Timed Actors
DRAFT

H. John Reekie

Dept. of Electrical Engineering and Computer Science
University of California at Berkeley
johnr@eecs.berkeley.edu

Abstract

This technical note explores the notion of time in the context of the actor model Actif. Actif tries to use the minimum number of constructs necessary to express actors that will work in a broad range of models of computation. To this end, in this report I add a single construct to Actif, which delays the readiness of an actor to fire an action by a certain amount of time. This has some interesting consequences for the interface automata used to compose actors.

This technical note can be referenced as:

"Timed Actors," H John Reekie, UC Berkeley, November 2002,
online at: `http://ptolemy.berkeley.eecs/~johnr`.

Or use the following BibTex citation:

```
@unpublished{Reekie0202,
  author = {H. John Reekie},
  title = {Timed Actors},
  month = {November},
  year = {2002},
  note = {Online at \href{http://ptolemy.eecs.berkeley.edu/~johnr/}
    {http://ptolemy.eecs.berkeley.edu/~johnr/notes/timedactors.pdf}}
}
```

1 Introduction

An actor in Actif consists of a set of actions, each of which consumes and produces a fixed number of tokens from and to each input and output channel. A firing automaton constrains the order in which actions can be fired.

Such actors can be readily translated into an interface automaton. (This has only been presented informally [2]; a more rigorous presentation is forthcoming.)

2 Timed Actif

Time is added to Actif with a single concept: after completing an action, the next set of eligible actions are not enabled until after a given amount of time has elapsed. This is indicated by the *after* keyword. (This concept is not new. Ptolemy II, for instance, has a `refireAt()` method to perform the same function.)

For example, here is an actor that emits a value ('1' in this case) every second:

```
actor tick () (int out) {
  action a (() -> [1]) next a after 1;
}
```

When it is first fired, the actor immediately emits the value '1' on its output port. The next eligible action is a; however, this action cannot be executed until exactly one second has elapsed. The firing automaton of this actor is shown in Figure 1a.

Here is an actor that reads an input token, then produces it on its output channel ten seconds later:

```
actor delay (word in) (word out) start a {
  state {
    word s;
  }
  action a ([x] -> []) next b after 10 {
    s = x;
  }
  action b ([] -> [s]) next a;
}
```

That is, when a executes, it consumes a token and stores it in the variable s. Exactly ten seconds later, b executes and writes the value to the output channel. The firing automaton of this actor is shown in Figure 1b.

This actor raises a number of questions. First, what happens if a second token arrives at the input within ten seconds? The answer is simple: nothing. This is an illegal condition, and should be detected when we try to compose this actor into a network that would produce this condition. If the actor is to allow this, it must be written specifically so that it can receive and buffer input tokens before producing the corresponding output token.

Second, is there a time delay between firing actions b and a? In this case, the answer is yes. I'm going to go out on a limb and say that there is always an

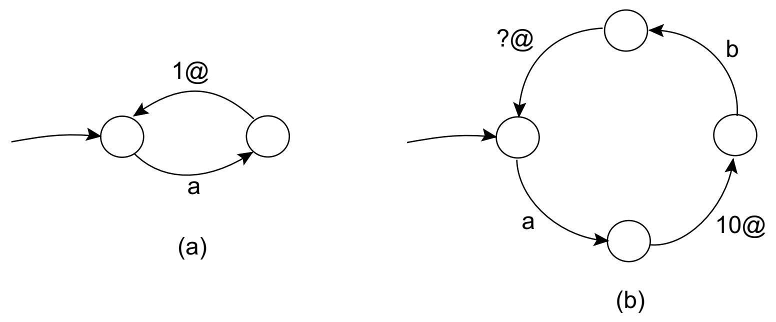

Figure 1: Firing automata of timed actors: a) the *tick* actor; b) the *delay* actor

implicit passing of time after any action that returns to the starting state—this is so that a subsequent sequence of firings can proceed at a different time, which is what one would expect in, say, a discrete-event simulation.

3 Timed Automata

An Actif actor can be translated into an interface automaton. In a previous note [2], I used automata to compose actors into dataflow networks. These automata include:

- 'get' and 'put' actions that interface to communication buffers, and
- 'fire' and 'return' actions that provide the interface to a controller, which governs the execution of the network of actors

For timed actors, we need a way to have the automata represent the passing of time. Alfaro *et al* present timed interface automata [1]; that work seems more suitable for modeling inexact real-time, whereas the approach in this note seems more suited to modeling exact simulation time.

3.1 Timed transitions

To represent the passing of time, an interface automaton can have one or more *timed transitions.* Such a transition represents the passing of an exact amount of time, and is labeled, say, 10@ to denote the passing of ten seconds[1], or t@ to denote the passing of t seconds, where t is a state variable of the actor. A transition that represents the passing of an "unknown" amount of time is marked ?@—in effect, this is saying that this automaton is prepared to allow time to pass as long as it is composed with another automaton that is also prepared to let time pass.

[1]For simplicity I assume that the unit of time is seconds.

One wya to generate a timed automaton of this nature from an Actif actor: after each action with an *after* clause, insert an additional state, with a timed transition marked with the appropriate value to the next state. In each cycle, insert a timed transition labeled ?@ just prior to the return to the start state.

Figure 2a through 2e illustrate a number of timed automata. Figure 2f is a little different; see Section 4.

3.2 Composing timed automata

What does a timed transition mean? When an automaton reaches a timed transition labeled t@, it is saying that time must advance by exactly t seconds when that transition is taken. When we compose actors into a network, we assume that all actors in the network share the same time. Therefore, timed transitions are a kind of shared event, as both automata in a composition must pass time together. If many automata representing actors in a network are composed, then this means that *all* actors pass time together. This matches well with the intuitive notion that time in a discrete-event simulation advances in global steps.

Suppose we are composing two automata with timed transitions t_1 and t_2. If t_1 and t_2 are the same, then we create a timed transition in the composition, labeled say t_1@. That is, the composition of the two automata also passes exactly t_1 seconds on this transition. In general, t_1 and t_2 are not identical. Suppose $t_1 > t_2$. Then the composition must have a shared transition labeled t_2—that is, the two automata will pass the smaller amount of time together. There will also be a "leftover" amount of time $t_1 - t_2$. This time will need to appear on a different transition in the composition, most likely as a result of combining this "leftover" time with a "don't-care" timed transition.

Thus, each pair of timed transitions can generate three different transitions, for the cases $t_1 < t_2$, $t_1 = t_2$, and $t_1 > t_2$. Following is the scheme I dreamed up for achieving this desired result.

1. Perform a pre-processing step on the automata P and Q. For each pair of timed transitions labeled t_1@ and t_2@ (where neither is ?@), add an additional state and two transitions as shown in Figure 3.

2. Compose the automata. Timed transitions are treated like shared actions, provided that certain conditions are met. These conditions are detailed below; in essence, they ensure that both automata agree to pass the exact same amount of time together. If this cannot be guaranteed, then no transition is produced in the composition.

Here are the rules for producing a timed transition in the composition. p_1 etc denotes a predicate on time values only.

(1)	$?$	$\otimes$	$?$	$\Rightarrow$	$?$
(2)	t_1	$\otimes$	$?$	$\Rightarrow$	t_1
(3)	$?$	$\otimes$	t_2	$\Rightarrow$	t_2
(5)	p_1/t_1	$\otimes$	p_2/t_2	$\Rightarrow$	$(p_1 \wedge p_2 \wedge t_1 = t_2)/t_1$

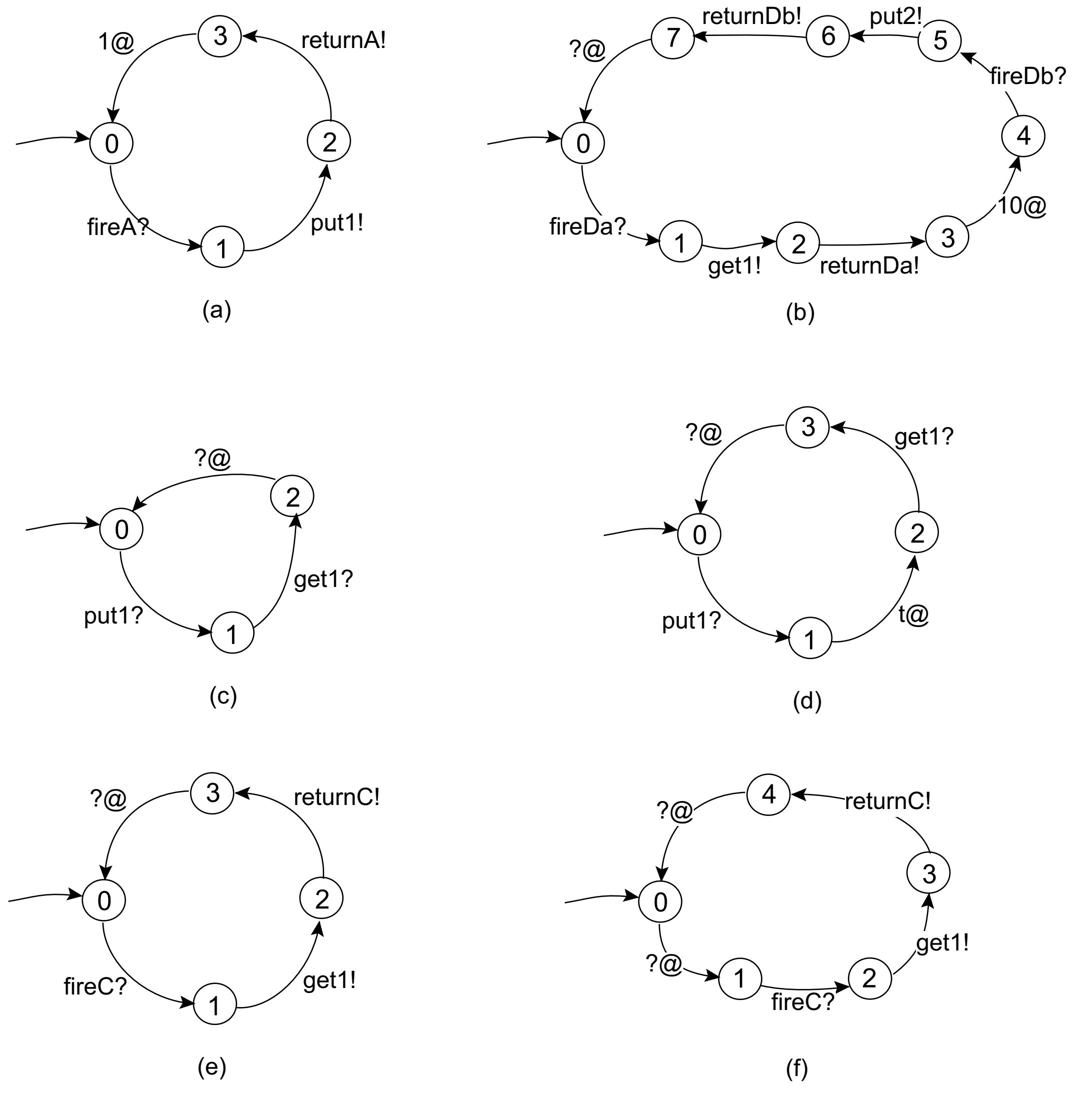

Figure 2: Some timed interface automata: a) the *tick* actor; b) the *delay* actor; c) a single-place buffer; d) a single-place buffer with time delay; e) a consumer with post-fire delay; f) a consumer with pre- and post-fire delay

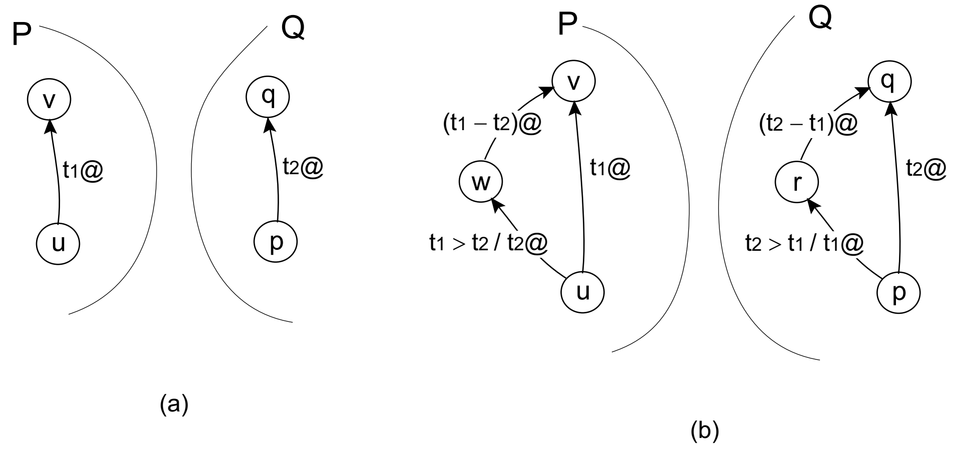

Figure 3: Pre-processing of timed interface automata: a) before; b) after.

An interpretation of the rules:

1. If two transitions don't care how much time passes, then together they don't care either.
2. If one transition passes t_1 and the other doesn't care, then together they pass t_1.
3. Same as above.
4. If both transitions pass a certain amount of time, then together they agree to pass the same amount of time as long as their time values are equal and their predicates are true.

Any other possible pairings of timed transitions do not appear in the composition. The result of applying the above rules to the automata in Figure 3b is illustrated in Figure 4, assuming that the two automata also have other "don't care" transitions that compose with the $t_1 - t_2$ and $t_2 - t_1$ transitions.

4 Examples of interface automata composition

In this section I will illustrate composition of some automata from Figure 2. First, consider the composition of the *tick* actor (Figure 2a) with the simple timed buffer (Figure 2c). Assuming that the system is single-threaded, we can prune any path that has nested fire-return actions (as in [2]), resulting in the automaton in Figure 5a. The composition of the *tick* actor and the timed buffer passes exactly one second at the completion of each cycle. When this is further composed with the consumer in Figure 2e, the result is as shown in Figure 5b.

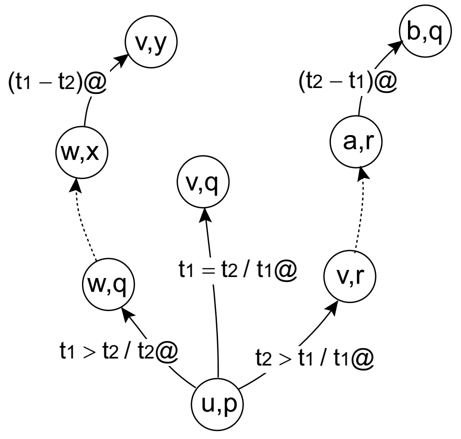

Figure 4: Result of composing timed transitions

The buffer in Figure 2d delays a value that it received by t seconds before allowing it to be read. This buffer can be used to illustrate some of the issues that occur with these timed interface automata in a simpler way than using the *delay* actor. When the *tick* actor is combined with this buffer, the result is shown in Figure 5c. Of the three transitions created by the combination of the transitions labeled t@ and 1@, only one remains, for $1 > t$. The other two, $t = 1$ and $t > 1$ have been pruned—these transitions led to an illegal state (*tick* was ready to produce *put1*, but the buffer was not prepared to receive it), and were therefore pruned. Referring back to my earlier comment that composition of two automata should remove illegal conditions, we can see that this has indeed happened. The case where $t > 1$, that is, the delay is longer than the tick interval, is simply not present in the composition. We have not prevented it from occuring at run-time (assuming that we cannot determine t prior to execution), but we have at least exposed a condition necessary for correctness.

If the above automaton is then composed with the consumer of Figure 2e, the result is null. This consumer does not allow time to advance before it performs its first *get*, and so is incompatible with the buffer that delays its output. A consumer that does compose is the one shown in Figure 2f, which has a "don't-care" delay immediately before firing. This latter consumer does not, however, compose with the earlier automaton of 5a! The reason is that the ?@ transition *forces* synchronization of time: in a composition, *all* automata must be prepared to pass time together with this "don't-care" timed transition, even if the amount of time is zero.

As a result, there is not a straight-forward translation of timed Actif actors into timed interface automata. (Additional analysis of the network will be needed to derive the correct automata.) It may be possible to somehow loosen the meaning of ?@ so that there is; but this would introduce other complications.

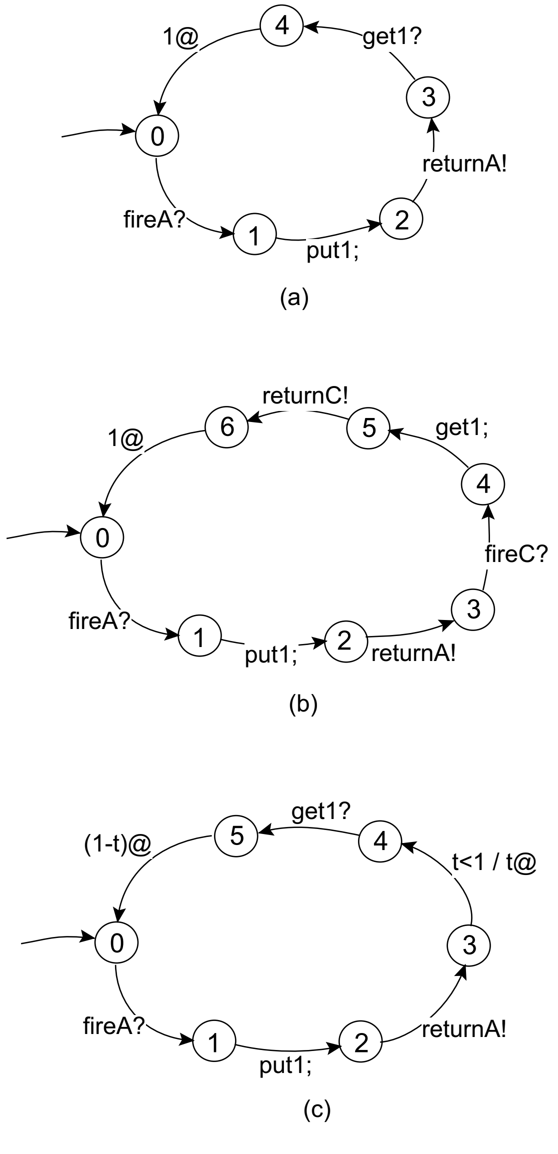

Figure 5: Automata composition: a) *tick* and the simple timed buffer; b) *tick*, the simple timed buffer, and the simple timed consumer; c) *tick* and the timed buffer with delay

This needs to be investigated further.

5 Concluding remarks

Timed interface automata of the nature described here appear to have useful properties for composing networks of actors and buffers. As we have seen, if we can generate timed interface automata for timed Actif actors, we can answer some questions about whether they compose into a network. With some additional analysis, constraints on the timing embedded into actors can be derived (for example, that the time delay must be less than a certain amount if the composition is to remain live).

A picture is beginning to emerge about what a “model of computation” is in the context of Actif and interface automata. So far, we have seen the following elements that vary according to the model of computation:

- The mapping from an actor’s firing automaton into an interface automaton
- The interface automata of buffers
- Additions to automata composition (pruning of illegal-fire-return sequences in single-threaded models of computation; timed transitions in timed models of computation)

References

[1] Luca de Alfaro, Thomas A. Henzinger, and Marielle Stoelinga. Timed interfaces. In *Proceedings of the Second International Workshop on Embedded Software*. Lecture Notes in Computer Science, Springer-Verlag, 2002.

[2] H. John Reekie. Interface automata and actif actors. Online at http://ptolemy.eecs.berkeley.edu/ johnr/notes/atomicbuffer.pdf, November 2002.