μ-charts and Z: examples and extensions

Greg Reeve and Steve Reeves
Department of Computer Science
University of Waikato
Hamilton, NEW ZEALAND
{gregr,stever}@waikato.ac.nz

Abstract

μ-Charts are a way of specifying reactive systems, i.e. *systems which are in some environment to which they have to react, based on the well-established formalism Statecharts.*

This paper gives (very abbreviated) examples of translating μ-charts to Z, which is itself a well-established language for specifying computational systems with tried and tested methods and support tools which guide its effective use in systems development. We undertake this translation in order that investigation of the modelled system can be performed before expensive and lengthy implementation is considered.

We also present an extension of the μ-charts and the related Z to deal with a simple command language, local variables and integer-valued signals.

1 Introduction

μ-Charts [4] are a visual representation used for the specification of cyclic components of reactive systems, *i.e.* systems which are in some environment to which they continually react; mechanisms driven by graphical user interfaces or those driven by signals received on a communication interface are examples of such systems. They extend finite state transition diagrams by adding modularisation through hierarchical decomposition, *i.e.* allowing states to contain other μ-charts, and by parallel composition, *i.e.* allowing the modelling of separate communicating processes. In both these cases μ-charts can then communicate via instantaneously broadcast signals.

The μ-chart formalism that is the basis of the translation, given in [6], is itself based on a preceding variant called Mini-Statecharts, and these are themselves based on the original Statecharts ([1]). μ-Charts, or some of their predecessors, are widely used by engineers in specifying and designing many sorts of reactive systems. Furthermore, unlike many visually-based notations, they have a denotational semantics which gives a precise and well-defined meaning to each chart.

Overall, our strategy for specifying and reasoning about reactive systems has two key aspects: to allow ourselves to exploit the visual nature of μ-charts and the specification structuring properties of Z; to be in a position to use a reliable proof assistant.

The fact that μ-charts or similar formalisms are widely used by engineers, and the fact that Z is widely used by software engineers, were also important reasons for our strategy. Finally, it is certainly the case that having both model-checking-based and deduction-based methods at our disposal to investigate systems is advantageous.

Z/EVES [7] is a type checking and theorem proving tool for Z specifications. Theorems can be defined and proofs attempted at any time. Z/EVES was developed by ORA [3] and is used here to prove properties about the Z translated from μ-charts. Very importantly, Z/EVES has been developed and used over a number of years on many different projects, some very large and with safety-critical components. We have high confidence in its correct embodiment of the logic of Z and therefore high confidence in the properties we prove of our systems. Confidence in the usefulness and reliability of a proof assistant cannot be over-valued.

2 The central locking example

In Figure 1 we give a first example of a μ-chart. This example is taken from [4]. It specifies the central locking system for a car and considers, amongst other things, how such a system should react in the case of a crash. The system is required to unlock all the doors if a crash happens.

2.1 The chart

States in a μ-chart are shown by ellipses (double ellipses denote start states for their respective μ-chart) and transitions are labelled as shown. A transition is triggered if the signals appearing before the '/' are present (if there is nothing written there then the transition is always triggered) and

1530-1362/00 $10.00 © 2000 IEEE

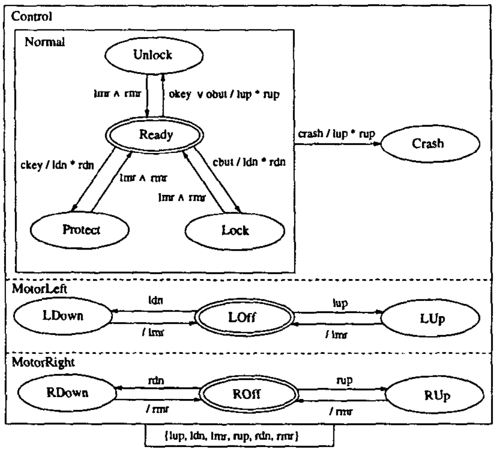

Figure 1: **The central locking system**

they emit the signals in the set after the '/'. The dashed lines show that the three sub-charts *Control*, *MotorLeft* and *MotorRight* work in parallel. The signals appearing in the small box at the bottom of the chart's border indicate that the named signals are broadcast throughout the enclosed charts.

If the system is in its starting configuration and receives (from the underlying hardware) the signal *ckey* then it moves to state *Protect* and emits and broadcasts to all sub-charts the signals *ldn* and *rdn*. These signals simultaneously cause *MotorLeft* to move to state *LDown* and *MotorRight* to move to state *RDown*, *i.e.* modelling locking of the doors. At the next tick of the system clock, *MotorLeft* and *MotorRight* will each move to their respective states *LOff* and *ROff* and emit the signals *lmr* and *rmr* respectively. (It is assumed here that locking takes only one unit of time.) The signals *lmr* and *rmr* will be instantaneously broadcast, so *Control* will move from *Protect* to *Ready*, emitting nothing.

2.2 The Z

The Z translation process gives us, broadly, Z state schemas associated with each μ-chart state and Z operation schemas associated with each transition, together with an operation schema, that we usually call *Step*, which describes what happens during one step of the system (which can be thought of as the processes' behaviour at each tick of a system's global clock).

A crucial part of the philosophy of Z is that of an *observation*. An observation can be thought of as a window, with a name to uniquely identify it, through which we can look at some part of the system being modelled. A Z state schema collects together those observations which go to make some useful conceptual part of the system being modelled. Typically, the whole model is made of several state schemas, each describing a conceptually meaningful aspect of the system, combined together.

A Z state schema has two parts: that above the line shows us what observations of the system go to make up this part of the state; that below the line places constraints on allowable values of the observed quantities when the system is in this part of the state. A Z operation schema also has two parts: that above the line again mentions observations, though this time the "before" and "after" values of observations may be referred to—by convention the "after" value of an observation is denoted by priming the relevant label. As with the state schemas, below the line in an operation schema we are given constraints that the observations must satisfy when this operation happens. Typically, these constraints tell us what relationships exist between "before" and "after" values of the observations via predicates over unprimed and primed labels respectively.

Throughout this paper we will give only highlights of the translation. The full story is available in [5].

Referring to the central locking system translation, we have state schemas:

$$\begin{array}{|l}
\hline
\textit{Control}_c \\
c_{Control} : \mu_{State} \\
\hline
\end{array}
\qquad
\begin{array}{|l}
\hline
\textit{InitControl} \\
\textit{Control}_c \\
\hline
c_{Control} = \textit{normal} \\
\hline
\end{array}$$

$$\begin{array}{|l}
\hline
\textit{Normal} \\
\textit{Control}_c \\
\hline
c_{Control} = \textit{normal} \\
\hline
\end{array}
\qquad
\begin{array}{|l}
\hline
\textit{Crash} \\
\textit{Control}_c \\
\hline
c_{Control} = \textit{crash} \\
\hline
\end{array}$$

$$\begin{array}{|l}
\hline
\textit{Normal}_c \\
c_{Norm} : \mu_{State} \\
\hline
\end{array}
\qquad
\begin{array}{|l}
\hline
\textit{InitNormal} \\
\textit{Normal}_c \\
\hline
c_{Norm} = \textit{ready} \\
\hline
\end{array}
\qquad
\begin{array}{|l}
\hline
\textit{Ready} \\
\textit{Normal}_c \\
\hline
c_{Norm} = \textit{ready} \\
\hline
\end{array}$$

$$\begin{array}{|l}
\hline
\textit{Unlock} \\
\textit{Normal}_c \\
\hline
c_{Norm} = \textit{unlock} \\
\hline
\end{array}
\qquad
\begin{array}{|l}
\hline
\textit{Lock} \\
\textit{Normal}_c \\
\hline
c_{Norm} = \textit{lock} \\
\hline
\end{array}
\qquad
\begin{array}{|l}
\hline
\textit{Protect} \\
\textit{Normal}_c \\
\hline
c_{Norm} = \textit{protect} \\
\hline
\end{array}$$

which describe the sequential μ-charts *Control*, *Normal* and *Crash*. Note that the hierarchical nature of μ-charts means that some states are not atomic, like *Crash*, but are both states and charts, like *Control* and *Normal*.

We also have operation schemas like:

$$\begin{array}{|l}
\hline
\delta_{NormalCrash} \\
\textit{Normal} \\
\textit{Crash}' \\
i? : \mathbb{P}\,\textit{Signal} \\
\textit{active}\_ : \mathbb{P}\,\mu_{State} \\
o_{Ctrl}!, o! : \mathbb{P}\,\textit{Signal} \\
\hline
\textit{active}(\textit{control}) \\
\textit{crash}_s \in i? \cup (o! \cap f_{Control}) \\
o_{Ctrl}! = \{\textit{lup}, \textit{rup}\} \\
\hline
\end{array}$$

1530-1362/00 $10.00 © 2000 IEEE

$$\begin{array}{|l}
\multicolumn{1}{l}{\delta_{ReadyProtect}} \\
\hline
Ready \\
Protect' \\
i? : \mathbb{P}\, Signal \\
active\_ : \mathbb{P}\, \mu_{State} \\
o_{Norm}!, o! : \mathbb{P}\, Signal \\
\hline
active(normal) \\
ckey \in i? \cup (o! \cap f_{Norm}) \\
o_{Norm}! = \{ldn, rdn\} \\
\hline
\end{array}
\qquad
\begin{array}{|l}
\multicolumn{1}{l}{\delta_{OffDownLeft}} \\
\hline
Off_l \\
Down_l' \\
i? : \mathbb{P}\, Signal \\
active\_ : \mathbb{P}\, \mu_{State} \\
o_{Norm}!, o! : \mathbb{P}\, Signal \\
\hline
active(motorleft) \\
ldn \in i? \cup (o! \cap f_{MotorLeft}) \\
o_{Norm}! = \{\} \\
\hline
\end{array}$$

which describe the transitions from *Normal* to *Crash*, *Ready* to *Protect* and from *LOff* to *LDown* in *MotorLeft*.

To give an idea of what this translation is capturing we consider the operation schema $\delta_{NormalCrash}$ in more detail. Above the line we mention observations that this operation depends upon. First we include the state schemas *Normal* and *Crash'*. *Normal* is unprimed and so denotes the state that the transition starts from and *Crash'* is primed, denoting that this is the state that the transition ends in. $i?$ is the input set of signals that the transition is reacting to, *active* introduces a predicate that tells us which of the specifications charts are currently active (see below for a more detailed description of this predicate) and $o_{Ctrl}!$ and $o!$ describe, respectively, the output signal sets from the chart *Control* and from the whole system.

Below the line we have three predicates, to be read in logical conjunction. The first says that this operation can happen only when *Control* is an active chart. The second says that the operation (and so the transition it translates) can happen only if the signal $crash_s$ (subscripted to differentiate this value from the state name) is in the set of signals either input from the environment or fed back within the chart. The final predicate says that in any case where this operation happens (*i.e.* when the translated transition fires) the output signals are *lup* and *rup*.

As mentioned above, each operation schema contains a predicate that describes which of the specification's charts are currently active. The translation described in [6] was modified to include this predicate and hence more fully capture the modularization and compositional principles inherent in the μ-charts formalism. Using this predicate to abstract on the "activeness" of individual sequential charts allows their translation to be performed independently of their position in any specification hierarchy. Details of the specification's hierarchical nature only becomes evident when all of the charts are combined in the schema *Step* as described below. For a more detailed discussion the interested reader is referred to [5].

Finally, we give the schema *Step* which describes the processes' possible behaviours at each cycle (tick of the system clock) with respect to the state of each process and the input signals which have been presented to the system since the last cycle. This schema is defined as follows:

$$\begin{array}{|l}
\multicolumn{1}{l}{Step} \\
\hline
\Delta Control_c \\
\Delta Normal_c \\
\Delta MotorLeft_c \\
\Delta MotorRight_c \\
i?, o! : \mathbb{P}\, Signal \\
\hline
\exists\, o_{Ctrl}!, o_{Norm}!, o_{Norm}!, o_{Norm}! : \mathbb{P}\, Signal; \\
\quad active\_ : \mathbb{P}\, \mu_{State} \bullet \\
\quad (active(control) \Leftrightarrow true) \wedge \\
\quad (active(normal) \Leftrightarrow c_{Control} = normal) \wedge \\
\quad (active(motorleft) \Leftrightarrow true) \wedge \\
\quad (active(motorright) \Leftrightarrow true) \wedge \\
\quad o! = o_{Ctrl}! \cup o_{Norm}! \cup o_{Norm}! \cup o_{Norm}! \wedge \\
\quad \delta_{Control} \wedge \delta_{Normal} \wedge \delta_{MotorLeft} \wedge \delta_{MotorRight} \\
\hline
\end{array}$$

where:

$$\begin{aligned}
\delta_{MotorRight} \mathrel{\widehat{=}} {} & \delta_{OffUpRight} \vee \delta_{UpOffRight} \vee \delta_{OffDownRight} \\
& \vee \delta_{DownOffRight} \vee \epsilon_{MotorRight} \vee Inactive_{MotorRight}
\end{aligned}$$

which just gathers together in disjunction all the possible operations in the chart *MotorLeft* (including 'doing nothing' in $\epsilon_{MotorRight}$ and being an inactive state in $Inactive_{MotorRight}$), and similarly for $\delta_{Control}$, δ_{Normal} and $\delta_{MotorLeft}$.

Further details of the Z are not important for our discussion here—save to say that they ensure that the intended meaning of the chart is faithfully captured so that we have the benefit of a second description of the model. With this model we can go on to use Z support tools to infer facts about the original chart model inductively.

2.3 Using Z/EVES to demonstrate a problem

If the system is in its initial configuration when a crash happens, which gives rise to the signal $crash_s$, at the same instant as the system emits the *ckey* signal (perhaps as a result of the crash affecting the electronics of the vehicle), *MotorLeft* and *MotorRight* each have a choice about what they do next since, for example, *MotorLeft* in state *LOff* given both the signals $\{ldn, lup\}$ can either move to state *LDown* or to state *LUp*.

We can show that the system has this unwanted and dangerous behaviour by trying to prove the following predicate. This predicate constrains the schema *Step* so that it describes all behaviours of the specification, when it is in states *Ready* (in the chart *Normal*), *LOff* (of chart *MotorLeft*) and *ROff* (of chart *MotorRight*), and both the input signals $crash_s$ and *ckey* have been emitted by the system.

$$\begin{aligned}
Step[& c_{Control} := normal, c_{Norm} := ready, c_{MotorLeft} := off_l, \\
& c_{MotorRight} := off_r, i? := \{crash_s, ckey\}];
\end{aligned}$$

Z/EVES is used to simplify this predicate using all the logical and Z rules that are embodied in it. From this simplification we get:

$$\begin{aligned}&o! = \{lup\} \cup (\{rup\} \cup (\{ldn\} \cup \{rdn\})) \wedge \\ &c'_{Norm} = protect \wedge c'_{Control} = crash \\ &\wedge (c'_{MotorLeft} = up_l \wedge \neg\, c'_{MotorRight} = up_r \\ &\quad \Rightarrow c'_{MotorRight} = down_r) \\ &\wedge (c'_{MotorRight} = up_r \wedge \neg\, c'_{MotorLeft} = up_l \\ &\quad \Rightarrow c'_{MotorLeft} = down_l) \\ &\wedge (c'_{MotorRight} = up_r \vee c'_{MotorLeft} = up_l \\ &\quad \vee c'_{MotorLeft} = down_l \wedge c'_{MotorRight} = down_r)\end{aligned}$$

which describes the dangerous situation where it is possible for one or both the motors to lock the doors (the motors move the locks)—which is of course exactly what we don't want to happen.

This shows a typical use of tools like Z/EVES: we formulate properties that we want the system to satisfy (or not satisfy) and then use the tool (which embodies the underlying semantics of Z and hence, via our translation, of μ-charts) to prove (or disprove) them. In this way, at the specification stage of development, we can satisfy ourselves that the system correctly models desirable properties before going on to the expensive step of implementation.

The problem we showed above can be easily fixed in this case: we simply block the unwanted locking motor movements by changing the triggers on the relevant transitions (from *LOff* to *LDown* in *MotorLeft* and similarly in *MotorRight*) to $ldn \wedge \neg\, crash$ and $rdn \wedge \neg\, crash$. If we do this then given the same predicate we now get the following simplified predicate from Z/EVES:

$$\begin{aligned}&c'_{Control} = crash \wedge c'_{MotorLeft} = up_l \\ &\wedge c'_{MotorRight} = up_r \wedge c'_{Norm} = protect \\ &\wedge o! = \{lup\} \cup (\{rup\} \cup (\{ldn\} \cup \{rdn\}))\end{aligned}$$

which is what we want since this says that both motors unlock as their only "choice". (There is still a, more subtle, problem with this specification which the reader might like to ponder.)

3 Extending μ-charts

This section introduces the extensions to μ-charts that we will be translating into Z. The extensions include the addition of local variables, integer-valued signals and a simple command language for transitions that is used to manipulate these new specification attributes.

Our motivation for adding these extensions to the μ-charts language, apart from the "state explosion" problem of [9], is the need to model systems that react to values and parameters from the environment in which the system being specified resides.

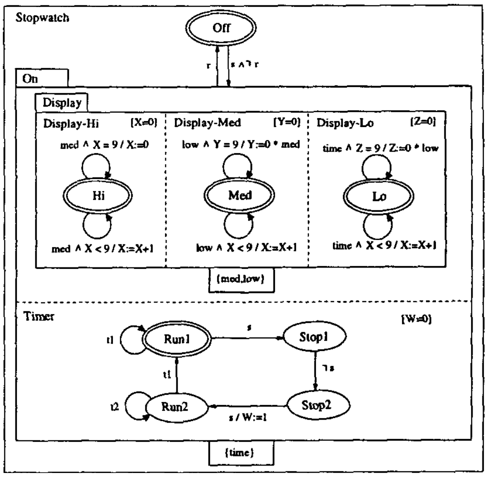

Figure 2: **A stopwatch**

We describe each of the parts of the extension in turn using the μ-chart pictured in Figure 2 as an example. This chart is derived from an example in [8] describing a stopwatch that is assumed to have an external 1 MHz clock.

The transitions labelled *t*1 are in each case an abbreviation for two transitions with the labels $\neg\ s \wedge W < 10^5/W := W + 1$ and $\neg\ s \wedge W \geq 10^5/X := 0 * time$, respectively. Similarly the transition labelled *t*2 represents two transitions with labels $s \wedge W < 10^5/W := W + 1$ and $s \wedge W \geq 10^5/W := 0 * time$.

3.1 Local Variables

Local variables are local to the sequential μ-chart in which they occur and their value can be referenced and updated by transitions within their μ-chart. Local variables are considered to be of type integer here but could easily be extended to be of arbitrary (Z-definable) type. Each sequential μ-chart in a specification containing local variables describes the value that the variable is initialised to when the chart is entered, *e.g.* $[X = 0]$ in Figure 2.

The local variables in the example are W, X, Y and Z where X, Y and Z are used to represent the display digits of the stopwatch, and W is used to convert the 1MHz clock into one that ticks every tenth of a second. Without local variables the digits would be difficult to represent and the calculation of seconds *etc.* from the 1MHz clock almost impossible, since it is certainly infeasible (from a design and reasoning point-of-view) to have 10^5 plus states in the μ-chart, *i.e.* we would have the "state explosion" mentioned above.

Proceedings of the Seventh Asia-Pacific Software Engineering Conference (APSEC'00)
1530-1362/00 $10.00 © 2000 IEEE

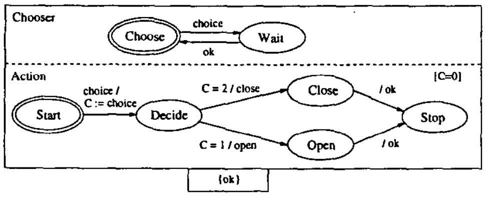

Figure 3: **Specifying a very simple menu-driven system using μ-charts**

3.2 Integer-valued signals

As an example of using integer-valued signals consider Figure 3. This is an example of an *interactive* system, which is another important sort of system that we will be extending our work to deal with. This example is of a very simple menu driven system. It consists of two charts. The *Chooser* moves to a wait state once the user has chosen an item from the menu, allowing further interaction only once the *ok* signal has been received, signifying the fact that the chosen action has been completed. The *Action* chart initiates appropriate action depending on the value carried by the integer-valued signal *choice*. This example, while very simple, shows how a typical menu-driven GUI could be specified (and then investigated) within our framework.

In fact, just as for local variables, we can allow signals to carry any Z-definable value.

3.3 Command Language

The command language that we introduce in this section is based on [8]. It allows the local variables and integer-valued signals described above to be manipulated by transitions within a sequential μ-chart.

In the example in Figure 2 the command $low \wedge Y = 9/Y := 0 * med$ informally says that, assuming the chart is currently in state *Med*, if the signal *low* is input and the local variable Y has a value of nine then after this transition the local variable Y has been updated to have value zero and the signal *med* is emitted. Since this signal is fed back, it will cause some transition in *Display-Hi*, depending on the value of the local variable X in that chart.

4 Updating the Translation

In this section we examine the changes in the translation process needed for the extensions to local variables and a command language.

At the top-most level (*i.e.* between states *On* and *Off*) the translation works just as before, so we get state schemas:

$$\begin{array}{|l} \textit{Stopwatch}_c \\ \hline c_{Stopwatch} : \mu_{State} \\ \hline \end{array} \quad \begin{array}{|l} \textit{InitStopwatch} \\ \hline \textit{Stopwatch}_c \\ \hline c_{Stopwatch} = \textit{off} \\ \hline \end{array} \quad \begin{array}{|l} \textit{Off} \\ \hline \textit{Stopwatch}_c \\ \hline c_{Stopwatch} = \textit{off} \\ \hline \end{array}$$

$$\begin{array}{|l} \textit{On} \\ \hline \textit{Stopwatch}_c \\ \hline c_{Stopwatch} = \textit{on} \\ \hline \end{array}$$

and operation schemas like:

$$\begin{array}{|l} \delta_{OffOn} \\ \hline \textit{Off} \\ \textit{On}' \\ i? : \mathbb{P}\,\textit{Signal} \\ \textit{active}_{-} : \mathbb{P}\,\mu_{State} \\ o_{Stopwatch}!, o! : \mathbb{P}\,\textit{Signal} \\ \hline \textit{active}(\textit{stopwatch}) \\ s \in i? \cup (o! \cap f_{Stopwatch}) \\ \neg\, r \in i? \cup (o! \cap f_{Stopwatch}) \\ o_{Stopwatch}! = \{\} \\ \hline \end{array} \quad \begin{array}{|l} \delta_{OnOff} \\ \hline \textit{On} \\ \textit{Off}' \\ i? : \mathbb{P}\,\textit{Signal} \\ \textit{active}_{-} : \mathbb{P}\,\mu_{State} \\ o_{Stopwatch}!, o! : \mathbb{P}\,\textit{Signal} \\ \hline \textit{active}(\textit{stopwatch}) \\ r \in i? \cup (o! \cap f_{Stopwatch}) \\ o_{Stopwatch}! = \{\} \\ \hline \end{array}$$

The *Timer* sub-chart gives rise to the expected sorts of the state schemas, like:

$$\begin{array}{|l} \textit{Timer}_c \\ \hline c_{Timer} : \mu_{State} \\ W : \mathbb{N} \\ \hline \end{array} \quad \begin{array}{|l} \textit{InitTimer} \\ \hline \textit{Timer}_c \\ \hline c_{Timer} = \textit{run}1 \\ W = 0 \\ \hline \end{array} \quad \begin{array}{|l} \textit{Run}1 \\ \hline \textit{Timer}_c \\ \hline c_{Timer} = \textit{run}1 \\ \hline \end{array}$$

$$\begin{array}{|l} \textit{Run}2 \\ \hline \textit{Timer}_c \\ \hline c_{Timer} = \textit{run}2 \\ \hline \end{array} \quad \begin{array}{|l} \textit{Stop}1 \\ \hline \textit{Timer}_c \\ \hline c_{Timer} = \textit{stop}1 \\ \hline \end{array} \quad \begin{array}{|l} \textit{Stop}2 \\ \hline \textit{Timer}_c \\ \hline c_{Timer} = \textit{stop}2 \\ \hline \end{array}$$

and operation schemas like:

$$\begin{array}{|l} \delta_{Run1Stop1} \\ \hline \textit{Run}1 \\ \textit{Stop}1' \\ i? : \mathbb{P}\,\textit{Signal} \\ \textit{active}_{-} : \mathbb{P}\,\mu_{State} \\ o_{Timer}!, o! : \mathbb{P}\,\textit{Signal} \\ \hline \textit{active}(\textit{timer}) \\ s \in i? \cup (o! \cap f_{Timer}) \\ W' = W \\ o_{Timer}! = \{\} \\ \hline \end{array} \quad \begin{array}{|l} \delta^{1}_{Run2Run1} \\ \hline \textit{Run}2 \\ \textit{Run}1' \\ i? : \mathbb{P}\,\textit{Signal} \\ \textit{active}_{-} : \mathbb{P}\,\mu_{State} \\ o_{Timer}!, o! : \mathbb{P}\,\textit{Signal} \\ \hline \textit{active}(\textit{timer}) \\ \neg\, s \in i? \cup (o! \cap f_{Timer}) \\ W < 100000 \\ W' = W + 1 \\ o_{Timer}! = \{\} \\ \hline \end{array}$$

Here we can see how the local variable W has been dealt with by the translation. It first appears in the state schema $\textit{Timer}_c$, it is initialised (to 0) in *InitTimer* and it then appears in each of the operation schemas. Note that where a transition does not change W (like the transition from *Run*1 to *Stop*1) then in the corresponding operation schema

($\delta_{Run1Stop1}$) the "before" and "after" values of W are equated, *i.e.* the value of W does indeed not change due to the transition being modelled. In contrast, in a transition like that from *Run*2 to *Run*1, the command attached to the transition increments W, and so in the corresponding operation schema $\delta^1_{Run2Run1}$ the "after" value of W is one more than its "before" value.

Finally, in this tour of the Z highlights, the display chart for the low digits gives:

$$\begin{array}{|l}\hline \textit{Display-Lo}_c \\ \hline c_{\textit{D-Lo}} : \mu_{State} \\ Z : \mathbb{N} \\ \hline\end{array} \qquad \begin{array}{|l}\hline \textit{InitDisplay-Lo} \\ \hline \textit{Display-Lo}_c \\ \hline c_{\textit{D-Lo}} = lo \\ Z = 0 \\ \hline\end{array} \qquad \begin{array}{|l}\hline \textit{Lo} \\ \hline \textit{Display-Lo}_c \\ \hline c_{\textit{D-Lo}} = lo \\ \hline\end{array}$$

$$\begin{array}{|l}\hline \delta^1_{Lo} \\ \hline \textit{Lo} \\ \textit{Lo}' \\ i? : \mathbb{P}\, \textit{Signal} \\ \textit{active}\_ : \mathbb{P}\, \mu_{State} \\ o_{\textit{D-Lo}}!, o! : \mathbb{P}\, \textit{Signal} \\ \hline \textit{active}(\textit{display-lo}) \\ \textit{time} \in i? \cup (o! \cap f_{\textit{D-Lo}}) \\ Z = 9 \\ Z' = 0 \\ o_{\textit{D-Lo}}! = \{\textit{low}\} \\ \hline\end{array} \qquad \begin{array}{|l}\hline \delta^2_{Lo} \\ \hline \textit{Lo} \\ \textit{Lo}' \\ i? : \mathbb{P}\, \textit{Signal} \\ \textit{active}\_ : \mathbb{P}\, \mu_{State} \\ o_{\textit{D-Lo}}!, o! : \mathbb{P}\, \textit{Signal} \\ \hline \textit{active}(\textit{display-lo}) \\ \textit{time} \in i? \cup (o! \cap f_{\textit{D-Lo}}) \\ Z < 9 \\ Z' = Z + 1 \\ o_{\textit{D-Lo}}! = \{\} \\ \hline\end{array}$$

Again we can see how the commands are translated quite directly into Z from the chart.

Once we have this translation we can again investigate it. An example property that we'd like to ensure holds in this system is that the values of the local variables X, Y and Z are never bigger than 9. This can be tested by proving the predicate:

$$\begin{aligned}\forall b : \textit{Step} \bullet b.X \leq 9 \wedge b.Y \leq 9 \wedge b.Z \leq 9 \Rightarrow& \\ b.X' \leq 9 \wedge b.Y' \leq 9 \wedge b.Z' \leq 9&\end{aligned}$$

b here is a *binding* (a record-like structure which contains each label of the indicated schema *Step* and associates each label with a value of the appropriate type) whose component values are accessed by the 'dot' mechanism. So, $b.X$ accesses the value associated with the *Step* label X, and so on for the other labels.

The predicate is an invariant which says that, whatever the current state, if the values are currently in range then they will be in the next state too. Since the initial state clearly has the variables in range, this together with the invariant gives us the result we want.

5 Conclusions

We have shown how μ-charts can be used to give a clear and intuitively attractive way to specify reactive systems. By interpreting the charts in an established language like Z, which enjoys established support tools like Z/EVES, we are thus able to investigate the systems we specify, and reason about their properties. We also showed how suggested extensions to μ-charts (commands, local variables and integer-valued signals) can be used and, in turn, translated into Z, so extending our experimental and investigative power. Indeed our example in Figure 3 shows how we can bring the very important problem of specifying interactive systems into our grasp, and this area is likely to form one of our main focuses in the future.

Acknowledgments

We would like to thank: the referees for their comments; our colleagues on the ISuRF project [2]; the New Zealand government's Foundation for Research, Science and Technology (FRST) for a grant which funds this work.

References

[1] D. Harel. Statecharts: A visual formalism for complex systems. *Science of Computing*, pages 231–274, 1987.

[2] www.cs.waikato.ac.nz/Research/fm/isurf.html.

[3] www.ora.on.ca.

[4] J. Philipps and P. Scholz. Compositional specification of embedded systems with statecharts. In M. Bidoit and M. Dauchet, editors, *TAPSOFT '97: Theory and Practice of Software Development*, number 1214 in LNCS, pages 637–651. Springer-Verlag, 1997.

[5] Greg Reeve and Steve Reeves. μ-Charts and Z: Extending the translation. Technical Report 00/11, Department of Computer Science, University of Waikato, 2000.

[6] Greg Reeve and Steve Reeves. μ-Charts and Z: Hows, whys and wherefores. In W. Grieskamp, T. Santen, and B. Stoddart, editors, *Integrated Formal Methods 2000: Proceedings of the 2nd. International Workshop on Integrated Formal Methods*, LNCS 1945. Springer-Verlag, 2000.

[7] M. Saaltink. The Z/EVES system. In J. Bowen, M. Hinchey, and D. Till, editors, *Proc. 10th Int. Conf. on the Z Formal Method (ZUM)*, volume 1212 of *Lecture Notes in Computer Science*, pages 72–88. Springer-Verlag, Berlin, April 1997.

[8] P. Scholz. *Design of Reactive Systems and their Distributed Implementation with Statecharts*. PhD thesis, Institut für Informatik, Technische Universität München, August 1998. TUM-I9821.

[9] Peter Scholz. An extended version of mini-statecharts. Technical Report TUM-I9628, Technische Univerität München, 1996.