SCS&E Report 9407
March, 1994

Time Constrained Buffer Specifications in CSP+T and Timed CSP

John J. Zic

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING
THE UNIVERSITY OF NEW SOUTH WALES

Abstract

A finite buffer with time constraints on the rate of accepting inputs, producing outputs and message latency is specified using both Timed CSP and a new real-time specification language, CSP+T. CSP+T adds expressive power to some of the sequential aspects of CSP and allows the description of complex event timings from within a single sequential process. On the other hand, Timed CSP encourages event timing descriptions to be built up in a constraint-oriented manner with the parallel composition of several processes. Although these represent two complementary specification styles, both provide valuable insights into specification of complex event timings.

E-mail:
johnz@cse.unsw.edu.au

1 Introduction

There has been considerable effort recently in extending Hoare's CSP [7] and Milner's CCS [8, 9] to allow formal reasoning about real-time systems. Examples of such systems are commonly found in communication protocols where the response to a message is required before the message becomes obsolete, or where message outputs need to be spaced so as to avoid overflow conditions at the receiving end.

The author proposed some informal time (and probability) extensions to CSP in [13] where a special *DELAY* process allowed temporal separation between any two successive events. At approximately the same time, Reed and Roscoe [11] introduced a similar special process *WAIT* into CSP and provided a complete semantics (based on Timed Failures) for their Timed CSP language. Gerber, Lee and Zwarico [6] introduced a *timed action* operation (effectively, a time was associated with each prefix operation) to temporally separate adjacent events into their extended CSP and provided an Timed Acceptance semantics for the language. Quemada and Fernandez [10] proposed an extension to the LOTOS specification language [2] by associating an enabling interval with each event. This time interval represents the time over which a process may engage the event.

There are difficulties in describing and specifying complex system timings using this type of construction (where an event timing is defined solely by its immediate predecessor). Complex timings and the ability to define the future behaviour of a system will inevitably rely on a set (with more than one element) of preceding events in the process' execution. Furthermore, these events may have occurred a long time in the past execution of the process. To address these problems, the author in [14] proposed an extended CSP (called CSP+T) which associated an enabling interval with each event, and also allowed this interval to be expressed as a function of one or more *marker events.*

It will be seen that the CSP+T approach differs fundamentally from the Timed CSP solution. Timed CSP captures timing constraints by the parallel composition of a set of processes, each of which describes a specific timing constraint. These timing constraints are viewed as representing timed refinements of the system, and facilitate the specification and proof of system timings. However, the algebraic manipulation of complex timing requirements from a parallel composed system into (essentially) a simpler sequential system may, in some cases, be impossible without extending Timed CSP to allow a way of recording and using the time at which specific events occurred in the system's execution.

On the other hand, CSP+T describes timing constraints of sequential processes in sequential terms as much as possible, reducing the need for the addition of any parallel processes to express timing constraints. Further, the additional expressive power of the language now allows some complex parallel systems to be rendered as sequential processes which could not be done with only interevent timing constructs such as *DELAY*.

This paper is organised as follows. First, we present the problem, which is the modelling of a store-and-forward communication system with specific quality of service requirements. This is done by representing the system as a finite length buffer with appropriate input, transit and output timing requirements. Second, the CSP extensions for CSP+T are presented. Third, we develop a solution in terms of the existing Timed CSP algebra. Finally, an alternative solution using the CSP+T notation is presented.

2 The problem

A store-and-forward communication network may be abstractly represented by a finite length message *buffer*. Messages injected into the network at a particular node appear some time later at another node in the same order as they were sent.

Besides this most abstract functionality of order preservation, a communication system may also need to provide end users with some real-time performance. For example, maximum and average message delays, throughput, reliability, probability of loss of a message, and other client requirements may be important [1, 4].

This paper attempts to describe a communication system with the following characteristics given by a (somewhat) naive client:

- up to *128* messages in transit at any time,
- message latency in the range [*2*, *5*] time units,
- message input rate set to *1* message per time unit, and
- message output rate of *1* message per two time units.

As these timing constraints stand, there will be problems with any implementation. Firstly, the fact that the output message rate is half that of the message input rate means that the any finite sized buffer will eventually either overflow or not meet the message transit delay requirements. Secondly, the output must simultaneously satisfy the transit delay requirements as well as the output timing requirement for a given (fixed) input timing. Eventually all of these conditions cannot be simultaneously satisfied, and the system fails (in some manner).

3 A brief description of the extensions

The CSP+T *syntax* is a superset of the basic untimed deterministic CSP syntax presented by Hoare [7]. The fundamental changes to the untimed algebra are that:

- A new event, $\star$, is introduced to denote process instantiation into both the algebra and traces models.
- A new event operator $\bowtie$ is introduced, which is used in conjunction with a variable to record the time at which an event occurs. These times are taken from the set of positive Real numbers, with successive event times forming a monotonic nondecreasing sequence. We allow any number of successive events in a single process trace to have the same time. It is the designer's responsibility to explicitly mention any limitations on the number of computations done over any particular time period (including zero).
- Each event now has a time interval associated with it. This time interval represents a choice at which times the event must occur. These intervals are continuous, and are usually expressed *relative* to a set of events.
- Only deterministic timed processes can be described in the algebra to date. A semantics for processes making internal or nondeterministic choice is currently being formulated.

The major change to the traces model is that events are now *pairs*, $t.e$, where t is the global *absolute* time at which event e is observed. This is the same as Timed CSP traces model with the addition of the new instantiation event.

We consider each of the above items in turn.

3.1 Process instantiation event

Each system of process definitions requires that it is *instantiated* before it can execute. As such, a special process instantiation event denoted by "$\star$" is introduced into the algebra (and in the corresponding traces model). This event is unique, in that it must be associated with a unique, global time. It represents the global time at which the system of processes may start.

Consider a process which engages in a single a event and then breaks. This process may be defined in untimed CSP by $P = a \longrightarrow \mathit{Stop}$. The traces set of this process is

$$\{\langle\rangle, \langle a \rangle\}.$$

In CSP+T, we prefix this process with the instantiation event in order to allow it to execute. If the timed version of P (called P'), is instantiated at time 1, then we have

$$P' = 1.\star \longrightarrow a \longrightarrow Stop,$$

and the (timed) traces set of P' is

$$\{\langle\rangle, \langle 1.\star\rangle, \langle 1.\star, s.a\rangle\}$$

where $s \in [1, \infty)$. Notice that the a event occurs only once in the interval $[1, \infty)$ for any particular process execution. Secondly, the times in traces descriptions are *absolute*, whereas the time intervals in the process description are *relative*.

3.2 Time capture operator

A new event operator $\bowtie$ is introduced so that writing $ev \bowtie v$ means that the time at which the event ev is observed in a process execution is recorded in the variable v. So, for example, if we have a process instantiated at time 1 which behaves as $1.\star \longrightarrow a \bowtie v \longrightarrow Stop$ then the variable v holds the *global* time at which the a event occurs. In this case, the time at which the a event occurs (and hence, the time recorded in the variable v) is going to be greater than or equal to the process instantiation time of 1.

All variables associated with the time capture operator (*marker variables*) are available as process global quantities, provided that the associated event is also globally observable. If the event is the subject of a restriction or abstraction operation, then the marker variable scope is similarly restricted.

Finally, marker variables may be initialised by using the time capture operator on any event which has a well defined, global time associated with it. Typically, this is the process instantiation event. For example,

$$1.\star \bowtie \{u, v, x, y\} \longrightarrow a \bowtie v \longrightarrow Stop$$

initialises variables u, v, x and y to the process instantiation time of 1. The variable v is then used to reference the time at which the a event occurs. The reference to the instantiation event (as it is initialised) is replaced by the reference to the time at which the a event is observed.

3.3 Event enabling interval

Each event in CSP+T is associated with a time interval, whether or not it is explicitly used. This represents the time over which the current event is regarded as being available to the process and its environment, relative to some preceding event from its current execution. In effect, the event enabling interval may be regarded as equivalent to a deterministic choice construction, with event labels drawn from the dense (Real number) interval defining the times at which the event may occur. Further, if the event has not occurred by the end of its enabling interval, the process effectively withdraws its offer to engage in that event. The process behaves as *Stop* if it cannot engage in an alternative event (after the "expiration" of the current event).

For example, an event a which has an enabling time interval of $[0, 1]$ is written as $[0, 1].a$ and must occur only *once* in the specified time interval (between 0 and 1 time units after a's preceding event). Another event $[1, 1].b$ must occur precisely at one time unit immediately after its preceding event. An example process which uses event enabling intervals is $0.\star \longrightarrow [1, 2].a \longrightarrow Stop$. This is a process which will engage in a single a event only between 1 and 2 time units since it was instantiated (at time 0), and then break.

If a time interval is not explicitly mentioned with an event, the least defined interval $[0, \infty)$ is assumed. That is, the event associated with this interval is allowed to occur at any time since the immediately preceding event.

These intervals are defined in terms of functions over a set (including the empty set) of marker variables. When there are no marker variables referenced, then the enabling interval is defined for the immediately preceding event (as above). More typically, however, the expression is given in terms of one or more marker variables. For example, a clock which is instantiated at time *0* and then *tick*s once every time unit after that may be defined by

$$\begin{aligned} RealClk &\mathrel{\widehat{=}} 0.\star \bowtie v \longrightarrow TimedClk \\ TimedClk &= \mu X \bullet E.tick \bowtie v \longrightarrow X \end{aligned}$$

where $E = \{s | s = rel(1, v)\}$ and the *rel* function is defined as follows. If the preceding (reference or marker) event occurs at time t_0 then $rel(x, v)$ denotes $x + v - t_0$. This convention allows us to combine conditions expressed relative to several different marker events in the definition of a single enabling interval.

The use of these enabling intervals presents two major sets of questions under parallel and sequential composition. For example, how processes that use these enabling intervals may be composed with each other (in parallel). Related to this, what other influences may determine when an event occurs within its enabling interval?

We consider each of the above points in the following discussion.

3.3.1 Process synchronisation

Consider the two simple processes $P = E_1.a \longrightarrow P$ and $Q = E_2.a \longrightarrow Q$, with identical untimed alphabets. Now suppose we compose these two in parallel as $0.\star \longrightarrow P \parallel Q$. The semantics of this composition are dependent on whether the values taken by E_1 and E_2 are identical, partially overlapping, or disjoint. If two intervals are identical, the composite process engages in a single synchronised a event within the interval. There are two possible behaviours for the parallel composition when the intervals become partially overlapping. We may choose to have the processes engage independently on events in the events outside of the intersection of the intervals E_1 and E_2, and synchronised within the intersection. Alternatively, we could completely disallow any events to be engaged that lie outside the intersection of the two enabling intervals. This latter case was selected in the original language design because it offers a simpler semantics.

Summarising:

- If $E_1 = E_2$ the processes synchronise on the a event once during this interval.
- If $E_1 \neq E_2 \wedge E_1 \cap E_2 \neq \{\}$ then the processes only synchronise during the interval $E_1 \cap E_2$. The system withdraws offers of engaging in events outside of this time.
- $E_1 \cap E_2 = \{\}$ then there are no times at which the processes may synchronise, and the composition behaves as *Stop*.

Additionally, we observe the following about synchronisation between processes:

- If the process' untimed alphabets are disjoint, the event enabling intervals play no direct role except to note that the resulting interleavings must be ordered so that the sequence of times in any timed trace are monotonic non-decreasing. For convenience, call this property **mndt**.
- If the untimed alphabets are partially disjoint, synchronisation depends both on the set of events which are in the intersection of the untimed alphabets and consideration of the event enabling intervals. Events outside of the intersection set (of the untimed alphabet) lead to interleavings possessing the **mndt** property.

- Explicit communication between processes via a channel may occur only if the sender and receiver processes have enabling intervals which intersect. For example,

$$(E_1.c!v \longrightarrow P) \parallel (E_2.c?x \longrightarrow Q(x))$$

will lead to a communication event $c.v$ in the interval defined by $E_1 \cap E_2$. If E_1 and E_2 are disjoint, and there are no other events offered, then the communication fails and the system stops. By setting the enabling interval on reception to $[0, \infty)$ and the sending process to E, then the system timing behaviour is set by the sending process and the communication occurs during E.

3.3.2 Sequential concerns

Prefix Consider the process

$$P = a \bowtie v \longrightarrow b \longrightarrow E.c \longrightarrow Stop,$$

where $E = \{t | rel(3, v) \leq t \leq rel(5, v)\}$. Since the c event must occur at any time from 3 time units to 5 time units since the occurance of the a event, the time at which the b event occurs must also be less than or equal to the maximum of c's enabling interval. If this is not the case, then the process breaks immediately after engaging the b, since there is no way that c could be engaged by the process. Hence this process has two separate behaviours which are dependent upon the time at which the b event occurs relative to the c event's enabling interval. Let $\lceil I \rceil$ represent the upper bound of an interval I, then process P may be rewritten as:

$$\begin{array}{lll} P = & a \bowtie v \longrightarrow & [0, \lceil E \rceil].b \longrightarrow E.c \longrightarrow Stop \\ & & \Box \\ & & (\lceil E \rceil, \infty).b \longrightarrow Stop \end{array}$$

Although it is possible to transform the generalisations of this case, such constructions should be avoided. As a general design principle, the timing intervals of a purely sequential process (consisting of prefix and choice operations) should be such that the entire expression does not abort due to enabling intervals alone. Any deliberate expression abortion will be due to other causes (such as parallel composed processes or explicitly designed timing exceptions).

Choice The prefix operation is regarded as a base case of the more general deterministic choice (or menu choice) operation. Choice in the CSP+T algebra selects from a finite subset of events from the untimed alphabet. However, as each event is associated with an enabling interval, this choice is between a possibly infinite set of timed events.

Choice between a set of events with disjoint enabling intervals is made according to the natural time order. That is, a given choice set at one point in time reduces to a smaller subset as time progresses. Events "expire" and are removed from the choice set. For example, assume that the original choice set (at time 0) is $\{[1, 1].a, [2, 3).b, [4, 5].c\}$. If the process does not engage in the a event at time 1, then the choice set is reduced to $\{[2, 3).b, [4, 5].c\}$. If the process then does not engage in the b event, then the choice set is reduced to $\{[4, 5].c\}$. If time progresses and the process fails to engage in the c event during the specified time interval, no further choices may be made and the process behaves as *Stop* from that point on. This should be regarded as a mistaken construction. Deterministic timed choice requires that at least one of the choices is taken during any process execution.

Furthermore, each of the choices should be *distinct*, paralleling the untimed CSP model. Distinct timed events either have disjoint enabling intervals, or disjoint untimed events, or both.

3.4 Deterministic process descriptions

CSP+T at present can only describe deterministic processes, and describing nondeterministic process behaviours is part of ongoing work.

The lack of nondeterminism means that the current language is limited as a *specification* technique, but not as an *implementation* technique. It is well known that nondeterministic process descriptions may be viewed as a way of underspecifying process behaviours (whether they be timed or untimed). Implementations on the other hand are seldom nondeterministic:

"Of course, $\sqcap$ is not intended as a useful operator for *implementing* a process. ... The main advantage of nondeterminism is in *specifying* a process.[7, last par. Section 3.2]"

4 Describing the system using Timed CSP

Consider a simple one-place buffer (with input channel *in* and output channel *out*) which has no timing constraints. The simplest implementation possible is given by

$$\mu\, X \bullet in?x \longrightarrow out!x \longrightarrow X \tag{1}$$

where an input is immediately output before allowing a further input.

If we introduce time into the above process, then it is possible to interpret the lack of any explicit temporal separation in Equation (1) between two successive events (such as *in* then *out* communications) in at least two ways.

In the first view, the lack of explicit timing may be interpreted as allowing successive events to occur at the same time while maintaining any sequencing order. For example, a sequence such as $a \longrightarrow b \longrightarrow \ldots$ is differentiated from the sequence $b \longrightarrow a \longrightarrow \ldots$, despite both events being observed at the same global time (according to the observer's watch, say). If both a and b are observed at time 1, the former ($a \longrightarrow b \longrightarrow \ldots$) has a trace $\langle 1.a, 1.b, \ldots \rangle$ while the latter has a trace $\langle 1.b, 1.a, \ldots \rangle$.

In the second view, the lack of explicit timing is interpreted as allowing events to occur at any time, again provided that any sequencing is preserved. A sequence such as $a \longrightarrow b \longrightarrow \ldots$ where the a occurs at time 0 for example would allow the b event to follow at any time taken from the half-open interval $[0, \infty)$ after the a, such as $\langle 1.a, 1.b \rangle$ or $\langle 1.a, 25.b \rangle$ or even $\langle 1.a, (\pi \times 10^{129}).b \rangle$.

The proposed extended CSP (CSP+T) as well as Reed and Roscoe's Timed CSP use this latter view. Other algebras adopt the former view, and use a temporal operation or process to provide the required interevent delay.

We start, then, with the Timed CSP model first proposed by Reed and Roscoe [11], which has been subsequently modified to eliminate the system delay constant [3] so that any event timing must be explicitly described using a *WAIT* process.

Producing a buffer which delays each message by the required delay is straightforward in this model:

$$SPB = \mu\, X \bullet in?x \longrightarrow WAIT\ I\ ; out!x \longrightarrow X \tag{2}$$

with the interval $I = [2, 5]$. This buffer accepts an input, then delays by an amount taken from the interval I, and then outputs the message. Notice that there is an asymmetry in this process. Despite ensuring that the input to output timing is correctly defined, the spacing between an output and a following input is defined to occur at any time in the interval $[0, \infty)$. Timing between input to input, output to output and input to output timings are *dependent on each other.*

The timings between inputs may be defined by *constraining* the above process by composing *SPB* in parallel with

$$IN = \mu\, X \bullet in?x \longrightarrow WAIT\ 1\ ; IN \tag{3}$$

and similarly, the output may be constrained by the parallel composition of *SPB* with

$$OUT = \mu X \bullet out!x \longrightarrow WAIT\ 2\ ; OUT. \tag{4}$$

Note that each of the buffers expressed in Equations (2), (3), and (4) implement only a single part of the required behaviour. Further, there is no message storage—only one single message is ever "in transit". Achieving the three goals simultaneously (specific message transit delay, differing input and output rates) cannot be done with a single process which is based on (1). Instead, we use a finite size buffer as a starting point. The buffer presented (*Buff0*) is based on the infinite buffer of example X9 in [7, p138]. [1]

$$Buff0 \mathrel{\widehat{=}} W_{\langle\rangle}$$

where

$$\begin{array}{lcl} W_{\langle\rangle} & = & in?x \longrightarrow W_{\langle x\rangle} \\ W_S & = & \ \textbf{if}\ \#S < 128 \\ & & \ \textbf{then}\ \left(in?x \longrightarrow W_{S^\frown\langle x\rangle} \,\Box\, out!S_0 \longrightarrow W_{S'}\right) \\ & & \ \textbf{else}\ out!S_0 \longrightarrow W_{S'} \\ & & \ \textbf{fi} \end{array} \tag{5}$$

This buffer, like any other implementation, must resolve what actions to take under "error" conditions such as the buffer filling or messages arriving at the incorrect rate. The original naive specification did not point out what buffer behaviour and event timings are acceptable when the buffer is encounters these error conditions, and any correct implementation would resolve these issues at the specification stage.

In view of the above discussion, let us modify the behaviour of the above buffer so that once it fills, it engages in a *Full* event (presumably signalled to the buffer's environment), and then breaks.

$$Buff1 \mathrel{\widehat{=}} X_{\langle\rangle}$$

where

$$\begin{array}{lcl} X_{\langle\rangle} & = & in?x \longrightarrow X_{\langle x\rangle} \\ X_S & = & \ \textbf{if}\ \#S < 128 \\ & & \ \textbf{then}\ \left(in?x \longrightarrow X_{S^\frown\langle x\rangle} \,\Box\, out!S_0 \longrightarrow X_{S'}\right) \\ & & \ \textbf{else}\ Full \longrightarrow Stop \\ & & \ \textbf{fi} \end{array} \tag{6}$$

The input and output timing requirements for this buffer using Timed CSP are captured by using the parallel composition of processes defined in Equations (3) and (4). The transit delay constraint may be met by introducing a another process into the parallel composition with the *Buff1* process, which spaces inputs to outputs using the parallel composition:

$$TD = in?x \longrightarrow (WAIT\ [2,5]\ ; out!x \longrightarrow Stop) \mathbin{|||} TD \tag{7}$$

with the parallel composition done using an interleaving to prevent the system from deadlocking at the very first recursive call.

Therefore, the system of processes

$$Buff1 \parallel IN \parallel OUT \parallel TD$$

gives the required timing characteristics for the buffer *provided that the buffer is not full.*

[1] This paper adopts the conventional notation **if** b **then** P **else** Q **fi** for the CSP conditional $P \not< b \not> Q$ where b is a boolean value, P and Q processes.

4.1 Discussion

This specification style is commonly referred to as *constraint-oriented*, and is used extensively in both timed and untimed Formal Description Methods. Each constraint may be regarded as representing a refinement step in moving from an untimed model to a timed model. Although this method is attractive in simple timing descriptions, it is the author's experience that the use of the $WAIT$ construct may lead to awkward and unnatural formulations of complex timing relationships.

Furthermore, the analysis of parallel composed systems (both timed and untimed) may require them to be reduced to equivalent sequential systems. However, processes such as

$$(a \longrightarrow P) \parallel (WAIT\ n\ ; b \longrightarrow Q) \tag{8}$$

cannot be reduced to a unique sequential process from within a model which defines timing properties solely using a $WAIT$-like operation. Rewriting the above system to a sequential form requires the introduction of a way of specifying the future behaviour of a process by the times at which preceding events occurred. In the above case, the process behaviour is determined by the time at which the first event occurs, and so recording this time in some manner is important in the "serialisation" of the system. This is identified both by Schneider [12] and Fidge [5]. Schneider proposes a new operator to do this, and creates a pre-normal form for a Timed CSP language. Fidge defines a way of labelling events such that causal relationships may be expressed as directed graphs which leads to a true concurrency semantics for a real-time process calculus based on CCS.

The approach taken in CSP+T differs from these. Rather than attempting to reduce a set of concurrent processes to some simpler sequential forms, the proposed extensions provide more expressive power to the sequential aspect of CSP. This reduces the need for introducing additional parallel constraining processes which may be difficult to analyse and also allows some algebraic manipulation of processes. Thus Equation (8) may be rewritten as:

$$\begin{array}{l} a \bowtie u \longrightarrow (P \parallel (E_1(u).b \longrightarrow Q)) \\ \Box \\ b \bowtie v \longrightarrow ((E_2(v).a \longrightarrow P) \parallel Q)\,. \end{array}$$

Note that the enabling intervals E_1 and E_2 have not been specified in this example, as it is meant to illustrate that the future behaviour of the system depends upon specific events in the system's execution. Of course, this is only one of many interpretations that may be expressed by this notation. The enabling intervals may also be defined solely in terms of a single marker variable, or alternatively, they may both be functions of both marker variables.

We now move back to the description of the store and forward communication system.

5 Describing the system using CSP+T

Recall that we are trying to describe a finite buffer with specific input, output and transit delay requirements. Since we do require that the buffer hold more than one item, we start again using the buffer given by Equation (5), and consider the specification of the input and output timings.

The buffer engages in only two events. Either it inputs a value and places it at the end of the queue, or it outputs the head of the queue. We therefore associate a marker variable with the input and output to capture their respective event enabling intervals. The enabling interval function for input is solely a function of the input marker variable. The enabling interval for output is similarly expressed in terms of the output marker variable. Let us assume again that the buffer engages in a final *Full* action before stopping once it is full.

Let E_i represent the input enabling interval, and E_o represent the output enabling interval. Then we set

$$\begin{aligned} E_i &= \{s \mid s = rel(1, v_i)\} \\ E_o &= \{t \mid t = rel(2, v_o)\}. \end{aligned}$$

The buffer stores input events as a queue of time-stamped events of the form $(t_i.x)$, where t_i represents the time at which a message contained in x was received. Let *eventof* be a function which strips out the time component of any such message.

A buffer which implements the appropriate input and output timings is

$$Buff2 \mathrel{\widehat{=}} Y_{\langle\rangle} \tag{9}$$

where

```
$Y_{\langle\rangle} = E_i.in?x \bowtie v_i \longrightarrow Y_{\langle v_i.x\rangle}$
$Y_S =$ if $\#S < 128$
        then   $E_i.in?x \bowtie v_i \longrightarrow Y_{S^\frown\langle v_i.x)\rangle}$
               $\Box$
               $E_o.out!eventof(S_0) \bowtie v_o \longrightarrow Y_{S'}$
        else $Full \longrightarrow Stop$
        fi
```

The transit delay constraint means that all of the messages held in the buffer differ between two and five time units since the time that they were input to the time they are output, relative to the current time. We now add in this constraint to *Buff2*. Define *age* to be a function which returns the age of the message at the head of the queue (by comparing the message time with the current time).

$$Buff3 \mathrel{\widehat{=}} Z_{\langle\rangle}$$

where

```
$Z_{\langle\rangle} = E_i.in?x \bowtie v_i \longrightarrow Z_{\langle v_i.x\rangle}$
$Z_S =$ if $\#S < 128$
        then if $age(S) < 2$
             then $E_i.in?x \bowtie v_i \longrightarrow Z_{S^\frown\langle v_i.x\rangle}$
             else if $2 \leq age(S) < 5$
                  then   $E_i.in?x \bowtie v_i \longrightarrow Z_{S^\frown\langle v_i.x\rangle}$
                         $\Box$
                         $E_o.out!eventof(S_0) \bowtie v_o \longrightarrow Z_{S'}$
                  else if $age(S) = 5$
                       then $[0, 0].out!eventof(S_0) \bowtie v_o \longrightarrow Z_{S'}$
                       else $Stop$
                       fi
                  fi
             fi
        else $Full \longrightarrow Stop$
        fi
```

Notice that this description is more *prescriptive* and *sequential* in nature than the previously given Timed CSP specification. Timed CSP specifications tend to be presented as a parallel composition of component processes, with each component represents a separate timing constraint. It describes the buffer's operation for the cases when the message age in the buffer is outside of the transit delay requirements in addition to the case when buffer overflow occurs.

An alternative, more *descriptive* or abstract specification in CSP+T could have used the transit delay constraint given in Equation (7) in parallel with the *Buff2* process given in Equation (9):

$$\mathit{Buff4} = \mathit{Buff2} \parallel \mathit{TD}.$$

In this system, we use the marker events and enabling intervals in specifying the input and output timing, and the transit delay requirement is specified as a constraining process on the *Buff2* process.

6 Conclusions

Timed CSP and similar related specification techniques define system timing within a sequential process by the use of a specific interevent delay. More complex timing relationships are described by using processes using such delays and composing them in parallel with each other, thereby producing a set of independent timing constraints.

However, there are some systems which cannot be described in this way, since the timing relationships are not independent of each other and preceding events. Such systems of processes cannot be converted into equivalent sequential forms (e.g. Equation (8)). To deal with such systems, we need to increase the expressiveness of the specification language.

This paper has introduced a new real-time description language, CSP+T, which addresses these problems. CSP+T extends the untimed CSP language in two ways. First, all events have an enabling time interval, over which the event is expected to be observed only once during any particular execution. The second extension is that these time intervals may be expressed in terms of a set of arbitrary marker events within a process' execution.

In order to focus the discussion, a (naively defined) time-constrained buffer was specified, first in Timed CSP, then in CSP+T. As was observed in this example, there is a difference in the two specification styles. Timed CSP encourages the use of parallel composed constraining processes to define event timing relationships, whereas CSP+T encourages the use of a single, sequential process to define event timing.

It is felt that although these styles are complementary, careful use of both approaches will prove to be beneficial in specifying complex system timings.

Acknowledgements I wish to thank Colin Fidge, Steve Schneider, Asis Goswami, Jacek Olszewski, and the anonymous referees for providing useful comments and insights on earlier drafts of this paper.

References

[1] ISO/TC 97/SC 16/ WG 6. Information Processing Systems – Open Systems Interconnection – Transport Service Definition – Connectionless mode transmission. Standard ISO-8072-1986-Addendum1, ISO, 1986.

[2] Ed. Brinksma. An Introduction to LOTOS. In H. Rudin and C.H. West, editors, *Protocol Specification, Testing, and Verification, VII.* Elsevier Science Publishers B.V., Amsterdam, May 1987.

[3] Jim Davies and Steve Schneider. A brief history of Timed CSP. Technical report, Programming Research Group, Oxford University, Oxford OX1 3QD UK, 1992.

[4] D. Ferrari. Client requirements for real-time communication services. Internet RFC, 1990 November.

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

[6] R. Gerber, I. Lee, and A. Zwarico. A complete axiomatization of real-time processes. Technical Report MS-CIS-88-88, Dept. of Computer and Information Science, School of Engineering and Applied Sciences, Uni. of Pennsylvania PA 19104, November 1988.

[7] C.A.R. Hoare. *Communicating Sequential Processes.* International Series in Computer Science. Prentice-Hall International (UK) Ltd, 66 Wood Lane End, Hemel Hempstead, Hertfordshire HP2 4RG UK, 1985.

[8] R. Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science.* Springer-Verlag, Berlin–Heidelberg–New York, 1980.

[9] Robin Milner. *Communication and Concurrency.* International Series in Computer Science. Prentice-Hall International (UK) Ltd, 66 Wood Lane End, Hemel Hempstead, Hertfordshire HP2 4RG UK, 1989.

[10] Juan Quemada and Angel Fernandez. Introduction of quantitative relative time into LOTOS. In H. Rudin and C.H. West, editors, *Protocol Specification, Testing, and Verification, VII*, pages 105–121. Elsevier Science Publishers B.V., 1987.

[11] G.M. Reed and A.W. Roscoe. A Timed Model for Communicating Sequential Processes. In *Automata, Languages, and Programming , 13th Intl. Colloqium Proceedings, Lecture Notes in Computer Science*, Berlin–Heidelberg–New York, 1986. Springer-Verlag.

[12] Steve Schneider. Unbounded Nondeterminism for Real-Time Processes. Technical Report TR-12, Programming Research Group, Oxford University, UK, 8-11 Keble Rd Oxford OX1 3QD, July 1992.

[13] J.J. Zic. A New Communication Protocol Specification and Analysis Technique. Technical Report TR287, Basser Department of Computer Science, July 1986.

[14] John J. Zic. *CSP+T: a formalism for describing real-time systems.* PhD thesis, Basser Department of Computer Science, University of Sydney, NSW 2006, July 1991.