2$

effect
$\langle on \land \delta = 1 \rangle\ ;\ \langle on \land (screen! = nil \lor (screen!, temp?) \notin approx(0.5)) \rangle \subseteq \langle \delta < 1.01 \rangle$

The variable *last_set* is introduced to denote the time when the screen was last set. This allows us to separate the *SetScreen* operations by more than 2 seconds: a reasonable length of time to avoid the problem illustrated in Figure 1. Whenever the thermometer is switched on, *last_set* is set to a time more than 2 seconds before the current time. This enables the *SetScreen* operation to occur as soon as possible after the thermometer is switched on.

The operation *SetScreen* sets the screen to a value of the environmental variable *temp*? between the times τ (denoting the start time of the operation) and τ' (denoting the end time of the operation) which are less than 0.01 seconds apart. The operation effectively uses *temp*? as an input. When used in this way, a proof obligation is required which shows that the operation does not constrain the environmental variable. In this case the proof obligation is satisfied since there is always a value of $screen!'$ such that $(screen!', t) \in approx(0.06)$ for any value t.

An assumption is made in the timed trace part of the class, that the absolute value of the rate of change of *temp*? is less than 0.2 degrees Celsius per second.

The use of *last_set* captures the desired property that the screen is not updated too often. The other desired properties, that *screen*! is not undefined for too long, and that the displayed temperature is within an acceptable error of the actual temperature, are captured by the effect predicate in the timed trace part of

the class. It states that, if the thermometer has been on for 1 second or more, then the screen is only blank or more than 0.5 degrees Celsius from the actual temperature for periods of less than 0.01 seconds.

Given all these constraints, it is worthwhile to prove that our specification is realisable, i.e., that it is not impossible to implement. The effect need only be satisfied when the assumption is true, i.e., when the temperature is changing at a rate of 0.2 degrees Celsius per second or less. We assume this is the case in the following.

The displayed temperature read at some time t may be inaccurate by as much as 0.06 degrees Celsius: 0.01 degrees Celsius due to the sampling error, and 0.05 degrees Celsius due to the rounding performed by the *SetScreen* operation. Therefore, the earliest the displayed temperature may be 0.5 degrees Celsius from the actual temperature is at (0.5-0.06)/0.2=2.2 seconds after t. Hence, to maintain the invariant in the timed trace part of the class, the temperature must be reread and the screen updated in a time less than 2.2+0.01=2.21 seconds after t.

Since the separation between *SetScreen* operations need only be greater than 2 seconds, the specification is realisable (see Figure 2).

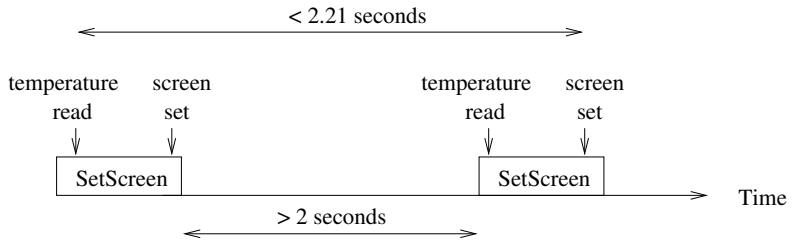

Figure 2: Timing constraints on *SetScreen*.

5. Semantics

To provide a semantics for Real-Time Object-Z, we show how to map the standard Object-Z semantics to timed traces. Smith [Smi95, §2.3] gives a history model for an Object-Z class in terms of sequences of states and operations. We introduce that semantics and then show how to relate it to timed traces.

5.1. Histories

Let *Ident* denote the set of all identifiers, and *Value* the set of all values of any type. A state is an assignment of values to a set of identifiers representing its attributes. It can be defined by a finite partial function from identifiers to values:

$$State == Ident \pfun Value$$

An operation can be defined as an identifier corresponding to the operation's name:

$$Operation == Ident$$

Note that operations in Real-Time Object-Z do not have input and output parameters: all communication is performed through environmental variables such as *temp*? and *screen*! in the example of Section 4. Since environmental inputs cannot be constrained by a class, and these are the only inputs to a class, all information about when operations can be refused by a class are captured by its histories. This enables a straightforward definition of refinement as shown in Section 6.

The history of an object consists of (possibly infinite) sequences of states and operations. The sequence of states is non-empty as there must be at least an initial state. The set of attributes of every state in the sequence comprises the state variables of the object's class[1] and hence must be the same. If the sequence of operations is finite, then the sequence of states has the initial state plus an element corresponding to the

[1] In Smith's model [Smi95], the attributes of states include, as well as state variables, all constants the object's class can refer to. Here we take an alternative view that the values of such constants are parameters to the semantics.

final state of every operation. Hence the sequence of states is one longer than the sequence of operations. If the sequence of operations is infinite, then so is the sequence of states.

These conditions are captured by the following schema where $\text{seq}_\infty X == \text{seq}\, X \cup (\mathbb{N}_1 \to X)$.

History

$$
\begin{array}{l}
states : \text{seq}_\infty State \\
ops : \text{seq}_\infty Operation \\
attributes : \mathbb{F}\, Ident \\
opids : \mathbb{F}\, Operation \\
\hline
states \neq \langle\rangle \\
\forall\, i : \text{dom}\, states \bullet \text{dom}(states\ i) = attributes \\
\text{ran}(ops) \subseteq opids \\
\forall\, i : \mathbb{N}_1 \bullet i \in \text{dom}\, ops \Leftrightarrow i + 1 \in \text{dom}\, states
\end{array}
$$

For example, a typical history of the *DigitalThermometer* class of Section 4, where the *SetScreen* operation samples a temperature of 25.5 degrees Celsius, is described by the following assignment of values to the schema variables.

$$
\begin{array}{l}
states = \langle \{(on, \text{false}), (screen, nil), (last\_set, 0)\}, \\
\qquad\quad \{(on, \text{true}), (screen, nil), (last\_set, 2)\}, \\
\qquad\quad \{(on, \text{true}), (screen, Temp(255)), (last\_set, 5.11)\}, \\
\qquad\quad \{(on, \text{false}), (screen, nil), (last\_set, 5.11)\}\rangle \\
ops = \langle On, SetScreen, Off\rangle \\
attributes = \{on, screen, last\_set\} \\
opids = \{On, SetScreen, Off\}
\end{array}
$$

5.2. Start and finish times

To map histories to timed traces, we extend the standard definition of an Object-Z history given above. The first extension is to allow for the start and finish times of each operation. The variable *start* denotes the sequence of start times of operations, and the variable *finish* denotes a sequence, with indices starting from 0, of finish times. We use *finish*(0) to represent the time at which the initialisation completed, and if the sequence of operations is finite we add an extra start time, with value ∞, representing that after the last operation, the state is stable forever.

TimedHistory

$$
\begin{array}{l}
History \\
start : \text{seq}_\infty\, \mathbb{T} \\
finish : \mathbb{N} \nrightarrow \mathbb{T} \\
\hline
\forall\, i : \mathbb{N} \bullet i \in \text{dom}\, ops \Leftrightarrow \{i, i+1\} \subseteq \text{dom}\, start \\
\text{dom}\, start \neq \mathbb{N}_1 \Rightarrow last(start) = \infty \\
\text{dom}\, finish = \{0\} \cup \text{dom}\, ops \\
\forall\, i : \text{dom}\, ops \bullet start(i) \leq finish(i) \\
\forall\, i : \text{dom}\, finish;\ j : \text{dom}\, start \bullet i < j \Rightarrow finish(i) \leq start(j)
\end{array}
$$

For example, a typical timed history extending the history in Section 5.1 could have the following assignment of values to *start* and *finish*.

$$
\begin{array}{l}
start = \langle 4.205, 5.102, 6.5, \infty\rangle \\
finish = \{(0, 0), (1, 4.215), (2, 5.11), (3, 6.51)\}
\end{array}
$$

5.3. Timed traces

The next extension is to add timed traces of variables. The timed trace of a variable is a mapping from time to the value of the variable at that time.

$$Trace == \mathbb{T} \rightarrow Value$$

We add a timed trace for every environmental variable and each state variable of the class. The names of environmental variables appear in the semantics without their "?" or "!" decorations. This information is captured instead by three sets of identifiers: *inputs* for environmental inputs, *output* for environmental outputs, and *locals* for local state variables. The local state variables are a subset of the attributes, the remainder of the attributes being environmental outputs.

The conditions under which a timed trace corresponds to an Object-Z history is defined below. (The notation $[x \ldots y]$ denotes a closed interval from x to y. The notation $(x \ldots y)$ denotes an open interval.)

TraceHistory

$$\begin{array}{l} TimedHistory \\ inputs, outputs, locals : \mathbb{P}\, Ident \\ trace : Ident \twoheadrightarrow Trace \\ \hline \langle inputs, outputs, locals \rangle \text{ partitions}(\text{dom } trace) \\ locals \subseteq attributes \land attributes \subseteq locals \cup outputs \\ \forall\, i : \text{dom } states;\ id : attributes \bullet \\ \quad (trace\ id)(\!|\ [finish(i-1) \ldots start(i)]\ |\!) = \{(states\ i)(id)\} \end{array}$$

A trace of a state attribute is stable with value $states(i)$ from the finish time of an operation, $finish(i-1)$, until the start time of the next operation, $start(i)$. The initial state, $states(1)$, is stable from the finish time of the initialisation, $finish(0)$, and, if the sequence of states is finite, the final state after the last operation is stable until the start time of the end-of-time event, i.e., infinity. Note that the value of the state trace is only determined for the stable periods between operations; it may be any value during the execution time of an operation.

The example timed history of Section 5.2 is extended as follows.

$$\begin{array}{l} inputs = \{temp\} \\ outputs = \{screen\} \\ locals = \{on, last\_set\} \\ trace = \{(temp, temp\_trace), (screen, screen\_trace), (on, on\_trace), (last\_set, last\_set\_trace)\} \end{array}$$

where

$$temp\_trace \in \{tt : \mathbb{T} \rightsquigarrow Celsius \mid \exists\, temp : tt(\!|\ [5.102 \ldots 5.11]\ |\!) \bullet (Temp(255), temp) \in approx(0.06)\}$$

$$screen\_trace \in \{tt : \mathbb{T} \rightarrow Screen \mid tt(\!|\ [0 \ldots 5.102]\ |\!) = \{nil\} \land tt(\!|\ [5.11 \ldots 6.5]\ |\!) = \{Temp(255)\} \land tt(\!|\ [6.51 \ldots \infty]\ |\!) = \{nil\}\}$$

$$on\_trace \in \{tt : \mathbb{T} \rightarrow \mathbb{B} \mid tt(\!|\ [0 \ldots 4.205]\ |\!) = \{\text{false}\} \land tt(\!|\ [4.215 \ldots 6.5]\ |\!) = \{\text{true}\} \land tt(\!|\ [6.51 \ldots \infty]\ |\!) = \{\text{false}\}\}$$

$$last\_set\_trace \in \{tt : \mathbb{T} \rightarrow \mathbb{T} \mid tt(\!|\ [0 \ldots 4.205]\ |\!) = \{0\} \land tt(\!|\ [4.215 \ldots 5.102]\ |\!) = \{2\} \land tt(\!|\ [5.11 \ldots \infty]\ |\!) = \{5.11\}\}$$

5.4. Operation intervals

We extend the definition further to include the intervals in which particular operations occur. This allows the timed trace part of a class to refer to intervals when a particular operation is occurring.

Intervals are contiguous sets of times:

$$\begin{array}{l} \mathbb{I} : \mathbb{P}(\mathbb{P}\, \mathbb{T}) \\ \hline \mathbb{I} = \{I : \mathbb{P}_1\, \mathbb{T} \mid (inf(I) \ldots sup(I)) \subseteq I\} \end{array}$$

where $inf(I)$ and $sup(I)$ stand for the infimum (greatest lower bound) and supremum (lowest upper bound), respectively, of the set I [FHM98].

For each operation in the Object-Z history, the set of time intervals over which it occurs is just the set of intervals between its start and finish times. From the constraints on start and finish times, no two of these intervals can overlap by more than just a single point of time.

RealTimeHistory

$$\begin{array}{l} \textit{TraceHistory} \\ \textit{occurs} : \textit{Operation} \nrightarrow \mathbb{P}\,\mathbb{I} \\ \hline \textit{occurs} = (\lambda\, \textit{op} : \textit{opids} \bullet \{i : \operatorname{dom} \textit{ops} \mid \textit{ops}(i) = \textit{op} \bullet [\textit{start}(i) \ldots \textit{finish}(i)]\}) \end{array}$$

The function *occurs* is derived from the *ops* sequence.

The example trace history of Section 5.2 is extended as follows.

$$\textit{occurs} = \{(\textit{On}, \{[4.205 \ldots 4.215]\}), (\textit{Off}, \{[6.5 \ldots 6.51]\}), (\textit{SetScreen}, \{[5.102 \ldots 5.11]\})\}$$

5.5. Class histories

An Object-Z class defines a possible set of histories for objects of that class. Smith [Smi95] gives a function $\mathcal{H}$ which given a class returns a set of possible histories of that class. We extend this function to map the Object-Z part of a Real-Time Object-Z class to a set of *real-time histories*, i.e., histories extended as in the previous sections.

$$\mathcal{H} : \textit{Class} \rightarrow \mathbb{P}\, \textit{RealTimeHistory}$$

The details of *Class* and the mapping from a class to a set of histories are the same as those given by Smith [Smi95][2], except that, in operation specifications, references to τ and τ' correspond to the start and finish times, respectively, of the operation. In addition, the semantics need to allow direct references to the environmental variables; such references treat an environmental variable as an explicit trace. The formalisation of these additional relationships within the framework used by Smith [Smi95] is straightforward and we do not give the details here.

5.6. Timed trace predicates

To give the semantics of a Real-Time Object-Z class, we also need to consider the timed trace part of the class. A timed trace predicate defines a set of real-time histories that satisfy the predicate. Let *TimedTracePred* denote the set of all timed trace predicates.

$$\textit{traces} : \textit{TimedTracePred} \rightarrow \mathbb{P}\, \textit{RealTimeHistory}$$

A real-time history, h, satisfies a timed trace predicate if the predicate is true when we replace any reference to a trace variable, v, by the corresponding value, $h.\textit{trace}(v)$. To allow operations in such predicates to represent Boolean values which are true in the sets of intervals in which they occur, we also need to replace any reference to an operation, *op*, by true in intervals in the set, $h.\textit{occurs}(\textit{op})$, and false elsewhere.

An interval expression, such as $\langle P \rangle$, where P is a predicate, is interpreted as the set of intervals such that P holds at all points in the interval:

$$\{\phi : \mathbb{I} \mid \exists\, \alpha, \omega, \delta : \mathbb{T} \bullet \alpha = \textit{inf}(\phi) \land \omega = \textit{sup}(\phi) \land \delta = \omega - \alpha \land \forall\, \tau : \phi \bullet P[\vec{v}(\tau)/\vec{v}]\}$$

where $\vec{v}$ stands for the vector of all timed trace variables, i.e., $\operatorname{dom} \textit{trace}$.

A Real-Time Object-Z class consists of the standard class components (augmented with environmental variables) and two real-time predicates specifying respectively the assumptions the class makes about environmental variables and the effect the class is to achieve on environmental variables.

[2] The type we refer to as *Class* is called *ClassStruct* by Smith.

RealTimeClass

$$class : Class$$
$$assumption, effect : TimedTracePred$$

The possible real-time histories of such a class C consist of those histories that, if the assumption holds, also satisfy the effect and are real-time histories of the corresponding Object-Z part of the class.

$$\mathcal{R} : RealTimeClass \rightarrow \mathbb{P}\, RealTimeHistory$$
$$\mathcal{R}(C) = \{h : RealTimeHistory \mid h \in traces(C.assumption) \Rightarrow h \in traces(C.effect) \cap \mathcal{H}(C.class)\}$$

6. Refinement

Refinement in Real-Time Object-Z can be performed by refining the Object-Z and timed trace parts of the class separately according to the rules of refinement of their respective notations. No new rules need to be developed.

For Object-Z, refinement is achieved by strengthening the initial condition and/or the postconditions of operations. Preconditions of operations cannot be weakened, as in Z refinement [WD96], due to Object-Z's *blocking* interpretation of operations [Smi00]. Under this interpretation operations cannot occur when their preconditions are not satisfied. In Z they can occur when their preconditions are not satisfied resulting in an undefined post-state.

Refinement in the timed refinement calculus consists of weakening of the assumptions and/or strengthening of the effects of specifications [Mah92]. That is, given specifications S_1 with assumption A_1, effect E_1 and output variables $\vec{x}$, and S_2 with assumption A_2, effect E_2 and output variables $\vec{x}$, S_1 is refined by S_2, denoted $S_1 \sqsubseteq S_2$, if, and only if, $A_1 \Rightarrow A_2$ (i.e, A_2 is at least as weak as A_1) and $A_1 \Rightarrow (\forall \vec{x} \bullet E_2 \Rightarrow E_1)$ (i.e., when A_1 is true, E_2 is at least as strong as E_1).

In Real-Time Object-Z, this strengthening and weakening of the timed trace predicates is performed in the context of the class's operation definitions. This is necessary so that Boolean variables representing operations in the timed trace predicates can be related to the environmental and local variables the operations access and modify.

Refining the Object-Z part of a class C restricts the possible post-states of operation occurrences and hence the possible real-time histories of the class, $\mathcal{H}(C.class)$. Similarly, refining the timed trace part of a class C restricts the real-time histories of the effect, $traces(C.effect)$, and/or increases the real-time histories of the assumption, $traces(C.assumption)$. The overall effect, therefore, is to restrict the real-time histories of class C given by $\mathcal{R}(C)$. Hence, a class, C, is refined by a class, D, if the histories of C contain the histories of D.

$$\_ \sqsubseteq \_ : RealTimeClass \leftrightarrow RealTimeClass$$
$$C \sqsubseteq D \Leftrightarrow \mathcal{R}(D) \subseteq \mathcal{R}(C)$$

As an example of refinement in Real-Time Object-Z, consider refining the digital thermometer of Section 4 as follows.

1. To allow at least 0.005 seconds for setting the screen after the temperature has been read, strengthen the postcondition of the *SetScreen* operation so that the temperature is read at least 0.005 seconds before the end of the operation.
2. Remove any assumptions about the external temperature.
3. Replace the desired effect with a constraint that the greatest separation of *SetScreen* operations is 2.1 seconds. Since *SetScreen* occurs in less than 0.01 seconds, this meets the original effect when the original assumption holds as was shown in Section 4.

The resulting class is specified below.

DigitalThermometer

$Screen ::= Temp\langle\langle \mathbb{Z} \rangle\rangle \mid nil$

$temp? : \mathbb{T} \rightsquigarrow Celsius$
$approx : \mathbb{R} \rightarrow (Screen \leftrightarrow \mathbb{R})$

$\forall s : Screen;\ t, e : \mathbb{R} \bullet$
$(s, t) \in approx(e) \Leftrightarrow s \neq nil \land Temp^{\sim}(s) * 0.1 \in t \pm e$

$on : \mathbb{B}$
$screen! : Screen$
$last\_set : \mathbb{T}$

$\neg\ on \Rightarrow screen! = nil$

INIT

$\neg\ on$

On

$\Delta(on, last\_set)$

$\neg\ on \land on'$
$last\_set' + 2 < \tau'$

Off

$\Delta(on, screen!)$

$on \land \neg\ on'$

SetScreen

$\Delta(screen!, last\_set)$

$on \land last\_set + 2 < \tau$
$\exists\, t : \mathbb{T} \bullet \tau \leqslant t \leqslant \tau' - 0.005 \land (screen!', temp?(t)) \in approx(0.06)$
$\tau' < \tau + 0.01 \land last\_set' = \tau'$

assumption
true

effect
$\langle on \land last\_set + 2 < \tau \land \delta > 0.1 \rangle \subseteq \langle true \rangle\ ;\ \langle SetScreen \rangle\ ;\ \langle true \rangle$

Although we can use the existing refinement rules of Object-Z and the timed refinement calculus to refine Real-Time Object-Z classes, these rules are not complete. That is, refinements of a class exist which cannot be reached using the existing rules. Such refinements would move timing and other information between the Object-Z and timed trace parts of the class. For example, the precondition of *SetScreen* could be weakened (not allowed in Object-Z refinement) to let the operation occur at any time when the thermometer was on. Provided the constraint on the minimum separation of *SetScreen* operations was captured by an additional effect predicate, the resulting class would be a refinement.

It is worth noting that it is possible to refine a *feasible* specification, i.e., one that can be implemented, to an infeasible one. This can occur if the Object-Z part of the class and the effect predicate are refined so that their traces no longer intersect. An example of this would be if we strengthened the effect of the thermometer class to state that *SetScreen* operations must have a duration of more than 0.01 seconds by adding the following predicate.

$\langle \neg\ SetScreen \rangle\ ;\ \langle SetScreen \rangle\ ;\ \langle \neg\ SetScreen \rangle \subseteq \langle \delta > 0.01 \rangle$

In the timed refinement calculus, a specification with assumption A, effect E and output variables $\vec{x}$ is feasible if whenever the assumption is true, there exists values of the output variables satisfying the effect,

i.e, $A \Rightarrow (\exists \vec{x} \bullet E)$. Whenever a refinement is performed on a feasible specification, there is a proof obligation that can be used to show that the refined specification is feasible. A similar proof obligation is required for refinement in Real-Time Object-Z. That is, when refining a feasible Real-Time Object-Z class, we need to show that the result is feasible. A Real-Time Object-Z class is feasible if whenever the assumption is satisfied by a real-time history, there exists a real-time history with the same traces for input variables of the class, which satisfies the effect and Object-Z part of the class.

$$\forall h : traces(C.assumption) \bullet (\exists h' : traces(C.effect) \cap \mathcal{H}(C) \bullet h'.inputs \lhd h' = h'.inputs \lhd h)$$

7. Conclusion

In this paper, we have shown how Object-Z can be integrated with the specification notation of the timed refinement calculus in order to formally specify and refine systems involving continuous variables and real-time constraints. Our approach has been to separate the two notations within a class, using essentially standard Object-Z in one part of the class and the timed refinement calculus in the other. This makes our notation, Real-Time Object-Z, more accessible to specifiers who already know Object-Z and more amenable to existing tool support than other approaches which introduce additional notation.

The integration of Object-Z and the specification notation of the timed refinement calculus presented in this paper has focussed solely on classes. Classes, however, are only the building blocks of more complex systems. In Object-Z, such systems are specified using instances of classes called objects. For example, an object of the digital thermometer class may be declared as a state variable in another class by the declaration $dt : DigitalThermometer$. This approach is also valid for Real-Time Object-Z. An alternative way of constructing systems from classes is by introducing a parallel composition operator similar to those found in process algebras. Both approaches are being investigated by the authors [SH00].

Acknowledgements

The authors would like to thank Colin Fidge for his useful comments. This work is funded by Australian Research Council Large Grant A49801500, *A Unified Formalism for Concurrent Real-Time Software Development*.

References

[Atk95] S. Atkinson. Formalizing the Eiffel library standard. In *Technology of Object-Oriented Languages and Systems: TOOLS 18*. Prentice Hall, 1995.

[DCZ96] J.S. Dong, J. Colton, and L. Zucconi. A formal object approach to real-time specification. In *3rd Asia-Pacific Software Engineering Conference (APSEC'96)*. IEEE Computer Society Press, 1996.

[DDR97] J.S. Dong, R. Duke, and G. Rose. An object-oriented denotational semantics of a (block-structured sequential) programming language. *Object-Oriented Systems*, 4(1):29–52, 1997.

[DDtHR95] D.A. Duce, D.J. Duke, P.J.W. ten Hagen, and G.J. Reynolds. Formal methods in the development of PREMO. *Computer Standards and Interfaces*, 17:491–509, 1995.

[FHM98] C.J. Fidge, I.J. Hayes, and B.P. Mahony. Defining differentiation and integration in Z. In J. Staples, M.G. Hinchey, and Shaoying Liu, editors, *IEEE International Conference on Formal Engineering Methods (ICFEM '98)*, pages 64–73. IEEE Computer Society Press, 1998.

[FHMW98] C.J. Fidge, I.J. Hayes, A.P. Martin, and A.K. Wabenhorst. A set-theoretic model for real-time specification and reasoning. In J. Jeuring, editor, *Mathematics of Program Construction (MPC'98)*, volume 1422 of *Lecture Notes in Computer Science*, pages 188–206. Springer-Verlag, 1998.

[Fis97] C. Fischer. CSP-OZ - a combination of CSP and Object-Z. In H. Bowman and J. Derrick, editors, *Formal Methods for Open Object-Based Distributed Systems (FMOODS'97)*, pages 423–438. Chapman & Hall, 1997.

[Fri95] V. Friesen. An exercise in hybrid system specification using an extension of Z. In A. Bouajjani and O. Maler, editors, *Proceedings Second European Workshop on Real-Time and Hybrid Systems*, pages 311–316. VERIMAG Laboratory, Grenoble, 1995.

[GKP98] R. Geisler, M. Klar, and C. Pons. Dimensions and dichotomy in metamodeling. In D.J. Duke and A.S. Evans, editors, *3rd BCS-FACS Northern Formal Methods Workshop*, Electronic Workshops in Computing. Springer-Verlag, 1998.

[HC98] A. Hussey and D. Carrington. Using Object-Z to specify a web browser interface. In P. Palanque and F. Paterno, editors, *Formal Methods in Human-Computer Interaction*, Formal Approaches to Computing and Information Technology, chapter 8. Springer Verlag, 1998.

[Joh96] W. Johnston. A type checker for Object-Z. Technical Report 96-24, Software Verification Research Centre, University of Queensland, 1996.

[KC99] S.-K. Kim and D. Carrington. Formalising the UML class diagram using Object-Z. In *2nd International Conference on Unified Modelling Language (UML'99)*, volume 1732 of *Lecture Notes in Computer Science*. Springer-Verlag, 1999.

[Mah92] B.P. Mahony. *The Specification and Refinement of Timed Processes*. PhD thesis, Department of Computer Science, University of Queensland, 1992.

[Mah94] B.P. Mahony. Using the refinement calculus for dataflow processes. Technical Report 94-32, Software Verification Research Centre, University of Queensland, October 1994.

[MD98] B.P. Mahony and J.S. Dong. Blending Object-Z and Timed CSP: An introduction to TCOZ. In *20th International Conference on Software Engineering (ICSE'98)*, pages 95–104. IEEE Computer Society Press, 1998.

[MD99] B.P. Mahony and J.S. Dong. Sensors and actuators in TCOZ. In J. Wing, J.C.P. Woodcock, and J. Davies, editors, *World Congress on Formal Methods (FM'99)*, volume 1709 of *Lecture Notes in Computer Science*, pages 1166–1185. Springer-Verlag, 1999.

[MD00] B.P. Mahony and J.S. Dong. Timed Communicating Object Z. *IEEE Transactions on Software Engineering*, 26(2):150–177, 2000.

[MH92] B.P. Mahony and I.J. Hayes. A case-study in timed refinement: A mine pump. *IEEE Transactions on Software Engineering*, 18(9):817–826, September 1992.

[OB97] W.R. Oliveira and R.S.M. Barros. The real numbers in Z. In D.J. Duke and A.S. Evans, editors, *2nd BCS-FACS Northern Formal Methods Workshop*, Electronic Workshops in Computing. Springer-Verlag, 1997.

[PA98] K. Periyasamy and V.S. Alagar. Extending Object-Z for specifying real-time systems. In *TOOLS USA'97: Technology of Object-Oriented Languages and Systems*, pages 163–175. IEEE Computer Society Press, 1998.

[RD93] G. Rose and R. Duke. An Object-Z specification of a mobile phone system. In K. Lano and H. Haughton, editors, *Object-Oriented Specification Case Studies*, Object-Oriented Series. Prentice Hall, 1993.

[SH99] G. Smith and I.J. Hayes. Towards real-time Object-Z. In K. Araki, A. Galloway, and K. Taguchi, editors, *1st International Conference on Integrated Formal Methods (IFM'99)*, pages 49–65. Springer-Verlag, 1999.

[SH00] G. Smith and I.J. Hayes. Structuring Real-Time Object-Z specifications. In W. Grieskamp, T. Santen, and B. Stoddart, editors, *2nd International Conference on Integrated Formal Methods (IFM'00)*, volume 1945 of *Lecture Notes in Computer Science*, pages 97–115. Springer-Verlag, 2000.

[Smi95] G. Smith. A fully abstract semantics of classes for Object-Z. *Formal Aspects of Computing*, 7(3):289–313, 1995.

[Smi97] G. Smith. A semantic integration of Object-Z and CSP for the specification of concurrent systems. In J. Fitzgerald, C.B. Jones, and P. Lucas, editors, *Formal Methods Europe (FME'97)*, volume 1313 of *Lecture Notes in Computer Science*, pages 62–81. Springer-Verlag, 1997.

[Smi00] G. Smith. *The Object-Z Specification Language*. Advances in Formal Methods. Kluwer Academic Publishers, 2000.

[Spi92] J.M. Spivey. *The Z Notation: A Reference Manual*. Prentice Hall, 2nd edition, 1992.

[WD96] J.C.P. Woodcock and J. Davies. *Using Z: Specification, Refinement, and Proof*. Prentice Hall, 1996.

[Zho91] Zhou Chaochen, C.A.R. Hoare and A.P. Ravn. A calculus of durations. *Information Processing Letters*, 40:269–271, December 1991.