An Expressive Real-Time CCS

Colin Fidge
Software Verification Research Centre
The University of Queensland
cjf@cs.uq.edu.au

John Žic
School of Computer Science & Engineering
The University of New South Wales
John.Zic@serg.cse.unsw.edu.au

Abstract. This paper describes a new 'real-time' process algebra that exhibits the advantages of both the 'constraint-oriented' and 'marker variable' specification styles. Its definition extends CCS, firstly with a basic notion of absolute time added to actions, and then with relative timing expressions which may refer to time markers.

1 Introduction

Although many proposals for real-time process algebras have been made, they generally suffer from overly complex semantics, especially where the concurrency operators are concerned, or rely on counter-intuitive priority-based concepts in order to achieve expressibility.

We define a new real-time process algebra which combines straightforward semantics with good expressive power. To do this we have combined the best features of two previously separate streams of real-time research, the 'constraint-oriented' [2, 1] and 'marker variable' [4] approaches. As a concrete illustration we describe an extended version of Milner's Calculus of Communicating Systems (CCS) [3].

2 Language definition

The language definition extends both the 'basic' and 'full' CCS languages. In the former case a simple definition of time is introduced via a single *absolute* time associated with all actions. Although conceptually adequate for expressing any desired real-time behaviour, this notation is too inconvenient for general use. We therefore then introduce marker variables and set-valued timing expressions to the full language so that concise and *relative* timing requirements can be stated.

2.1 Basic language

The basic language is identical to standard CCS [3, §2.4] with extra annotations on the prefix operator.

Let $\mathcal{A}$ be a set of action *names*; $\overline{\mathcal{A}}$ a set of *co-names*; $\mathcal{L} = \mathcal{A} \cup \overline{\mathcal{A}}$ the *labels*; $Act = \mathcal{L} \cup \{\tau\}$ the set of all possible *actions* including the *silent* action τ; f a *relabelling* function on actions; I an *indexing* set with $\{E_i \mid i \in I\}$ a family of expressions indexed by I.

Also let $\mathcal{T}$ be a set of *absolute time* values. If a discrete time model is desired $\mathcal{T}$ can be the set of natural numbers. For a continuous time model it is the non-negative reals. The following rules are independent of the time model adopted.

Let $\mathcal{E}$ be the set of *agent expressions*, defined as follows, where E, E_i are already in $\mathcal{E}$.

1. $\alpha @ t \stackrel{c}{\cdot} E$, a Prefix ($\alpha \in Act$; $t, c \in \mathcal{T}$)
2. $\sum_{i \in I} E_i$, a Summation
3. $E_1 \mid E_2$, a Composition

4. $E \setminus L$, a Restriction ($L \subseteq \mathcal{L}$)
5. $E[f]$, a Relabelling

We also assume the existence of a set of *agent constants* $\mathcal{K}$ each of which can be defined as follows, where $\mathcal{G}$ is the set of *agents*, i.e., agent expressions without free variables.

6. $A \stackrel{\mathrm{def}}{=} G$, agent constant Definition ($A \in \mathcal{K}, G \in \mathcal{G}$)

As usual the 'nil' agent $\mathbf{0}$ is defined as a choice between no alternatives [3, p.44]:

$$\mathbf{0} \stackrel{\mathrm{def}}{=} \textstyle\sum_{i \in \{\}} E_i \ .$$

The only unusual features are the absolute time t associated with each action, and the 'context' c associated with each prefix operator. The absolute time t is supplied by the specifier. It states that action α can occur at exactly time t only. Obviously this is very restrictive—in the full language far more powerful timing specification mechanisms are built from this simple basis, however.

The *context* c is used to support the definitions only. It defines the time at which the *causally* preceding action (if any) occurred and is used to ensure that time does not go backwards. It is not intended to be seen by the specifier. The context may therefore always be omitted in specifications, in which case a default value is assumed:

$$\alpha@t \cdot E = \alpha@t \stackrel{0}{\cdot} E \ .$$

The transitional semantics is closely based on standard CCS [3, §2.5] except that transition labels carry the absolute time at which each action occurred and the transition rule for prefix is extended.

A *labelled transition system* $(S, T, \{\stackrel{a}{\longrightarrow} \mid a \in T\})$ consists of a set S of *states* (taken to be $\mathcal{E}$), a set T of *transition labels* and a set of *transitions* $\stackrel{a}{\longrightarrow} \subseteq S \times S$ for each $a \in T$. Each transition label $a \in T$ is a pair (α, t) consisting of an action name $\alpha \in \mathit{Act}$, and an absolute time $t \in \mathcal{T}$ recording when the transition occurred. Transition rules for all basic CCS operators are shown in Figure 1.

The significant change is in **Act**. It states that action α, specified to occur at absolute time t, can do so only if t is greater than or equal to the context time c. In Figure 2, for instance, action c is specified to occur at time 4 but it cannot because its context states that the previous action occurred at time 5.

After an action is performed the subsequent behaviour E is updated in **Act** to record the fact that the most recent action occurred at time t. This is denoted $(E)^t$ and is defined as replacing the 'context' of the next prefix operator(s) in E with t. (This is sufficient because action prefix is the only operator that can directly perform an action.)

We can think of the absolute times in the transition rules as part of the action name. Thus, in Figure 3, action a performed at time 2 is distinct from action a at time 5.

Rule **Sum** tells us that time does not influence choices. This is because we view an agent expression as a non-constructive specification, not an executable implementation. Expression

$$a@5 \cdot \mathbf{0} + b@4 \cdot \mathbf{0}$$

thus *specifies* a system that *either* performs a at time 5 *or* action b at time 4. It does *not* state that only the earlier alternative is allowed, as illustrated by the three possible behaviours in Figure 3.

Rules $\mathbf{Com}_1$ and $\mathbf{Com}_2$ tell us that parallel agents do not interact when performing independent actions. In Figure 4, therefore, actions a and c occur entirely separately, in either order. If viewed as an interleaving of actions it thus appears that time goes backwards on the right-hand derivation. In fact, the times on these two transitions are *incomparable* in the partial ordering defined by causality. An anomaly arises only if we attempt to impose an *arbitrary* total order on independent actions. As noted by Fidge [2], the true, partial causal ordering never contradicts the temporal one or, as succinctly stated by Aceto and Murphy [1], such traces are "ill-timed but well-caused".

Furthermore, rule $\mathbf{Com}_3$ states that agents can interact only if they agree on the time at which the shared action occurs. Thus, in Figure 4, the shared action can occur at time 6 only, even though

$$\mathbf{Act} \qquad \frac{}{\alpha@t \overset{c}{\cdot} E \xrightarrow{(\alpha,t)} (E)^t} \qquad (t \geq c)$$

$$\mathbf{Sum}_j \qquad \frac{E_j \xrightarrow{a} E'_j}{\sum_{i \in I} E_i \xrightarrow{a} E'_j} \qquad (j \in I)$$

$$\mathbf{Com}_1 \qquad \frac{E \xrightarrow{a} E'}{E \mid F \xrightarrow{a} E' \mid F}$$

$$\mathbf{Com}_2 \qquad \frac{F \xrightarrow{a} F'}{E \mid F \xrightarrow{a} E \mid F'}$$

$$\mathbf{Com}_3 \qquad \frac{E \xrightarrow{(l,t)} E',\ F \xrightarrow{(\bar{l},t)} F'}{E \mid F \xrightarrow{(\tau,t)} E' \mid F'}$$

$$\mathbf{Res} \qquad \frac{E \xrightarrow{(\alpha,t)} E'}{E \setminus L \xrightarrow{(\alpha,t)} E' \setminus L} \qquad (\alpha, \overline{\alpha} \notin L)$$

$$\mathbf{Rel} \qquad \frac{E \xrightarrow{(\alpha,t)} E'}{E[f] \xrightarrow{(f(\alpha),t)} E'[f]}$$

$$\mathbf{Con} \qquad \frac{P \xrightarrow{a} P'}{A \xrightarrow{a} P'} \qquad (A \overset{\text{def}}{=} P)$$

Figure 1: Transition rules, where $\alpha \in Act$; $t, c \in \mathcal{T}$; $a \in T$; $E, F \in \mathcal{E}$; $G \in \mathcal{G}$; $l \in \mathcal{L}$; $L \subseteq \mathcal{L}$.

$$a@2 \cdot b@5 \cdot c@4 \cdot \mathbf{0}$$

$$\big\downarrow (a, 2)$$

$$b@5 \overset{2}{\cdot} c@4 \cdot \mathbf{0}$$

$$\big\downarrow (b, 5)$$

$$c@4 \overset{5}{\cdot} \mathbf{0}$$

Figure 2: Complete derivation tree showing that time cannot go backwards in prefixes.

the first agent is also prepared to perform it at time 7. If there is *no* time at which two agents agree to interact then the action is not possible. Clearly the semantics linking timing behaviour and concurrency is very simple in this model. Furthermore, the approach accords well with the 'constraint-oriented' specification style.

The bisimulation semantics of our language is very simple. The only significant change needed to standard CCS semantics is to replace the occurrences of action-name transition labels with name/time pairs in the definitions.

The equational laws of our revised language differ significantly from those of standard CCS [3, ch.3] in only two respects. The 'τ laws' need extra side-conditions to ensure that 'timed' silent actions do not block progress, and the 'expansion law' is no longer applicable in a true concurrency context (although a similar law can be modelled [1]).

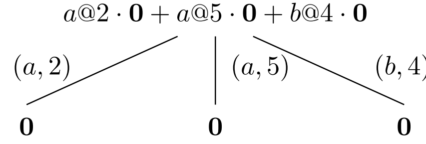

Figure 3: Derivation tree showing distinctness of timed actions and time-independent choice.

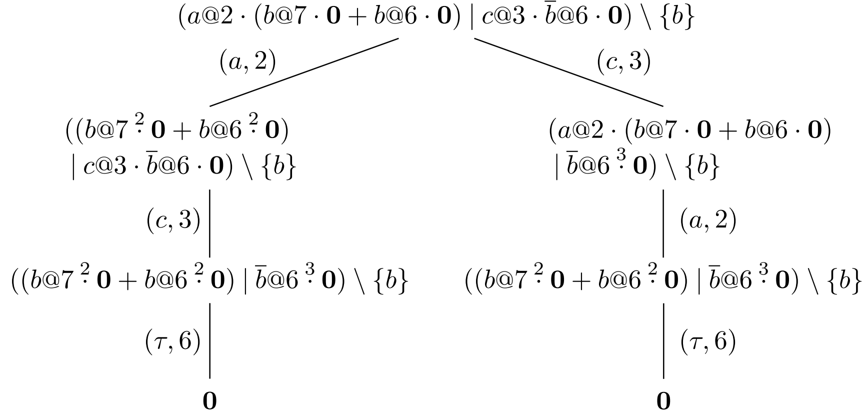

Figure 4: Derivation tree showing independence of parallel actions and the need to agree on times to interact.

2.2 The full language

Just as Milner extends basic CCS with a more expressive 'value-passing' notation, itself defined in terms of the basic language, we extend the primitive timing notion presented above with a much more expressive notation using marker variables and time sets.

Our full real-time language is an extension to Milner's value-passing calculus [3, §2.8], and can thus take advantage of CCS variables in timing expressions. Let $\mathcal{V}$ denote the set of CCS *value variable* names. We introduce a distinct set $\mathcal{M}$ of *marker variable* names which are used to hold absolute times. We also introduce $\mathcal{R}$ as the set of *non-empty* sets of absolute times from $\mathcal{T}$.

Let $\mathcal{C}$ be the finite functions from marker variables to absolute times. Thus some function $C \in \mathcal{C}$ defines a mapping from zero or more marker variable names to absolute times. Such functions are denoted $\{m \mapsto 1, n \mapsto 4, \ldots\}$ when variable m maps to 1, n maps to 4, etc. Where our basic language had 'contexts' consisting of a single time value c, the full language has contexts consisting of pairs $(c, C) \in \mathcal{T} \times \mathcal{C}$, i.e., an absolute time and a marker variable mapping. The absolute time defines when the causally preceding action occurred, as before. The mapping defines the values associated with previously encountered marker variables, if any.

Syntactically the major difference between the basic and full languages is a further extension to the prefix operator.

1. $\alpha@T\!: M \overset{(c,C)}{\cdot} E$, a Prefix ($\alpha \in Act;\ (c, C) \in \mathcal{T} \times \mathcal{C};\ T \in \mathcal{R};\ M \subseteq \mathcal{M}$)

Its context is extended as explained above. It also allows the action to be followed by a time-set-valued expression $T \in \mathcal{R}$, thus stating that this action may occur at any one of the times in T. T can be expressed using conventional set-comprehension notation, with free variables drawn from $\mathcal{M}$ and $\mathcal{V}$ as long as they are defined in C and the surrounding CCS variable scope, respectively. Furthermore the action can be followed by a set of marker variable names M. When (and if) the action occurs each of the variables in M will be assigned the value of the absolute time at which the action occurred, for use in subsequent timing expressions. (We also assume, but have not shown above, that action names in prefixes can be followed with message-passing parameters, as in value-passing CCS [3, p.55].)

	F	$\widehat{F}$
1.	$a@T\colon M \overset{(c,C)}{\cdot} E$	$a@T\{\vec{x}/\vec{v}\}\colon M \overset{(c,C)}{\cdot} E$ (with $\widehat{\ }$ over $T\{\vec{x}/\vec{v}\}$)
2.	$a@\{t_1 \dots t_n\}\colon M \overset{(c,C)}{\cdot} E$	$\sum_{\{t:t_1\dots t_n \mid t \geq c\}} a@t\colon \widehat{M \overset{(c,C)}{\cdot} E}$
3.	$a@t\colon M \overset{(c,C)}{\cdot} E$	$a@t \overset{c}{\cdot} (E)^{(t, C \oplus \widehat{\{m:M \cdot m \mapsto t\}})}$

Figure 5: Translations from full to basic prefix operators, where $a \in Act$; $T \in \mathcal{R}$; $M \subseteq \mathcal{M}$; $(c, C) \in \mathcal{T} \times \mathcal{C}$; $t, t_1, \dots, t_n \in \mathcal{T}$; $\vec{x} \subseteq \mathcal{V} \cup \mathcal{M}$; $\vec{v} \subseteq \mathcal{T}$.

A number of syntactic conveniences are used. As in the basic language, the contexts are not usually explicitly supplied by the user, so a simple default is assumed:

$$\alpha@T\colon M \cdot E = \alpha@T\colon M \overset{(0,\{\})}{\cdot} E \; .$$

We omit the set brackets for singleton sets, i.e.,

$$\alpha@t\colon m \overset{(c,C)}{\cdot} E = \alpha@\{t\}\colon \{m\} \overset{(c,C)}{\cdot} E \; ,$$

where $t \in \mathcal{T}$ and $m \in \mathcal{M}$. Also empty marker variable sets may be omitted:

$$\alpha@T \overset{(c,C)}{\cdot} E = \alpha@T\colon \{\} \overset{(c,C)}{\cdot} E \; .$$

Finally, timing expressions are made optional by assuming the least-defined default:

$$\alpha\colon M \overset{(c,C)}{\cdot} E = \alpha@\{t : \mathcal{T} | t \geq c\}\colon M \overset{(c,C)}{\cdot} E$$

(the earliest time at which α can occur is c because this is the time at which the preceding action happened).

Milner lets agents F in the value-passing calculus be rewritten in basic CCS via a simple translation function, denoted $\widehat{F}$ [3, p.56]. We extend this translation function further, as shown by the rules in Figure 5 for the new prefix operator. It allows expressions in the full language, with marker variables and arbitrary timing expressions, to be routinely re-expressed in the basic language, with simple absolute times only. Trivial rules for operators other than prefix allow the translation function to be recursively applied through an entire agent expression [3, p.56].

Rule 1 allows timing expressions to be eliminated, replacing them with the sets of absolute times they denote. It assumes that all elements of the set of free variables $\vec{x}$ in timing expression T are either defined CCS value variables from $\mathcal{V}$, or are marker variables from $\mathcal{M}$ that have a value defined in the current context C. Notation $T\{\vec{x}/\vec{v}\}$ then represents replacement of all free occurences of elements of $\vec{x}$ in T by the corresponding CCS 'value' or marker variable values, respectively. In the third node of Figure 6, for instance, the value of free variable m in expression $\{m + 1 \dots m + 3\}$ is bound by its appearance as a marker variable defined in context $(4, \{m \mapsto 2\})$, so it is replaced in the set expression to yield the set of absolute times $\{3, 4, 5\}$ (of which only the second two times are possible since the preceding action occurred at time 4).

Rule 2 then allows sets of absolute times to be eliminated by re-expressing them as a choice between the action occurring at each time in the set. This is a very powerful property, not enjoyed by other real-time process algebras. It is made possible only by our rejection of 'time-deterministic' choices [2] and is a major contributor to the simplicity of our semantics. Thus the third node in Figure 6, once reduced to

$$c@\{3, 4, 5\} \overset{(4,\{m \mapsto 2\})}{\cdot} \mathbf{0} \; ,$$

$$a@2\colon m \cdot b@4 \cdot c@\{m+1 \dots m+3\} \cdot \mathbf{0}$$

$(a, 2)$

$$b@4 \overset{(2,\{m \mapsto 2\})}{\cdot} c@\{m+1 \dots m+3\} \cdot \mathbf{0}$$

$(b, 4)$

$$c@\{m+1 \dots m+3\} \overset{(4,\{m \mapsto 2\})}{\cdot} \mathbf{0}$$

$(c, 4)$ $\quad$ $(c, 5)$

$$\mathbf{0} \qquad \mathbf{0}$$

Figure 6: Derivation tree showing the use of a marker variable to define relative timing and time-based choices.

$$a@2\colon m \cdot (b@4\colon n \cdot c@m+n \cdot \mathbf{0} \mid d@7\colon n \cdot e@m+n \cdot \mathbf{0})$$

$(a, 2)$

$$(b@4\colon n \overset{(2,\{m \mapsto 2\})}{\cdot} c@m+n \cdot \mathbf{0} \mid d@7\colon n \overset{(2,\{m \mapsto 2\})}{\cdot} e@m+n \cdot \mathbf{0})$$

$(b, 4)$ $\quad$ $(d, 7)$

$$(c@m+n \overset{(4,\{m \mapsto 2, n \mapsto 4\})}{\cdot} \mathbf{0} \mid d@7\colon n \overset{(2,\{m \mapsto 2\})}{\cdot} e@m+n \cdot \mathbf{0}) \qquad (b@4\colon n \overset{(2,\{m \mapsto 2\})}{\cdot} c@m+n \cdot \mathbf{0} \mid e@m+n \overset{(2,\{m \mapsto 2, n \mapsto 7\})}{\cdot} \mathbf{0})$$

$(d, 7)$ $\quad$ $(c, 6)$ $\quad$ $(b, 4)$ $\quad$ $(e, 9)$

$$(c@m+n \overset{(4,\{m \mapsto 2, n \mapsto 4\})}{\cdot} \mathbf{0} \mid e@m+n \overset{(2,\{m \mapsto 2, n \mapsto 7\})}{\cdot} \mathbf{0}) \qquad (c@m+n \overset{(2,\{m \mapsto 2, n \mapsto 4\})}{\cdot} \mathbf{0} \mid e@m+n \overset{(2,\{m \mapsto 2, n \mapsto 7\})}{\cdot} \mathbf{0})$$

$$(\mathbf{0} \mid d@7\colon n \overset{(2,\{m \mapsto 2\})}{\cdot} e@m+n \cdot \mathbf{0}) \qquad (b@4\colon n \overset{(2,\{m \mapsto 2\})}{\cdot} c@m+n \cdot \mathbf{0} \mid \mathbf{0})$$

Figure 7: Partial derivation tree showing inherited and independent marker name spaces.

can be re-expressed as

$$c@3 \overset{(4,\{m \mapsto 2\})}{\cdot} \mathbf{0} + c@4 \overset{(4,\{m \mapsto 2\})}{\cdot} \mathbf{0} + c@5 \overset{(4,\{m \mapsto 2\})}{\cdot} \mathbf{0}\ .$$

Rule 3 allows marker variable declarations to be removed, thus completing the translation of full prefix operators to basic ones. When a prefix has been reduced to a singleton timing specification t any marker variables $m \in M$ declared in the prefix will be mapped to this time *if* the action occurs. Thus the context of the subsequent action E is updated so that it records the occurrence of a at time t, and inherits context C updated to reflect the mapping of all markers in M to t. This is denoted using function 'over-riding' as $C \oplus \{m : M \cdot m \mapsto t\}$ which will add a new mapping for any m not in C or replace the existing mapping for an m already in C.

Importantly, the extended contexts are still maintained separately for causally independent actions. In Figure 7 the two parallel agents both initially inherit the value of marker m. Following their first actions they set n to 4 and 7, respectively. These two 'instances' of n belong to different marker name spaces and are thus distinct. Therefore, even though actions c and e both occur at relative time '$m + n$', they actually happen at different absolute times, due to the different contexts.

3 Case study: repeater system

Consider a 'repeater' that must read numbers from a memory-mapped i/o location without missing, duplicating or mis-reading them. Numbers appear periodically, every 50 time units, but are reliably readable for only 45 time units. There is a 5 time unit transition between each number, during which the value in the memory location is ill-defined. Thus the first value can be read reliably from time 0 to 45, inclusive, the second from 50 to 95, the third from 100 to 145, and so on.

Firstly we define a source S of such numbers as follows. It is an agent that represents the situation where an increasing series of numbers, starting at 0, appears at 'location' s, with timing as outlined above. It does not constrain how the numbers are 'read', however. It allows each number to be read several times, or not at all. Furthermore, it allows reads to occur during the transition period, in which case the error value '-1' is sent.

$$\begin{aligned} S &= S_0 \\ S_n &= \sum_{i \in n \ldots \infty} \Big(\overline{s}(i) @ \{t | i * 50 \leq t \leq i * 50 + 45\} \cdot S_i \\ &\qquad + \overline{s}(-1) @ \{t | i * 50 + 45 < t < (i+1) * 50\} \cdot S_i \Big) \end{aligned}$$

Agent S_n represents the situation where the last value read was in the nth period. The next 'read' may then occur in the ith period, where i is at least as great as n. Thus a number of periods may go by with no read occurring, or a number of reads may occur in the same period. When the next read *does* occur there are two possibilities. Either the read occurs at the correct time, i.e., within 45 time units of the beginning of period i, in which case the number of this period is sent on channel s, or the read occurs at the wrong time, in the last 5 time units of period i, in which case the error value -1 is sent.

Our goal is to define a repeater agent R which, when placed in such an environment, i.e.,

$$(S \mid R) \setminus \{s\}$$

will read and repeat values from S, on some channel r, so that the sequence $0, 1, 2, \ldots$ is correctly produced, with no duplicated, missing or 'error' values.

As a first attempt consider the following agent:

$$R1 = s(x) \cdot \overline{r}(x) \cdot R1 \ .$$

It reads values from s into variable x and repeats them on channel r. However it may read at *any* time, and thus may miss values entirely or read error values. Also $\overline{r}$ may occur any time after the preceding s, since no timing requirement is specified, so even if a value is correctly read it may take so long to repeat it that subsequent values are missed.

Let us use a marker variable m to correct the second of these problems:

$$R2 = s(x){:}\, m \cdot \overline{r}(x) @ \{t | t < m + 5\} \cdot R2 \ .$$

This agent guarantees that $\overline{r}$ will occur within 5 time units of the preceding action s. Thus, if the read was successful, i.e., occurred within the first 45 time units of the period, the value will be repeated before the next number becomes available. $R2$ may still fail to read the values correctly, however. It can still miss, duplicate or mis-read numbers.

As a further improvement we take advantage of the constraint-oriented specification style to ensure that s actions never occur 'between' values:

$$R3 = s(x) @ \{t | \exists n \geq 0 \cdot n * 50 \leq t \leq n * 50 + 45\}{:}\, m \cdot \overline{r}(x) @ \{t | t < m + 5\} \cdot R3 \ .$$

This ensures that s actions can occur only when a 'good' value is available. It thus constrains agent S so that its second alternative is never possible. But, although $R3$ will never read the error value, it can still skip numbers or read them more than once.

To complete the example we further constrain the repeater so that it will perform a read exactly once in each period:

$$
\begin{aligned}
R4 &= R4_0 \\
R4_n &= s(x)@\{t|n*50 \le t \le n*50+45\}\colon m \cdot \overline{r}(x)@\{t|t<m+5\} \cdot R4_{n+1}\ .
\end{aligned}
$$

Here the CCS value-passing variable n acts like a counter for time periods. It is incremented after each (successful) read so that the next read *must* occur in the subsequent period. $R4$ thus further constrains S.

We have now fully *specified* a solution to the repeater problem. It is still far from being an implementation, however. As a final illustration of the way timed agents can constrain one another we refine our solution to a closer approximation of a particular implementation. A key feature of agent $R4$ is its ability to decide *when* to perform a s action. In practice this would be implemented via a hardware clock. Therefore, let us define a satisfactory 'implementation' T, which derives its timing information from a clock C:

$$
\begin{aligned}
R5 &= (C \mid T) \setminus \{a\} \\
C &= \overline{a}@0\colon n \cdot C1 \\
C1 &= \overline{a}@(n+50)\colon n \cdot C1 \\
T &= a\colon m \cdot s(x)@(m+2) \cdot \overline{r}(x)@(m+4) \cdot T\ .
\end{aligned}
$$

Agent C represents a clock whose 'alarm' action a occurs every 50 time units, starting at time 0. Action $\overline{a}$ in agent $C1$ is defined to occur 50 time units since its causal predecessor, via marker n, and also defines a new value for marker n, which will be used in the *next* iteration. The repeater is then easily implemented as T by just waiting for each alarm, and reading and repeating the value in s. We have assumed that each of these actions takes exactly 2 time units in the actual implementation.

At this level of abstraction we have still made a powerful assumption that C is precisely synchronised with the incoming data stream. Indeed, it is interesting to note that C can also be thought of as implementing a 'data ready' signal associated with the availability of each character, merely by taking advantage of the associativity of the composition operator, i.e.,

$$((S \mid C) \mid T) \setminus \{a, s\}\ .$$

Thus this abstract model describes the behaviour of two distinct implementations.

4 Conclusion

A combination of 'constraint-oriented' and 'marker variable' specification techniques can be used to create a real-time process algebra with straightforward semantics and good expressive power.

References

[1] L. Aceto and D. Murphy. On the ill-timed but well-caused. In E. Best, editor, *Concur'93*, volume 715 of *Lecture Notes in Computer Science*, pages 97–111. Springer-Verlag, 1993.

[2] C. J. Fidge. A constraint-oriented real-time process calculus. In M. Diaz and R. Groz, editors, *Formal Description Techniques V (FORTE'92)*, pages 363–378. North-Holland, 1993.

[3] R. Milner. *Communication and Concurrency.* Prentice-Hall, 1989.

[4] J. Žic. Specifying a time constrained buffer in Timed CSP and CSP+T. *ACM Transactions on Programming Languages and Systems*, November 1994.