Idioms for μ-Charts

Grant Anderson and Greg Reeve and Steve Reeves
Department of Computer Science
University of Waikato
New Zealand

Abstract

This paper presents an idiomatic construct for μ-charts which reflects the high-level specification construct of synchronization between activities. This, amongst others, has emerged as a common and useful idea during our use of μ-charts to design and specify commonly-occurring reactive systems. The purpose of this example, apart from any inherent interest in being able to use synchronization in a specification, is to show how the very simple language of μ-charts can used as a basis for a more expressive language built by definitional extension.

1. Introduction

Reactive systems are systems that can react to input from the environment around them. Scholz [11] describes three different types of computer systems—transformational, interactive and reactive. *Transformational systems* have all the input they require available at the beginning of execution, and produce an output once the execution terminates. *Interactive systems* interact with their environment, but this interaction is determined by the system, rather than the environment (via prompts or allowing access to menu items at times when the system is ready to react to them). With *reactive systems*, on the other hand, it is the environment that determines the system-environment interaction (though many systems classed as interactive can be modelled as reactive, in fact).

μ-Charts ("micro-charts"), described in [6], [8], [7], can be used to provide an abstract model of reactive systems—abstract because the models assume properties (such as instantaneity in transitions) that are unlikely to be present in implementations of the systems.

μ-Charts themselves are descended from the Statecharts of Harel [1], embodied in the STATEMATE tool [3], a version of which has found its way into the UML notation [2].

Statecharts, however, are hard to give a simple semantics to. Features such as inter-level transitions, implicit interactions between 'orthogonal' charts and the interpretation via shared variables are at the heart of these complications.

μ-Charts avoid these complications, so forming a simple language which, nevertheless, is still expressive enough to describe real systems. Apart from being easier to understand and therefore use, the simplicity opens the door to a compositional semantics and the goal of a logic for charts becomes attainable.

Elsewhere (*e.g.* [9]) we have given a semantics for μ-charts via a translation to Z. This allows us to use tools like Z/EVES [10] to prove properties of our μ-chart-specified systems (*e.g.* [7]). Future research concerns deriving, via the connection with Z and its own logic and refinement rules (*e.g.* [5]), proof rules and ultimately refinement rules at the level of μ-charts themselves, *i.e.* our aim is a logic of μ-charts.

This paper contributes to this by showing how a high-level construct can be added to the chart language without having to extend the semantics or, therefore, the logic.

It introduces μ-charts in Section 2, discusses the concepts of preemption and instantaneity and how they pertain to the μ-charts language in Section 3, looks at an example of an idiomatic construct—*i.e.* a construct which can be built from basic charts and which is a good idiom for designing systems—in Section 5, and finally, in Section 6, we present some conclusions.

2. The basics of μ-charts

Here we give a short introduction to the syntax and semantics of μ-charts. However, anyone familiar with finite-state machines will find them very familiar and easy to understand, the only significant extension being that of states which are themselves charts.

Consider the upper chart in Figure 1. The chart's name *Example* appears in the top left-hand corner of the rectangle which delineates the chart. The double-walled oval named *A* is an *atomic state* which is also the *initial state* of *Example*. The rectangle *B* is another state of *Example* except that this time *B* is a state which has extra structure: it

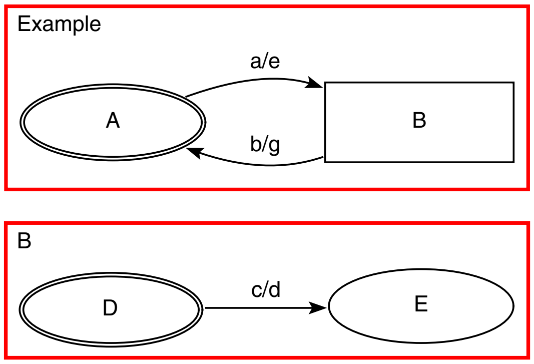


Figure 1. A simple μ-chart with decomposition

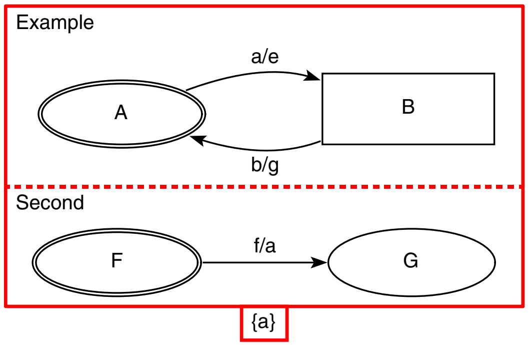


Figure 2. A μ-chart formed by composition

is itself a chart. The chart *B*'s initial state is *D* and its other state is *E*.

Charts change states in *steps* and at each step, because of the presence of certain *input signals*, *transitions* (denoted by arrows between states) happen if their *trigger expressions* (appearing to the left of the slash in the label for the transition and consisting of Boolean combinations of signals) are satisfied (*i.e.* made true) by the input set for the step. So, if the chart is in state *A* and the input set is $\{a\}$ then *Example* will move to state *B*. It will, in this step, also output the signal *e*, *i.e.* the expression appearing to the right of the slash in the transition's label.[1]

We call *B* a *decomposed state*, and note that a natural hierarchy of charts, formed by the way decomposition allow nesting of charts, can now occur.

Now consider the chart in Figure 2. Note that this is a *composition* of two charts:[2] one is *Example* as above and the other is *Second*, which has *F* as its initial state and another atomic state *G*, connected to *F* by a transition which is triggered by input *f* and gives output *a*. Note the set $\{a\}$ in the box at the foot of this chart: this tells us that the signal *a* is fed-back, as we shall see.

When two charts are composed they can each react to the inputs in a step, as before, but they can also react to any outputs that the other chart in the composition may produce. Since one fundamental property of μ-charts is that transitions happen *instantaneously* (more on this below) if one chart produces an output which, as input to the other chart, causes another transition then both transitions happen in the same step. As an example of this, imagine the input to the chart in Figure 2 is $\{f\}$ and that *Example* is in *A* and *Second* is in *F*. Then, *Second*'s transition is triggered because *f* is in the input, so it moves to *G* and *a* is output. Now, because *a* is fed-back within this chart, and because this has all happened instantaneously ('has taken no time'), *Example*'s transition labelled a/e will take place since its trigger is available as an input due to feed-back, so *Example* moves to *B* and the whole chart, in just one step, outputs $\{a, e\}$ in response to the input $\{f\}$.

3. Preemption and instantaneity

Because a chart can be in only one state[3] only one transition can occur in each chart in a given step. We therefore need to fix what happens when more than one transition is possible, *i.e.* when more than one transition is triggered, which can arise in the presence of decomposed state.

For example, consider Figure 1 again, it is possible that transitions are triggered in both *Example* and *B* in the same step. This happens if *Example* is in the state *B*, *B* is in the state *D* and the set of inputs to the chart is $\{c, b\}$. Two transitions are triggered: one is the transition in the chart *B*, from *D* to *E*, and the other is the transition in *Example*, from *B* to *A*.

If this occurs, then by a property of μ-charts called *preemption*, the highest-level transition in the decomposition hierarchy takes precedence. So, we find that due to the step above *Example* is in the state *A*, with the chart *B* *inactive*. However, both transitions are triggered, so the output for this step is $\{d, g\}$. In a sense the transition in *B* does occur, but the chart *B* becomes inactive immediately afterwards. This property of preemption is used for the synchronized transitions we discuss below.

[1]The language of transition labels is more expressive than this in general, but this simple sub-language suffices for this paper. In particular the empty expression is read as *true*, *i.e.* the transition happens no matter what the input set contains—including when it is empty. Also, we write conjunction as ','. Finally, when there is no output we omit the '/'.

[2]We have left *B* off here, but it is just as in Figure 1.

[3]We are dealing here with the case where no composition of charts is involved.

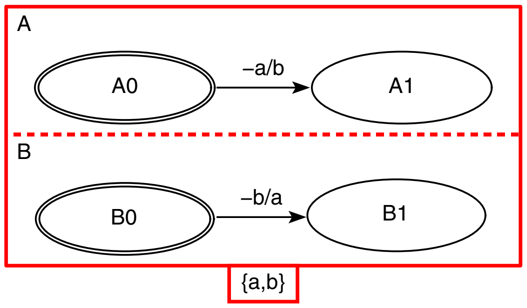


Figure 3. A chart where exactly one transition happens

4. Ensuring exactly one transition

One other basic chart that we shall require for the example we will introduce below is shown in Figure 3. In this chart, in the absence of any input, exactly one of the transitions will happen, *i.e.* either the one in chart *A* or the one in chart *B*, but not both and not neither. In this chart the triggers on the transitions are *negated*, so the transition in *A* happens only if *a* is not in the input, and similarly for the transition in *B*.

What happens (when we interpret the semantics) in the absence of any inputs is as follows. The transition in *A* is triggered (since *a* is not in the input), which causes the output *b* to be produced and fed back, which means that the transition in *B* cannot happen. Alternatively, the transition in *B* can be triggered if there are no inputs, which means that *a* is output and fed back, blocking the transition in *A* from happening. Furthermore, when the full formal semantics is consulted we also see that it cannot be the case that neither transition happens. So, exactly one of the transitions will happen.

5. An idiom: Synchronized Transitions

Here we look at a construct which has turned out to be useful when we used the charts on examples.

5.1 A vending machine problem

Consider the chart in Figure 4 (over the page). This is a chart based on an *event calculus* example given in [12]. It models a system consisting of a vending machine *Vending* and customers *Customer*1 and *Customer*2. The vending machine dispenses products which cost £2 and accepts a single £2 coin. Having been fed the appropriate money, it clunks internally and then dispenses the product for collection. The customer simply deposits a £2 coin and collects the product.

Note that the signals $l1$ and $l2$ mean that exactly one customer can move to their respective second state in a given step, so that they are never both about to put in their coins simultaneously.

Now imagine that the vending machine is in state $V2$ (in response to a first customer, who is now in their state $C12$ having put in their coin) when another customer walks up and puts in another coin. In the step in which the second customer deposits the coin and so moves to their state $C22$, the vending machine emits its *clunk* and moves to state $V3$. The first customer remains in their state $C12$.

In the next step, the vending machine emits the product (modelled by the signal *collect*) to which *both* customers can react so, in the real system being modelled, the question as to which customer gets the product arises. Of course, only the first is entitled, so we have a problem with our model. Synchronization will provide a solution.

5.2 Modelling synchronization

Using the properties of preemption and instantaneous feedback mentioned above, we can produce a system that can synchronize two transitions. The design ensures that the two transitions can only occur together, in a single step. It can never be the case that one of the two transitions occurs without the other also occurring. A simple example of this system is shown in figure 5 (over the page).

In both *Synch*1 and *Synch*2, the initial and final states are simple atomic ones. It is in the states $A2$ and $B2$ that the preemption and feedback occur.

If the chart *Synch*1 is in the state $A1$ and, receiving an a as input, makes the transition to the state $A2$, the subchart $A2$ becomes active, and begins in the state $A2$*Loop*. The only transition $A2$*Loop* can subsequently undergo is one that loops back to itself, whatever the input, emitting a $c1$ signal as output. Similarly, once *Synch*2 reaches $B2$, $B2$*Loop* becomes active, and begins to emit $c2$ signals.

So now imagine that at some time in the past a and b have each been input (perhaps at different steps), so we have *Synch*1 in $A2$, $A2$ in $A2$*Loop*, *Synch*2 in $B2$ and $B2$ in $B2$*Loop*. In the next step, $A2$ undergoes the loop transition and emits a $c1$. $B2$ also undergoes its loop transition and emits a $c2$. Note that $c1$ and $c2$ are fed-back so at this instant the input (due to feed-back) consists of $\{c1, c2\}$, which are inputs that trigger the transitions from $A2$ to $A3$ and from $B2$ to $B3$. By preemption, the $A2$-to-$A3$ and $B2$-to-$B3$ transitions take precedence over the loop transitions.

To summarize, in this single step, and instantaneously, the transitions in $A2$*Loop* and $B2$*Loop* are triggered, emitting $c1$ and $c2$, respectively, and this triggers the $A2$-to-$A3$ and $B2$-to-$B3$ transitions, so *Sync*1 moves to $A3$ and *Sync*2

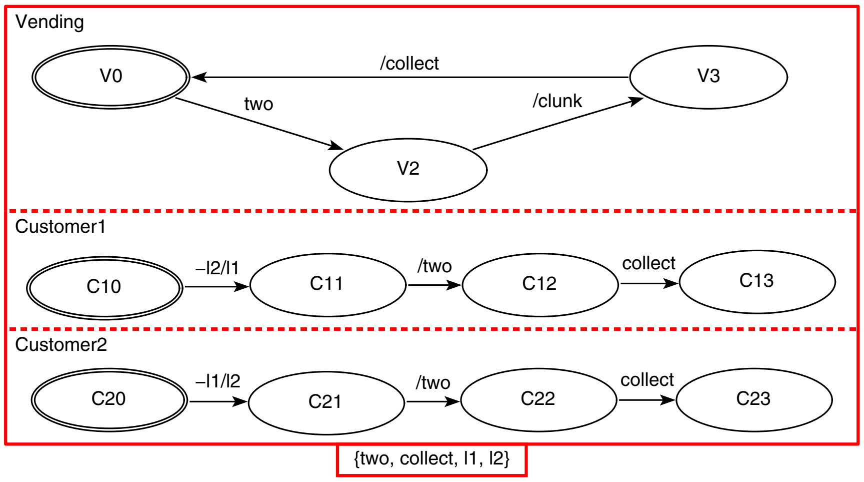


Figure 4. The vending machine example

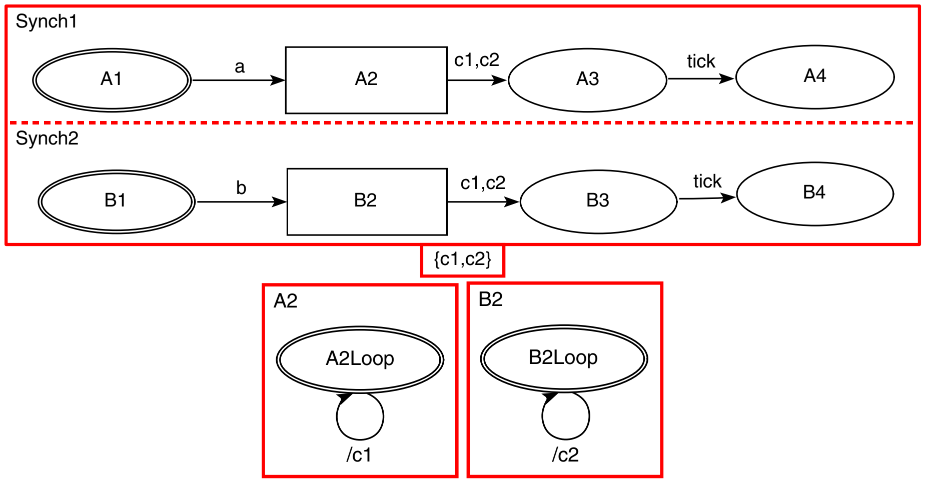


Figure 5. μ-charts demonstrating a synchronized transition

moves to $B3$, while the charts $A2$ and $B2$ become inactive.

5.3 A better vending machine

We can now model our vending machine system again and use synchronization to solve the contention problem we had previously.

The strategy is to force customers to synchronize with the vending machine when it is in $V2$. So, any customer not synchronized with the machine on this will not pass beyond $C11$ or $C21$ (depending on which customer we are dealing with) and so will not be waiting to collect a bar when the machine makes it available. Figure 6 (over the page) shows the chart for this.

This solution relies on the model embodying the asymmetry between *SynchVending* and the customers. For example, it makes no sense *in the intended model* for all three to synchronize together, *i.e.* it does not accord with the intended behaviour for them to be treated symmetrically. *SynchVending* has a different rôle from either customer.

Of course, in other situations we may wish to synchronize three (or more) charts so we must be able to do so. We have, so to speak, several dimensions along which synchronization can take place: either charts synchronize with each other in a symmetric fashion, or one chart plays the rôle of a vendor of a resource and the others play the rôles of customers for that resource, with the constraint that only one customer at a time can be served successfully. We will see this distinction being captured when we turn to devising a notation for synchronization, which we do next.

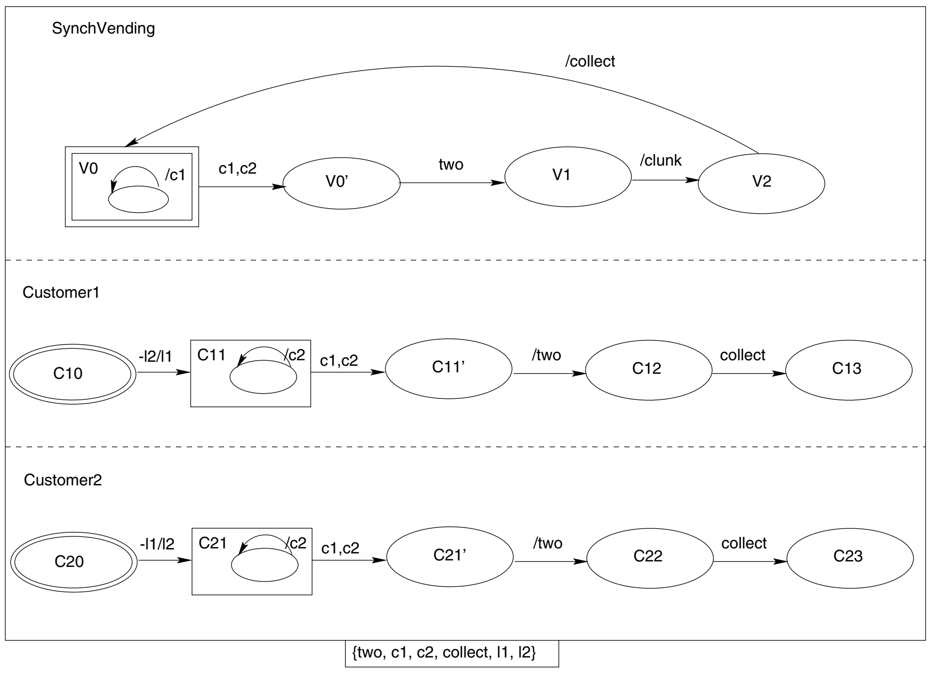


Figure 6. A better vending machine

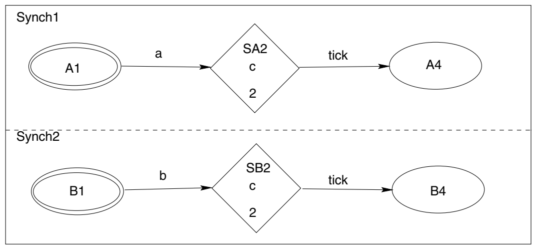


Figure 7. Notation for the simplest symmetric case

5.4 Some notation

Having identified this idiom we now want to develop some notation which makes it more convenient to use. The idea here will be to view the notation as a definitional equality—so whenever the notation is seen it can be replaced by a certain piece of μ-chart notation which uses just the basic language. Hence, any chart written in the extended language can be reduced to one in the original language, so no extra semantics is needed.

The simplest use of the synchronization idiom is the one demonstrated in Figure 5 where we have just two charts composed together in a symmetric fashion and synchronized on one pair of transitions (the ones labelled $c1, c2$ in that chart). This can clearly be generalized to as many charts as we wish.

In designing some notation for this we need to consider what information needs to be preserved in order that the extended notation can be replaced by exactly the right fragment of chart in the original, basic notation. The information we need for this simplest case is what the signals are that each non-atomic state emits (these are states $A2$ and $B2$ in this chart), which in this case are the signals $c1$ and $c2$. Once we have that we can replace the states $A2$ and $A3$ by a new sort of state—a *synchronizing state*—which we call $SA2$ in Figure 7, and similarly for $B2$ and $B3$.

Note that since the signals $c1$ and $c2$ are used only in the synchronization process the synchronization state needs to mention just their common suffix. We also add a number which shows how many charts are symmetrically synchronized: this is just for convenience when it comes to replacing such a state by its definition, and can equally well be calculated by counting-up the number of synchronization states which synchronize on the signal concerned. Finally, we omit the synchronizing signal c from the feed back box since such a signal is always understood to be fed back.

Now, when we look at the chart we understand that synchronization states with the same signal work together to give the desired behaviour.

So, whenever we see a chart like the one in Figure 7 we can calculate its meaning by replacing the new construct with the old construct that this example shows. Clearly, af-

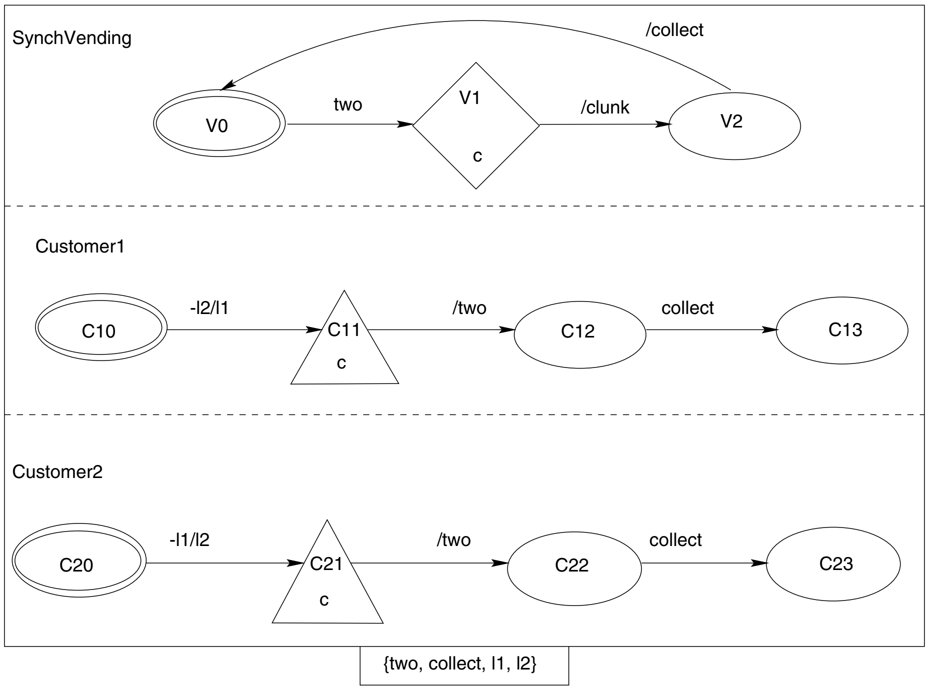


Figure 8. An asymmetric synchronizing composition

ter some experience, we will understand the meaning of the new construct well enough to write and read its meaning directly without having to construct the basic chart.

More thought has to go into the more complicated constructs like the one we have been dealing with in the vending machine. This is because we are dealing with three charts composed together and, as mentioned in the previous section, there are more degrees of freedom concerning the way in which they interact to synchronize.

With three charts the most straightforward possibility is for all three charts to synchronize symmetrically, and this is handled by generalizing the case by having a synchronization state in each chart which mentions all three synchronization signals (rather than just the two for our case above). This generalizes in the obvious way when we have *n* charts all synchronizing, as we shall see.

In our example (recall Figure 6) either $Customer1$ or $Customer2$ can synchronize with *SynchVending*, so this is not so simple a case because of the asymmetry between the vendor and the customers. To differentiate this case from the symmetric one we introduce another sort of synchronization state which denotes a customer rather than a vendor. Now, the vendor state synchronizes on either of the customer signals and the chart is now as given in Figure 8.

We can summarize all the above discussion by giving definitions of general cases.

In Figure 9 (over the page) we show how the i^{th} occurrence of *n* symmetric synchronizing states is defined.

In Figure 10 (over the page) we deal with the definition of the asymmetric case, which shows how the vendor and its $n-1$ customers are defined.

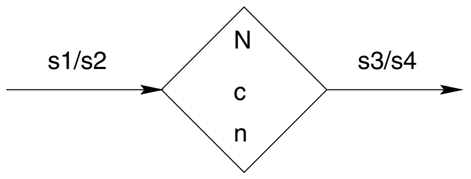
6. Conclusions

This paper set out to do two things: to give an example of incorporating an idiom into the language of μ-charts by definition and, more importantly since it has wide applicability and consequences, to argue for the design model of a simple language as a basis or kernel in which useful idioms can be defined.

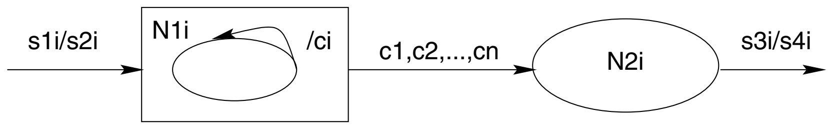
So, in this paper we have shown how a commonly recurring and useful construct, that of synchronization between subsystems, can be introduced to the basic language of μ-charts via definition.

In our work of designing charts for various systems we have identified further useful idioms—one concerns modelling a distributed storage mechanism where storage (or variables) can be shared across composed charts, and another concerns timers and delays which also turn out to be quite commonly required in designing reactive systems.

In each of these cases we are extending the language purely by definitional equality, which means that no new semantic constructs are needed to extend the language with the given idiom and, consequently, that all the intuitions



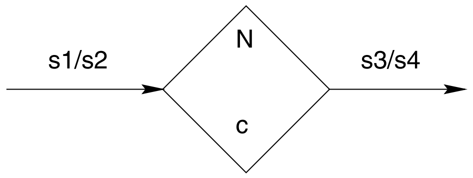
is defined to be



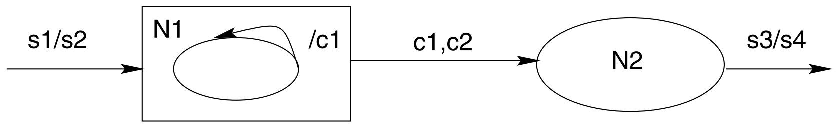
for i between 1 and n

Figure 9. General definition for the i^{th} symmetric synchronizing state

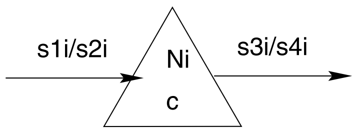
The vendor



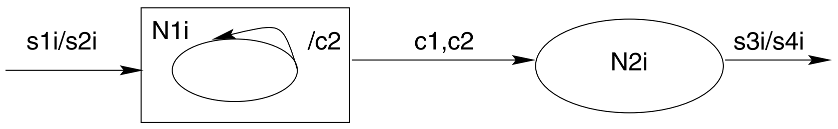
is defined to be



and the customer



is defined to be



for each of the customers with i= 2,...,n

Figure 10. General definition for the vendor and the i^{th} customer for asymmetric synchronizing states

about charts need not be modified when considering the extended language.

Further, any proof rules that we will develop for the basic language will induce derived rules about the new construct in a straightforward way.

One desirable goal in all this is to show what mechanisms that appear in, say, UML Statecharts, can be defined in the much simpler and so formally more tractable μ-charts.

Of course, other features of Statecharts are ruled-out from being merely definitional extension of μ-charts because they are semantically problematic, *e.g.* as mentioned before (and as pointed out by Scholz) the inter-level transitions fall into this category. However, rather than seeing this as a shortcoming of μ-charts we see it as a criticism of Statecharts—after all, the possibility of giving a simple, compositional semantics to a language is a touchstone of its quality.

Finally, and for all the reasons given above, this technique of building a small, simple language and then extending it by definition is one that we have used in the past to good effect (*e.g.* [4]) and we commend it as a design technique.

7. Acknowledgments

The authors would like to acknowledge funding from the New Zealand Government's New Economy Research Fund (NERF), administered by the Foundation for Research, Science and Technology (FRST) for the continued funding of this research.

References

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

[2] B. P. Douglas. UML Statecharts. Technical report, I-Logix Inc. downloaded from http://www.ilogix.com on 22/5/01.

[3] David Harel, Hagi Lachover, Amnon Naamad, Amir Pnueli, Michal Politi, Rivi Sherman, Aharon Shtull-Trauring, and Mark Trakhtenbrot. STATEMATE: A working environment for the development of complex reactive systems. *IEEE Transactions on Software Engineering*, 16(4):403–414, April 1990.

[4] M. C. Henson and S. Reeves. Investigating Z. *Journal of Logic and Computation*, 10(1):1–30, 2000.

[5] M. C. Henson and S. Reeves. Program development and specification refinement in the schema calculus. In *ZB 2000: formal specification and development in Z and B: first International Conference of B and Z Users, York, UK, August 29–September 2, 2000: proceedings*, volume 1878, pages 344–362. Springer-Verlag Inc., 2000.

[6] 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.

[7] Greg Reeve and Steve Reeves. μ-Charts and Z: Examples and extensions. In *Proceedings of APSEC 2000*, pages 258–265. IEEE Computer Society, 2000.

[8] 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, pages 255–276. Springer-Verlag, 2000.

[9] Greg Reeve and Steve Reeves. μ-Charts and Z: Hows, whys and wherefores. Technical Report 00/6, Department of Computer Science, University of Waikato, 2000.

[10] Mark Saaltink. The Z/EVES system. In J.P. Bowen, M. G. Hinchey, and D. Till, editors, *ZUM '97: The Z Formal Specification Notation. 10th International Conference of Z Users. Proceedings*, pages 72–85, Berlin, Germany, 3–4 April 1997. Springer-Verlag.

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

[12] W.J.Stoddart. The event calculus vsn 2. Technical Report SCM-96-1, University of Teeside, 1997.