2017 24th Asia-Pacific Software Engineering Conference Workshops

Characterising Sound Visualisations of Specifications using Micro-charts and Refinement

Colin Pilbrow
Department of Computer Science
University of Waikato, Hamilton
New Zealand
Email: colinpilbrow@gmail.com

Steve Reeves
Department of Computer Science
University of Waikato, Hamilton
New Zealand
Email: steve.reeves@waikato.ac.nz

***Abstract*—For validation or for communication with a client, it is useful to create a visualisation of a specification. It is important that the visualisation does not mislead the user. In this work we look at how to characterise the conditions for a micro-chart visualisation to be sound. This is done by introducing a new operator to the micro-chart semantics, allowing us to use Z data refinement to find a refinement relation between a Z specification and its (claimed sound) micro-chart visualisation. If the relation does not hold, then the visualisation is not sound.**

I. Introduction

Visualisations are a useful tool to help us to understand complex systems. They can be used to help validate systems, or become a middle ground for communication between the programmer and the client. However, an incorrectly built visualisation can be misleading. An unsound visualisation could fail to show the behaviour of a specification, and hence fail to show an error in it, or fail to show behaviour that is missing from the system. This is part of an ongoing investigation which aims at characterising the soundness of a visualisation using refinement.

In this paper we will be investigating the use of μ-charts (pronounced "micro-charts") as visualisations of Z specifications and introduce a new operator to the μ-chart semantics which separates the μ-chart into its component operations. This operator makes it easier to understand the μ-chart and use refinement, and hence, easier to prove that the visualisation is sound.

II. Stopwatch specification

In this section, we provide a specification of a stopwatch in Z. Z is a formal specification language that uses standard mathematical notation to describe and model computing systems precisely and unambiguously. As it has been used for many decades in both academia and industry we leave the details to some of the many books written about it, for example [1]–[4]. In industry, Z is used for safety-critical systems, such as air traffic control [5].

Stopwatch is the state schema (it defines the set of states that the system can be in, i.e. the state space) which has two observations (which name values that can be observed), the current time on the stopwatch and whether the stopwatch is paused or playing. The observation *time* cannot have a value larger than *maxTime*:

$$\begin{array}{|l}
\textit{Stopwatch} \\
\hline
\textit{time} : \mathbb{N} \\
\textit{playing} : \textit{BOOL} \\
\hline
\textit{time} \leq \textit{maxTime} \\
\hline
\end{array}$$

The following schemas are operation schemas that specify how the state changes when the operations are used. Initially no time has passed and the stopwatch is paused:

$$\begin{array}{|l}
\textit{Init} \\
\hline
\textit{Stopwatch}' \\
\hline
\textit{time}' = 0 \\
\textit{playing}' = \textbf{false} \\
\hline
\end{array}$$

Pushing the Pause/Play button causes the stopwatch to start playing if it is paused, or pause if it is currently playing:

$$\begin{array}{|l}
\textit{Pause/Play} \\
\hline
\Delta \textit{Stopwatch} \\
\hline
\textit{time} = \textit{time}' \\
\textit{playing} = \neg\, \textit{playing}' \\
\hline
\end{array}$$

If the stopwatch is reset then time is reset to 0 and the stopwatch is paused again:

$$\begin{array}{|l}
\textit{Reset} \\
\hline
\Delta \textit{Stopwatch} \\
\hline
\textit{time}' = 0 \\
\textit{playing}' = \textbf{false} \\
\hline
\end{array}$$

Every time the *Tick* happens the stopwatch will either increase the time it has recorded by one second if it is playing, or stay paused (and not increase the time passed) otherwise:

$$\begin{array}{|l}
\textit{Tick} \\
\hline
\Delta \textit{Stopwatch} \\
\hline
(\textit{playing} = \textbf{true} \wedge \textit{time}' = \textit{time} + 1 \\
\vee\ \textit{playing} = \textbf{false} \wedge \textit{time}' = \textit{time}) \\
\textit{playing} = \textit{playing}' \\
\hline
\end{array}$$

978-1-5386-2649-8/17 $31.00 © 2017 IEEE
DOI 10.1109/APSECW.2017.16

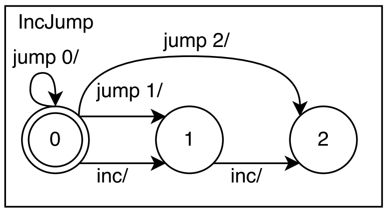

Fig. 1: Simple Sequential Microchart

III. Visualising the specification

A system visualisation is a tool used to help users understand and analyse a system. The purpose of a visualisation is to allow users to quickly and easily understand complex information, and it reinterprets the formal mathematical notation of the specification in a visual language that clients can, hopefully, better understand. We will be focusing exclusively on the property of *soundness*, i.e. that the visualisation is accurate and does not mislead the user. Other important visualisation properties include usability, clarity, and aesthetics; however we will not be addressing these properties in this paper. Z in Practice [4] has a large section about animating Z, including subsections for what, why, how and when, in the validation chapter.

IV. Microcharts

μ-Charts (with a large "C", to differentiate the language from the things it denotes) is a graphical language which uses states and guarded transitions to show the behaviour of a reactive system. μ-Charts is a subset of UML statecharts that includes a formal semantics defined in Z [6]–[8]. We have chosen to use μ-Charts as we have found that μ-charts can create more sophisticated visualisations than state machines, while the formal semantics lets us approach the problem differently.

A μ-chart receives sets of input signals from the environment (we assume that these arrive together, perhaps each time the environment in which the chart is embedded makes a notional "tick") which can cause the state of the μ-chart to change, and when it may also output signals back to the environment. All this happens (at this level of abstraction) instantaneously.

The semantics of μ-charts also lets us compose charts together, include local variables, and feed signals back into the chart that may instantaneously trigger additional transitions. These properties allow us to visualise a complex system in fewer states than the state machines we investigated previously in [9]. μ-charts have been used to represent PIMs, which are models of the dynamic behaviour of interface designs [10].

Figure 1 is an example of a simple sequential μ-chart. In this trivial visualisation, we see that states can be reached by jumping directly from the initial state, or by incrementing through each state. Simple sequential μ-charts are similar to state machines, but the largest difference is how the transitions are triggered. μ-chart transitions are labelled. On the left hand side of the / in the label of the transitions there is a *guard*. This is a set of signals, and when all of these signals are received from the environment (or via feed back), the transition will trigger, moving the system into a new state and outputting any signals listed on the right hand side of the / (the *action*). So, in figure 1, if we receive the signals *jump* and 1 while we are in the initial state, state 0, then the system will transition to state 1 and output nothing.

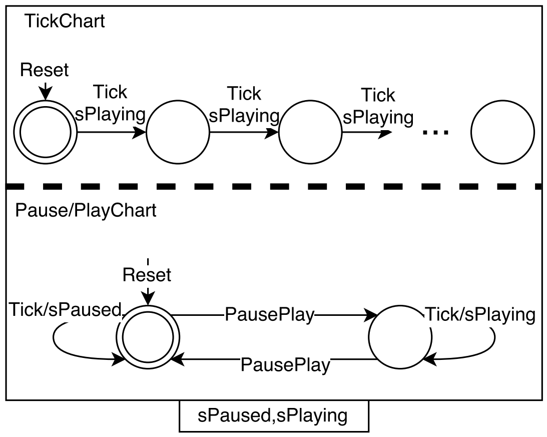

Fig. 2: Composed Microchart

A. Visualising Z Specifications using Microcharts

In this section we look at how μ-charts can be used to create more sophisticated visualisations than state machines. Let us first look how we may compose multiple charts together.

We have used composition to create Figure 2, a visualisation of the stopwatch specification. The top chart shows what the current time value is, while the bottom chart shows whether the stopwatch is paused or playing. The ability to compose several charts allows us to separate the different observations in the specification into separate charts. This allows us to quickly see which operations affect which observations, and by removing the need to visualise every possible combination of values we can greatly reduce the number of states in the visualisation.

The bottom chart outputs feedback signals *sPlaying* and *sPaused* which the top chart uses in its guards in some transitions to ensure that the state only changes when the stopwatch is playing. Feedback and input/output all happen instantaneously on the same tick, so if one chart outputs a signal in a transition which is the guard of a transition in another chart then both transitions occur at the same time.

Secondly, we can build μ-charts that include local variables, such as in figure 3.

Local variables are used to store information. Compared to figure 2 that visualises each possible value of time as a different state, figure 3 uses a local variable to store the current value of time, greatly reducing the number of states.

The transition guards can also include comparisons and the actions can include assignments. For example *time* $:= 0$ resets the value of our local variable *time* to 0. We can also use value-laden signals to assign our local variables with values from the environment.

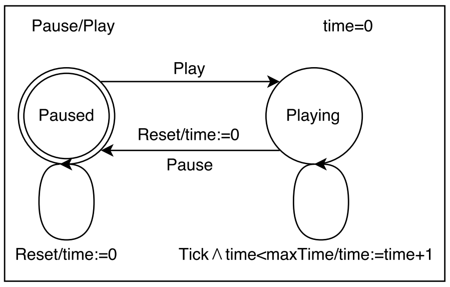

Fig. 3: Microchart with Local Variable

V. Soundness of Visualisations

Our goal is to prove that these visualisations, and other μ-chart visualisations, are *sound*.

If the visualisation shows that some action is possible, then it must also be possible in the specification. For example, the *IncJump* visualisation shows that we can use either the *Inc* operation or the *Jump* operation to move from state 0 to state 1. So, if this is a sound visualisation, then this should also be possible in the specification, and the visualisation would be unsound if either operation did not allow this in the specification. Clearly, a μ-chart that visualises incorrect or impossible behaviour is misleading and, we say, unsound.

If all traces and changes of state in the μ-chart visualisation are also in the specification, then the visualisation is sound.

Under scrutiny, this is a very informal definition of soundness, as it is not always straightforward to see how to identify behaviour that the μ-chart shows but which is not in the specification, or even what it means for such behaviour to be *in* the specification at all. We aim to characterise this so that we can formally investigate whether a μ-chart visualisation is sound, and we will do so with refinement.

A. Principle of Substitutivity

We use refinement because of the basic idea behind it, the Principle of Substitutivity (PoS).

> It is acceptable to replace a program by another, provided it is impossible for a user of the program to observe that the substitution has taken place. If a program can be acceptably substituted by another, then the second program is said to be a refinement of the first. [11]

If the user, via the substituted program (or visualisation), can do everything they did with the original then it must be an acceptable solution.

A thorough description of the formal Z refinement rules can be found elsewhere, for example in [1], [11], we only give a brief description of the purpose of each of the rules.

Data refinement includes a relation R between the visualisation state space and the specification state space. This allows us to transform the observations in the specification to states in the μ-chart, and defines how we are visualising the specification.

Firstly, the new visualisation (in our case) set of initial states (those states of *CState* that are characterised by *CInit*) must be a subset of the old (specification) set of initial states (those states of *AState* that are characterised by *AInit*). If this were not true, it would mean that the visualisation could start in an incorrect state:

$$\forall CState' \bullet CInit \Rightarrow \exists AState' \bullet AInit \land R'$$

Then, for each visualisation operation COp_i, we must test for applicability and correctness: (a) it can be used in all the situations that the specification of the corresponding operation AOp_i says it can be used (and perhaps more situations), which is formalised via the idea of precondition, expressed in Z by **pre** in the schema calculus; and (b) the results of its use are amongst the results specified for the operation (that is, it does not do anything to the state outside of the possibilities of what can be done to the state as allowed by the specification).

$$\forall AState;\ CState;\ \bullet\ \mathbf{pre}\, AOp_i \land R \Rightarrow \mathbf{pre}\, COp_i$$

$$\forall AState;\ CState;\ CState' \bullet \\ \mathbf{pre}\, AOp_i \land R \land COp_i \Rightarrow \exists AState' \bullet R' \land AOp_i$$

VI. Microchart Semantics

Schemas are constructed for each state, transition and chart, and the schema calculus is used to integrate these schemas into a layered system.

For example, amongst the semantic schemas for a simple composed μ-chart, the lower-level operation schemas specify the behaviour of the transitions in each chart being composed together, while the higher-level operation schemas describe the behaviour of the system after the lower-level charts have been composed. The top-level chart, which we call *SWSys*, has input and output observations for the signals to and from the environment and specifies the overall behaviour of the chart based on what signals are being received and what the current state is. *SWSys*, and the lower-level schemas, can be generated automatically using the free, open-source software *ZooM*.

We present the *SWSys* schema for the *IncJump* μ-chart and *Stopwatch* μ-chart below. This is the result after all the simplification that is possible using the schema calculus has been done. The low-level schemas and axiomatic definitions are therefore not provided here. Schema $Chart_{IJ}$ contains the current state c, while in_{IJ} and out_{IJ} are types for input and output signals.

The *SWSys* schema predicates are divided into several cases based on the current state and the input signals being received from the environment. For example, the second line of the $SWSys_{IncJump}$ is the case when the input signals being received are *Jump* and 1, the current state c is 0 and the new state c' is 1. This is the same as the transition we looked at previously in section IV, and for simple sequential μ-charts it is easy to match the transitions in the chart with cases in the semantic schema.

The μ-chart semantics we are using in this paper are an adaptation of classic μ-charts, as we are specifically using

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{SWSys}_{\textit{IncJump}}} \\
\Delta \textit{Chart}_{IJ} \\
i_{IJ}? : \mathbb{P}\, \textit{in}_{IJ} \\
o_{IJ}! : \mathbb{P}\, \textit{out}_{IJ} \\
\hline
(\textit{Op Jump} \in i_{IJ}? \wedge N\,0 \in i_{IJ}? \wedge c = S\,0 \wedge c' = S\,0 \vee \\
\textit{Op Jump} \in i_{IJ}? \wedge N\,1 \in i_{IJ}? \wedge c = S\,0 \wedge c' = S\,1 \vee \\
\textit{Op Jump} \in i_{IJ}? \wedge N\,2 \in i_{IJ}? \wedge c = S\,0 \wedge c' = S\,2 \vee \\
\textit{Op Jump} \in i_{IJ}? \wedge (0 \in i_{IJ}? \vee 1 \in i_{IJ}? \vee 2 \in i_{IJ}?) \wedge \\
c = S\,1 \wedge c' = S\,1 \vee \\
\textit{Op Jump} \in i_{IJ}? \wedge (0 \in i_{IJ}? \vee 1 \in i_{IJ}? \vee 2 \in i_{IJ}?) \wedge \\
c = S\,2 \wedge c' = S\,2 \vee \\
\textit{Op Inc} \in i_{IJ}? \wedge c = S\,0 \wedge c' = S\,1 \vee \\
\textit{Op Inc} \in i_{IJ}? \wedge c = S\,1 \wedge c' = S\,2 \vee \\
\textit{Op Inc} \in i_{IJ}? \wedge c = S\,2 \wedge c' = S\,2) \wedge \\
o_{IJ}! = \{\} \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{SWSys}_{\textit{Stopwatch}}} \\
\Delta \textit{Chart}_{SW} \\
i_{SW}? : \mathbb{P}\, \textit{in}_{SW} \\
o_{SW}! : \mathbb{P}\, \textit{out}_{SW} \\
\hline
(\textit{Op Tick} \in i_{SW}? \wedge (t^{\sim}\, c_{\textit{Tick}}) < \textit{maxTime} \wedge \\
c'_{\textit{Tick}} = t\,(t^{\sim}\, c_{\textit{Tick}} + 1) \wedge c_{PP} = p\,\mathbf{true} \wedge c'_{PP} = p\,\mathbf{true} \vee \\
\textit{Op Reset} \in i_{SW}? \wedge \\
c'_{PP} = p\,\mathbf{false} \wedge c'_{\textit{Tick}} = t\,0 \vee \\
\textit{Op PausePlay} \in i_{SW}? \wedge \\
c'_{\textit{Tick}} = c_{\textit{Tick}} \wedge c_{PP} = p\,\mathbf{true} \wedge c'_{PP} = p\,\mathbf{false} \vee \\
\textit{Op PausePlay} \in i_{SW}? \wedge \\
c'_{\textit{Tick}} = c_{\textit{Tick}} \wedge c_{PP} = p\,\mathbf{false} \wedge c'_{PP} = p\,\mathbf{true} \vee \\
\textit{Op Tick} \in i_{SW}? \wedge \\
c'_{\textit{Tick}} = c_{\textit{Tick}} \wedge c_{PP} = p\,\mathbf{false} \wedge c'_{PP} = p\,\mathbf{false} \vee \\
\textit{Op Tick} \in i_{SW}? \wedge c_{\textit{Tick}} = t\,\textit{maxTime} \wedge \\
c'_{\textit{Tick}} = c_{\textit{Tick}} \wedge c_{PP} = p\,\mathbf{true} \wedge c'_{PP} = p\,\mathbf{true}) \wedge \\
o_{SW}! = \{\} \\
\hline
\end{array}
$$

them for visualisations of Z specifications. The most notable difference is that we allow some syntactic sugar, such as the use of ellipses for a repeating pattern, and transitions with no visible start state, which can be used from any state. We are using *do-nothing* semantics, so there are additional cases for when we try to use an operation where there is no appropriate outgoing transition. In this case we simply stay in the same state.

Next, we provide the top level schema for the composed stopwatch visualisation, $\textit{SWSys}_{\textit{Stopwatch}}$. While the previous example only had one observation for the current state, c, we now have two observations c_{PP} and $c_{\textit{Tick}}$, which observe the current state for each of the two sub-charts.

As can be seen from the $\textit{SWSys}_{\textit{Stopwatch}}$ example, it is not possible to match a transition in the visualisation with a single case in the predicate part of the schema. All feedback signals that were present in the lower-level schema have been removed, and how a particular operation behaves both depends on and changes the state of each composed μ-chart. So, the first case in the predicate describes the change in states when the *Tick* operation is used while the bottom chart is in the *Playing* state and the top chart is in any state but *sTime*. When the operation is used with this precondition, the bottom state stays in *Playing* and the top chart increments to the next state.

However, if we want to use Z data refinement we need to have operations in the visualisation that match the operations in the specification. For this purpose, we introduce an additional layer to the μ-chart semantics which may be used to separate the overall behaviour of the chart into several schemas that describe the behaviour of the chart when a particular operation is used.

VII. Operation operator

From the specification we are visualising we have a set of operations, each with their own input and output observations. For each operation we restrict the schema to show only the changes to the system when that operation is used. We do this by ensuring that the set of input signals includes exactly the operation name as well as signals with the same types as any input observations of the operation. Similarly, we need to ensure that the output signal set matches the output observations of the operation. When using the operation, no other signals should be input by the environment, and the system should not output any other signals. Once we have the input and output observations, we remove the signals from the schema using hiding. This ensures that the resulting schema will match the appropriate operation, and we can use data refinement. If the signal observations were still present in the schema then it would not be a match for the operation defined in the specification.

A. Operation details

The operation operator is defined in Z. The purpose of this operator is to hide the signals, and replace them with the appropriate observations for the operation. With the input and output observations from a given operation schema *OperationName* and the μ-Chart schema, $\mu\,\textit{OperationName}$ describes a change in the state of the μ-chart rather than the specification when *OperationName* is used. In the predicate, we have *SWSys*, where the input and output signals have been hidden. When using *OperationName*, we require that the input signals include the name of the operation being used, as well as a signal for each of the input observations. Similarly, we require the output signals to include exactly the output observations.

$$
\begin{array}{|l}
\multicolumn{1}{l}{\mu\,\textit{OperationName}} \\
\Delta \textit{Chart} \\
\textit{input}_i? : \textit{Type}_i \ldots \textit{input}_j? : \textit{Type}_j \\
\textit{output}_o! : \textit{Type}_o \ldots \textit{output}_p! : \textit{Type}_p \\
\hline
\exists\, i_C?, o_C! : \mathbb{P}\,\mu\,\textit{Signals} \mid \\
i_C? = \{\mu\,\textit{Op OperationName}, \mu\,i_i\,\textit{input}_i?, \ldots \mu\,i_j\,\textit{input}_j?\} \\
\wedge\ o_C! = \{\mu\,o_o\,\textit{output}_o!, \ldots \mu\,o_p\,\textit{output}_p!\} \bullet \textit{SWSys} \\
\hline
\end{array}
$$

For example, when we use the *Reset* operation, we expect *Reset* to be the only signal being input to the μ-chart:

$$\begin{array}{|l}
\multicolumn{1}{l}{\mu Reset} \\ \hline
\Delta Chart_{SW} \\ \hline
\exists i_{SW}? : \mathbb{P}\, in_{SW}, o_{SW}! : \mathbb{P}\, out_{SW} \bullet \\
\qquad i_{SW}? = \{\mu\, Op\, Reset\} \wedge o_{SW}! = \{\} \wedge SWSys \\ \hline
\end{array}$$

Below we have applied the operation operator to the *Jump* operation of the visualisation in Figure 1. The *Jump* schema in the specification has a single input, the number that the state will jump to. Therefore, when we use the operation operator to determine how the *Jump* operation affects the μ-chart, we know that we need to input a number from 0 to 2, called $n?$. We know that the input signals being input from the environment must be *Jump* and $n?$ if we are using the *Jump* operation. Finally, we know that there will be no output signals:

$$\begin{array}{|l}
\multicolumn{1}{l}{\mu Jump} \\ \hline
\Delta Chart_{SW} \\
n? : \{0, 1, 2\} \\ \hline
\exists i_{SW}? : \mathbb{P}\, in_{SW}, o_{SW}! : \mathbb{P}\, out_{SW} \bullet \\
\qquad i_{SW}? = \{\mu\, Op\, Jump, N\, n?\} \wedge o_{SW}! = \{\} \wedge SWSys \\ \hline
\end{array}$$

We can substitute *SWSys* and simplify this schema further to get a predicate with only three cases. If we are in state 0 we jump to the state number being input, otherwise, we do not change states:

$$\begin{array}{|l}
\multicolumn{1}{l}{\mu Jump} \\ \hline
\Delta Chart_{SW} \\
n? : \{0, 1, 2\} \\ \hline
c = S\,0 \wedge c' = S\,n? \vee \\
c = S\,1 \wedge c' = S\,1 \vee \\
c = S\,2 \wedge c' = S\,2 \\ \hline
\end{array}$$

Similarly, we can build the schema $\mu\, Inc$ by applying the operation operator to the μ-chart and using the other operation in the specification:

$$\begin{array}{|l}
\multicolumn{1}{l}{\mu Inc} \\ \hline
\Delta Chart_{SW} \\ \hline
c = S\,0 \wedge c' = S\,1 \vee \\
c = S\,1 \wedge c' = S\,2 \vee \\
c = S\,2 \wedge c' = S\,2 \\ \hline
\end{array}$$

Using $\mu\, Inc$, $\mu\, Jump$ along with the initialisation and state schemas from the original μ-chart semantics, we can find a refinement relation between the specification and the visualisation.

The following schemas are the result of applying the operation operator to the composed stopwatch visualisation for each of the operations *Reset*, *Pause/Play*, and *Tick*. First, $\mu\, Reset$ shows that when we use the reset operation, the μ-chart will return to the paused state, and the initial time 0 state:

$$\begin{array}{|l}
\multicolumn{1}{l}{\mu Reset} \\ \hline
\Delta Chart_{SW} \\ \hline
c'_{PP} = cPaused \wedge c'_{Tick} = t\,0 \\ \hline
\end{array}$$

$\mu\, PP$ does not change the value of time, but toggles between the paused and playing states:

$$\begin{array}{|l}
\multicolumn{1}{l}{\mu PP} \\ \hline
\Delta Chart_{SW} \\ \hline
c'_{Tick} = c_{Tick} \wedge c_{PP} = cPlaying \wedge c'_{PP} = cPaused \vee \\
c'_{Tick} = c_{Tick} \wedge c_{PP} = cPaused \wedge c'_{PP} = cPlaying \\ \hline
\end{array}$$

Finally, $\mu\, Tick$ increases the current time, unless the stopwatch has paused or has reached the maximum time allowed. In this schema we use the relational inverse of the state observation. This lets us check that we have not yet reached the maximum time, and allows us to transition into the next consecutive state by adding 1 to the current time:

$$\begin{array}{|l}
\multicolumn{1}{l}{\mu Tick} \\ \hline
\Delta Chart_{SW} \\ \hline
(t^{\sim}\, c_{Tick}) < maxTime \wedge \\
c'_{Tick} = t\,(t^{\sim}\, c_{Tick} + 1) \wedge c_{PP} = cPlaying \wedge c'_{PP} = cPlaying \\
\vee\; c'_{Tick} = c_{Tick} \wedge c_{PP} = cPaused \wedge c'_{PP} = cPaused \\
\vee\; c_{Tick} = t\, maxTime \wedge \\
c'_{Tick} = c_{Tick} \wedge c_{PP} = cPlaying \wedge c'_{PP} = cPlaying \\ \hline
\end{array}$$

VIII. Refinement Example

In order to use Z data refinement, we needed operations with matching input and output observations. We now have the operation operator which gives us the visualisation formally written in Z with the input and output observations we require. We will work through the proof of only one of the operations, *Reset*.

The retrieve relation shows how the observations in the specification are related to the states in the visualisation:

$$\begin{array}{|l}
\multicolumn{1}{l}{R} \\ \hline
Stopwatch \\
Chart_{SW} \\ \hline
t^{\sim}\, c_{Tick} = time \wedge \\
(c_{PP} = cPaused \wedge playing = \mathbf{false} \\
\vee\; c_{PP} = cPlaying \wedge playing = \mathbf{true}) \\ \hline
\end{array}$$

We also need the preconditions of our *Reset* and $\mu\, Reset$ operations. The precondition of the visualisation operations will always be true, as we are using the *do−nothing* semantics and the operations can be used in any state.

$$\mathbf{pre}\, Reset \equiv \mathbf{true}$$
$$\mathbf{pre}\, \mu\, Reset \equiv \mathbf{true}$$

Firstly, we test *Init*. $\mu\,Init$ can easily be found using the μ-chart semantics. The initial states in the visualisation are identified by the double circles, where the time is 0 and the watch is paused:

$$\forall\, Chart'_{SW} \bullet \mu\, Init \Rightarrow \exists\, Stopwatch' \bullet Init \land R'$$

$\equiv$\{definitions\}

$$\begin{array}{l} \forall\, Chart'_{SW} \bullet t^{\sim} c'_{Tick} = 0 \land c'_{PP} = cPaused \Rightarrow \\ \exists\, playing' : Bool, time' : \mathbb{N} \bullet \\ playing' = \mathbf{false} \land time' = 0 \land R' \end{array}$$

$\equiv$\{one point rule, simplify\}

$$\begin{array}{l} \forall\, Chart'_{SW} \bullet t^{\sim} c'_{Tick} = 0 \land c'_{PP} = cPaused \Rightarrow \\ t^{\sim} c'_{Tick} = 0 \land c'_{PP} = cPaused \end{array}$$

$\equiv$\{logic\}

$$\mathbf{true}$$

Since the initial test was passed, we can test applicability. However, since the precondition for each visualisation operation is **true**, we can quickly see that applicability is true for each operation. Here we show that for *Reset*:

$$\forall\, Chart_{SW};\ Stopwatch;\ \bullet \mathbf{pre}\, Reset \land R \Rightarrow \mathbf{pre}\, \mu\, Reset$$

$\equiv$\{precondition of *Reset* as above\}

$$\forall\, Chart_{SW};\ Stopwatch;\ \bullet \mathbf{pre}\, Reset \land R \Rightarrow \mathbf{true}$$

$\equiv$\{logic\}

$$\mathbf{true}$$

The full proof tests each of the operations in the specification. We will finish with the contractual correctness proof of *Reset* as an example:

$$\begin{array}{l} \forall\, Stopwatch;\ Chart_{SW};\ Chart'_{SW};\ \bullet \\ \mathbf{pre}\, Reset \land R \land \mu\, Reset \Rightarrow \exists\, Stopwatch' \bullet R' \land Reset \end{array}$$

$\equiv$\{definitions\}

$$\begin{array}{l} \forall\, Stopwatch;\ Chart_{SW};\ Chart'_{SW};\ \bullet \\ \mathbf{true} \land R \land c'_{PP} = cPaused \land c'_{Tick} = t\,0 \Rightarrow \\ \exists\, Stopwatch' \bullet R' \land time' = 0 \land playing' = \mathbf{false} \end{array}$$

$\equiv$ \{one point rule, simplify\}

$$\begin{array}{l} \forall\, Stopwatch;\ Chart_{SW};\ Chart'_{SW};\ \bullet \\ R \land c'_{PP} = cPaused \land c'_{Tick} = t\,0 \Rightarrow \\ t^{\sim} c'_{Tick} = 0 \land c'_{PP} = cPaused \end{array}$$

$\equiv$\{logic and definitions\}

$$\mathbf{true}$$

With further proofs we find that refinement relations exist for both the Stopwatch visualisation and the IncJump visualisation. Therefore, we conclude that both visualisations are sound.

Although we have constructed the operation operator to help ease the discovery of refinement relations, this is not the only possible use. If the user is not experienced with μ-charts, or is investigating particularly complex μ-charts with decomposition and feedback, it may be necessary to refer to the semantics of the μ-chart to completely understand the meaning and ensure that nothing has been missed. Separating the chart into its component operations was useful for data refinement, and it is also useful for understanding the meaning of the μ-chart piece by piece.

IX. Conclusion

In this paper we have added a new operator to the μ-chart semantics. This operator allows us to create schemas that describe how the μ-chart responds to operations defined in Z specifications. By building these schemas we can find refinement relations between the specification and the μ-chart. We are using the μ-charts to visualise Z specifications, and by finding a refinement relation, we can ensure these visualisations are sound. Typically, refinement is used to determine a relation between an abstract and concrete specification. However, it is easier by comparison to determine if a refinement relation exists between the specification and its μ-chart visualisation as the applicability rule will always be true. As the visualisations become more complex, they become large and unusable. This method can continue to be used even with large complex visualisations.

X. Acknowledgement

Our thanks to Judy Bowen for her comments, discussions and input to this work.

References

[1] J. Woodcock and J. Davies, *Using Z: specification, refinement, and proof*. Prentice Hall Englewood Cliffs, 1996, vol. 39.

[2] D. Lightfoot, *Formal specification using Z*, 2nd ed. Palgrave, 2001.

[3] J. Jacky, *The way of Z: practical programming with formal methods*. Cambridge University Press, 1997.

[4] R. Barden, S. Stepney, and D. Cooper, *Z in Practice*. Prentice-Hall, Inc., 1995.

[5] N. White. (2010) Formal methods in air traffic control. [Online]. Available: https://www.slideshare.net/AdaCore/white-open-do

[6] G. Reeve and S. Reeves, "The syntax and semantics of μ-charts," Technical Report 04/2004, Department of Computer Science, University of Waikato, Tech. Rep., 2004.

[7] ——, "μ-charts and Z: Hows, whys, and wherefores," in *International Conference on Integrated Formal Methods*. Springer, 2000, pp. 255–276.

[8] G. Reeve, "A refinement theory for μcharts," Ph.D. dissertation, The University of Waikato, 2005.

[9] C. Pilbrow and S. Reeves, "Using state machines for the visualisation of specifications via refinement," in *Proceedings of the 24th Australasian Software Engineering Conference, ASWEC 2015, Volume II, Adelaide, SA, Australia, September 28 - October 1, 2015*, 2015, pp. 106–110. [Online]. Available: http://doi.acm.org/10.1145/2811681.2811702

[10] J. Bowen and S. Reeves, "A simplified Z semantics for Presentation Interaction Models," in *International Symposium on Formal Methods*. Springer, 2014, pp. 148–162.

[11] J. Derrick and E. A. Boiten, *Refinement in Z and Object-Z: foundations and advanced applications*. Springer Science & Business Media, 2013.