Properties as Processes : their Specification and Verification

Joel Kelso and George Milne

School of Computer Science and Software Engineering
University of Western Australia
{joel,george}@csse.uwa.edu.au

Abstract. This paper presents a novel application of an untimed process algebra formalism to a class of timing-critical verification problems usually modelled with either timed automata or timed process algebra. We show that a formalism based on interacting automata can model system components, behavioural constraints and properties requiring proof without elaborating the underlying process-algebraic formalism to include explicit timing constructs; and that properties can be verified without introducing temporal logic, model-checking, or refinement relation checking. We demonstrate this technique in detail by application to the Fischer mutual-exclusion protocol, an archetypal example of a system that depends of timing constraints to operate correctly.

1 Introduction

Many complex systems are most naturally modelled as collections of components that operate and interact concurrently. Such modelling allows a problem to be decomposed into parts having behaviour that is, in isolation, readily described. To operate correctly, some complex systems rely on timing relationships between certain critical actions shared by two or more components. In order to verify the correctness of such systems, the tools and methodologies used must be capable of expressing timing constraints and temporal properties in a manner clearly comprehensible to the user.

The contribution of this paper is twofold. Firstly, it demonstrates an intuitive way of describing relative orderings among timing intervals as processes (i.e., as state machines), which can be naturally composed with system model processes to supply the timing-critical aspects of the model's behaviour. This separation of timed and untimed behaviour helps specification, as it allows greater freedom in partitioning the work of constructing complex models.

Secondly, this paper describes how formal protocol verification may be achieved by use of a process algebraic equivalence checker coupled with the *concurrent composition* of (1) a system description, as a process, and (2) a process which describes the property requiring proof, which is also presented as a state / action / new-state type process. When both types of object are modelled as processes, there is no need for design engineers to learn a separate property description language or model checking tool in addition to a language with which to express system behaviour. We believe that this simplification, and the ability to express properties requiring verification in a state machine type manner, is of real value in encouraging engineers to adopt formal description and verification methods.

This paper uses the well-known Fischer protocol to illustrate both this treatment of timing representation, and the composition-based property verification technique.

A key feature of the methodology described in this paper is the central role of the concurrent composition operator. Concurrent composition is the fundamental mechanism for constructing system models in the process algebra paradigm; its use for this purpose warrants no additional comment. In our methodology, concurrent composition plays two further significant roles, namely to *enforce* timing constraints, and as the core of the *composition-based verification* technique.

1.1 Timing Constraints as Processes

Rather than encoding timing constraints as an integral part of a system model (which is the usual case with timed automata [1] and timed process algebra [17] modelling), timing constraints are encoded as separate processes that express relationships between time intervals.

This is accomplished by first determining which actions in the system model signify the boundaries of time critical time intervals, and then defining *timing constraint processes* which express the allowable sequences of occurrences of these events. When these processes are *composed* with the system model, they enforce the timing relationships that they encode.

In this way the modelling of system behaviour can be decoupled from the timing constraints. This simplifies model development and experimentation, since timed aspects of a model can be altered without modification of the time-insensitive aspects.

1.2 Properties as Processes

The idea of expressing properties requiring proof as processes is a well-known process algebraic technique, described for example in [5, 15, 19]. In the case where a correctness specification is a complete description of a system, verification proceeds by checking that the system implementation process is *equivalent* to the specification process according to some semantic equivalence relation.

Frequently, however, total behavioural equivalence between two processes is not the goal of the proof process. For certain systems, verifying correctness consists of determining that certain properties do in fact hold for an implemented system. Such properties do not constitute a complete specification but are rather a particular relationship between a number of distinct actions. Verification of such properties then requires the demonstration that the occurrence of the property actions in the constructed system model process have the same sequence of occurrence designated by the property process.

One technique for accomplishing this is to *abstract* all non-property actions from the model process, and then check that the model *refines* the property process according to a semantic ordering relation (see [19] chapter 14 for example).

In this paper we describe an alternative proof mechanism that avoids the introduction of the concept of process refinement orderings, and in which concurrent composition plays a crucial role.

In section 2 we present the mechanism which underlies our property verification technique. In section 3 we demonstrate the technique by application to the Fischer protocol, showing in detail how timing constraints *and* correctness properties are formulated as processes. In section 4 we discuss the significance of this work and contrast it with related work.

2 Checking Properties via Composition

We show how the verification of a class of properties, safety properties, can be performed in an process algebra (or *interacting automata*) based framework by making use of the concurrent composition of processes and process equivalence testing – provided that the process composition operation has certain features.

Our description of this technique is framed in terms of the Structural Operational Semantics approach to formalising process behaviour [18]. Under this approach, processes are identified with *labelled transition systems* (LTS). A LTS is a rooted directed graph where each edge is labelled with an *action*. Each vertex of the graph is a distinct *state* of the process, and each edge represents a *transition* between states, with transition labels determining the interaction between the process and its environment (or with other processes).

Labelled transition systems admit a variety of different equivalence relations and orderings, such as *trace equivalence*, *testing equivalence* and *bisimulation*. The technique we present here can be used with any of these process equivalence relations, yielding criteria for the fulfillment of safety properties which vary in sensitivity to internal (unobserved) process nondeterminism. Trace equivalence is assumed here, since it is both simple and sufficiently discriminating for the examples in this paper.

2.1 Safety Properties and Concurrent Composition

A *safety property* of a system is a property which states that “nothing bad” will ever happen. When expressed as a process, a *safety property process* exhibits only allowable behaviours – the set of behaviours that a system must not overstep if it is to fulfill that property.

Note that property processes can usually be expressed in terms of only a *subset* of the actions in the system model process. The actions which occur in a property process can be thought of as the actions that the property “cares about” – those that directly affect the truth or falsity of the property in a particular system. All the other actions occurring in the system model processes, while they might be vital for the functioning of the model, should be ignored by the property. This is significant since properties can be much simpler than a full system specification.

The concurrent composition of processes is used to verify that a system correctly satisfies a particular safety property using the following procedure:

1. The system model process is composed with the property process so that *they synchronise only for the events in the property process.*
2. This composite process is compared to the system model process: if the two are equivalent, then the system fulfills the safety property.

This procedure is summarised by an equation that must hold in order for system S to fulfill property P:

$$S * P \cong S \tag{1}$$

where $S * P$ denotes the concurrent composition of S and P.

To see how the *concurrent composition* of processes can be used to perform a safety property check, consider an example property process P and two different system component processes S and T, pictured in Figure 1. Process P represents the property that all occurrences of actions a and b must begin with a and then strictly alternate.

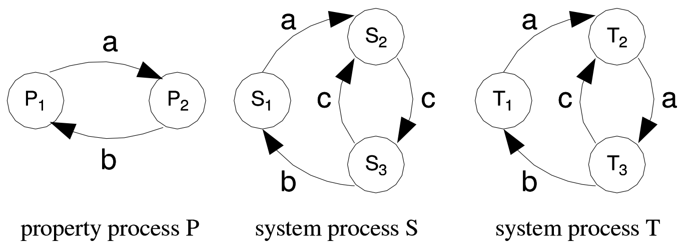

Fig. 1. Example property and system processes.

By having P operate in parallel with S and synchronising on actions a and b, P can be considered to be "supervising" S, watching for occurrences of actions a and b. Let us follow the possible activity of the combined process $S * P$.

Both processes begin in state 1. In state S_1, S can perform action a and transition to state S_2. Since P and S synchronise on action a, P participates in this action and also transitions from state P_1 to state P_2.

In state S_2, S can perform action c and transition to state S_3. Since P is uninterested in action c, S is free to perform this action without any change in P. In state S_3, S may again perform c and return to state S_2. S may thus perform any number of c actions while P remains in state P_2.

In state S_3, S may also perform a b action. In this case, P must be in state P_2, and is also ready to perform action b, returning both processes to state 1.

In this example, process S is never prevented from performing an action by P. The behaviour of the composite process $S * P$ is thus equivalent to S, so S satisfies property P.

Process T provides a contrasting example of a process that fails to satisfy property P. By again following the concurrent behaviour of T and P we can see how this is detected.

The initial behaviour of T and P is the same as S and P – both participate in action a and transition to state 2. In state T_2, T can only perform action a. Since T and P synchronise on action a, P must also perform a. But in state P_2, P is only ready to perform action b. Neither process allows the other to continue, and so the composite behaviour of T and P ends at this point. This behaviour (a single occurrence of a) is clearly not equivalent to the behaviour of T, so T does not satisfy property P.

These examples are extremely simple, but the technique operates correctly for arbitrarily complex processes, including those where both system and property are nondeterministic. The soundness of this property checking technique is proven in [9].

In order for this proof technique to work, the concurrent composition operation must have two important characteristics. Firstly, the operator must be able to enforce synchronisation for actions in the property process, while allowing free asynchronous activity for other actions. Secondly, the operator must allow *multi-way synchronisation.* It must allow two *or more* processes to participate in an interaction, so that property processes can synchronise on the *same actions* present in the system processes. This enables a property process to monitor system processes and restrict their behaviour to activity that correctly satisfies the property. If the system processes *do* contravene the specified property, then the equivalence check will detect the fact that the composite system's activity has been curtailed, signifying that the safety property is not satisfied.

The concurrent composition operator of the CIRCAL formalism [14], used in this paper, has these characteristics. The CCS [16] parallel composition operator cannot be used in this manner since CCS synchronisation operates with complementary pairs of events, which are eliminated in the resulting composite process. The CSP [8] *generalised parallel* operator is suitable since the set of synchronisation events is an explicit parameter to the operator, and the operator allows multi-way composition.

2.2 Modelling With CIRCAL

In this paper we adopt the CIRCAL process algebra [14, 15] for our definition of model components, constraints and properties. Several different notations and toolsets have been developed for defining complex systems as CIRCAL processes. XCircal [15], used in this paper, is a C-like language in which the CIRCAL process algebra operators have been embedded, while [5] defines an intuitive and precise diagrammatic notation for CIRCAL processes. Also under development is a library of functional language combinators (in Haskell) for defining and manipulating CIRCAL processes [10], and a visual programming interface for building processes in diagrammatic form.

These representations build upon the same underlying CIRCAL process formalism, and enable modellers to exploit the formalism's important features. Three of the formalism's characteristics are particularly relevant to our proof technique. *Firstly*, the CIRCAL composition operator fulfills the partial synchronisation requirement necessary for the composition-based property verification technique.

Secondly, the CIRCAL composition operator is a multi-way operator in which an action shared by two processes remains visible in the composite process, enabling additional processes to participate in the event. This allows processes that implement behavioural constraints, diagnostic "probes" (see for example [15]), and correctness properties to be composed into a system model without having to modify the original processes.

Thirdly, the fact that transitions are labelled with *sets* of events allows arbitrary finite relations and functions to be constructed and incorporated into a model. These can be used to connect and adapt process components, or as model components in their own right.

3 Modelling and Verification Methodology

In this section we outline our modelling and verification methodology, then illustrate the methodology by application to a timing-dependent concurrent mutual-exclusion protocol. The methodology proceeds in three phases.

1. The first phase consists of identifying critical actions in the system being modelled and constructing processes that capture the essential details of the system's behaviour. This involves constructing explicit transition systems for parts of the system that can be modelled as simple finite state behaviours, and using concurrent composition and abstraction operations to construct larger, more complex systems in a hierarchical fashion. This phase is illustrated in section 3.2. At this stage the detailed time-critical aspects of the model may be ignored.
2. In the second phase, the model events that delimit critical timing intervals are identified. Timing constraint processes that specify the necessary relationships between these intervals are then constructed and composed together to obtain a timed system. This phase is illustrated in this paper in section 3.3.
3. The third phase consists of the definition and verification of required system properties, which is accomplished by the construction of property processes and application of the constraint-based verification technique. This phase is illustrated in section 3.4.

3.1 Modelling the Fischer Protocol

The Fischer Protocol [13] is a distributed algorithm for ensuring critical section mutual exclusion between a number of concurrent processes. The protocol is simple yet relies on timing constraints among its processes for correct operation. It has become a standard for demonstrating verification techniques for timed systems, see for example [2, 12, 21].

We demonstrate our property verification methodology by treating part of the specification for correct operation of a Fischer protocol system as a safety property. We model both the system and the protocol's essential correctness property (mutual exclusion) as processes, and verify that the modelled system satisfies the property.

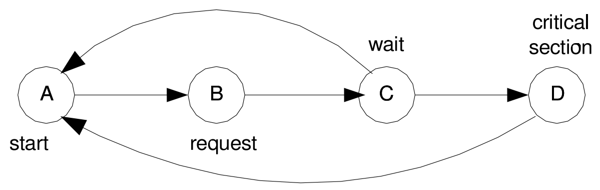

Fig. 2. The Fischer protocol worker process states.

System Description A Fischer protocol system consists of N *workers*. Each worker goes about some independent activity (not modelled here) and occasionally attempts to perform some activity which needs to be protected by a *critical section.* It is assumed that in order to operate correctly, the system must have the property that at most one worker is performing its critical section activity at any instant.

To enact the Fischer protocol, the workers interact by reading from and writing to a shared register. The register can take on one of $N + 1$ values, one for each process plus an "empty" state Z. Figure 2 shows the basic operational cycle of a Fischer protocol worker. Workers wait (or perform their non-critical activity) in the *start* state (A) until the register becomes empty. They may then indicate their intention to enter their critical section moving to the *request* state (B), in which case they must set the register to indicate the fact, and then make the transition to the *wait* state (C) within a certain time period. In the wait state the worker will either notice that another worker has made a later request, in which case this worker aborts its attempt to enter its critical section and returns to the start state; or the waiting period will elapse and the worker enters the *critical section* state (D). Eventually the worker exits its critical section and returns to the start state, setting the shared register to the empty state.

3.2 Process Models of System Components

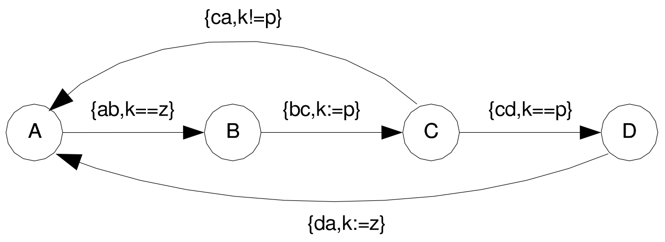

Fig. 3. The Fischer worker process model.

In the construction of our model of a Fischer protocol system, we utilise processes to model two quite different classes of object. In the following section we use processes to model abstract *temporal constraints* needed for the correct operation of Fischer's protocol. This leaves us free to model, in this section, the physical elements of the protocol system without regard to timed behaviour.

Worker processes are modelled in CIRCAL as behavioural processes in a straightforward way: a diagram of the worker process model is shown in Figure 3. The process has four states A, B, C and D. The transitions are labelled with two varieties of actions. There are actions of the form xy, where x and y are states; the purpose of these actions is to signal the activity of the process at every transition. As we shall see later, these actions will be shared with constraint and property processes in order to refine and analyse the system model.

Each transition is also labelled with an additional actions that indicate the worker's interaction with the shared register (which is also modelled as a process). These actions take the form $k := a$, where the worker writes a process name a to the register; or $k == a$ (or $k! = a$), where the worker reads and tests the value of the register. These actions are shared with the process model of the register, and coordinate the activity of the worker process with the register process.

The XCircal code for constructing a worker process is given in Figure 4. [1]

```
Process Fischer(Event ab,bc,ca,cd,da,ksetz,keqz,ksetp,
                      keqp,kneqp) {
  Process A, B, C, D;
  A <- (ab keqz) B
  B <- (bc ksetp) C
  C <- (ca kneqp) A + (cd keqp) D
  D <- (da ksetz)
  return A
}
```

Fig. 4. The XCircal code for the worker process.

This prototypical worker process is instantiated with events named to indicate the worker process in which they occur. For reasons that will become clear later, actions involving the empty register state also tagged with the worker's name. For example, the action $pk == z$ indicates that process P is testing to see if the register's value has value Z.

```
FischerP <- Fischer(pab,pbc,pca,pcd,pda,pksetz,pkeqz,ksetp,
                    keqp,kneqp)
FischerQ <- Fischer(qab,qbc,qca,qcd,qda,pksetz,qkeqz,ksetq,
                    keqq,kneqq)
```

The Shared Register Model Figure 5 shows a process which models the shared register for a system of two worker processes. The process has one state for each worker process, plus one state representing the "empty" state of the register (labelled z). *Write* actions of the form $k := a$ lead from every state to the state a. For each state a, *read* actions of the form $k == a$ lead from a to itself. For clarity, Figure 5 omits the read transitions of the form $k!{=}a$: for each A these are present as looping transitions for all states other than A. Worker processes performing write actions cause the register to change state, and worker processes will only be able to perform read actions if the register is in a compatible state. For brevity, we have omitted XCircal code for the remainder of the transition systems.

The use of this register model has an additional side-effect on the system model. Because the register process contains only single-action transitions, it prevents the si-

[1] Since XCircal does not allow them in event names, the non-alphabetic characters are transcribed to mnemonic characters in an obvious way.

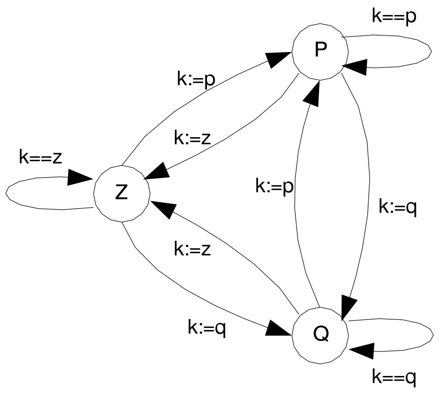

Fig. 5. The shared register process for a system with two worker processes P and Q.

multaneous occurrence of those actions which might otherwise arise through composition. This does not limit the scope of our verification, since inter-process concurrency is modelled through the arbitrary interleaving of actions, which is the approach used in process algebra that do not have simultaneous actions (e.g. CSP). In other contexts where it is desirable to model truely simultaneous access to registers, CIRCAL can be used to construct register models with a variety of concurrency semantics.

The Untimed System Figure 6 shows the Fischer protocol system of two worker processes, using a simple but powerful (and fully formal) diagramming notation introduced in [5]. In this notation, each rectangle represents an abstraction boundary containing one or more processes: all actions other than those appearing as "ports" on the rectangle's perimeter are abstracted and hidden from the exterior. In a simplified version of the notation, employed here, lines simply connect ports with identical names (thus denoting a single shared action). At the innermost level, processes are ultimately represented by transition diagrams. For reasons of space, we only show a single level of nesting in a diagram: the internal structure of internal processes are represented instead by process names.

There are several things to note about the composite system.

- The complete Fischer protocol system consists of the *concurrent composition* of the worker processes, the shared register process, and a register-access mediation process (see below).
- For actions which involve the empty state Z, communication between the register and worker processes are mediated by an additional process M (pictured in Figure 7). If these each of these events were modelled by a single system-wide action, this would force each action to be synchronised across all worker processes. This is clearly incorrect, since it would require all workers to rendezvous for reads or writes involving the Z register value. Considering the intended behaviour more carefully, we can see that outside of the register itself, the action of each worker setting (or checking) a particular register value are distinct events which can occur

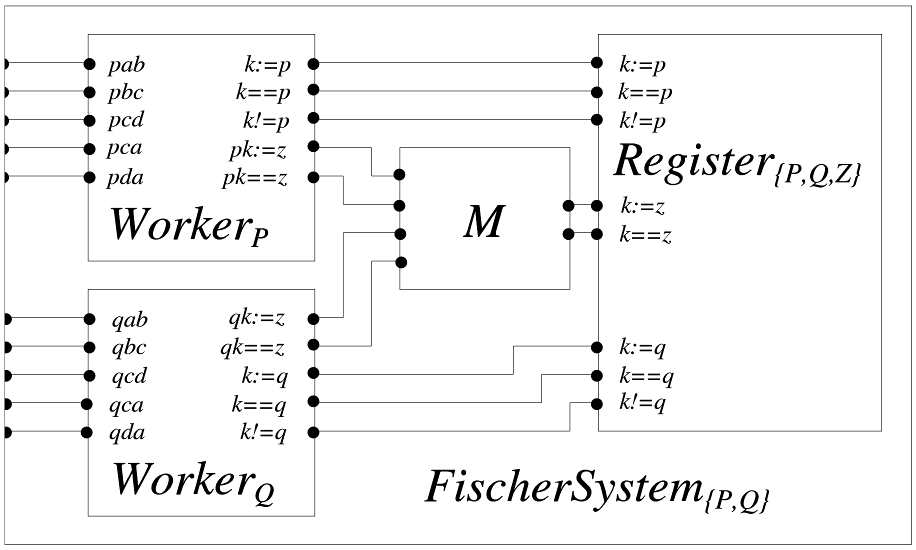

Fig. 6. Diagram of untimed Fischer protocol system.

independently. Process M acts as a junction that allows asynchronous access to shared register actions. This is an example showing how CIRCAL's simultaneous action transitions can be used to define a stateless process that encapsulates a simple relation, in this case mapping separate process write actions onto an internal register write action.

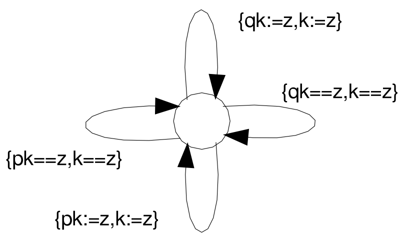

Fig. 7. Register multiplexer process M

– All the register actions are abstracted from the *FischerSystem* process. What remains visible to the outside are the transition-marking actions for each worker process.

The XCircal code that defines an (untimed) Fischer system with two workers is:

```
FischerSystem <- (FischerP * FischerQ * Register * Multiport) -
```

```
(pksetz pkeqz qksetz qkeqz kneqp kneqq kneqz
 ksetp ksetq ksetz keqp keqq keqz)
```

3.3 Process Models of Timing Constraints

Since the Fischer protocol relies on timing constraints among its worker processes for correct operation, the untimed model of the Fischer protocol presented above is inadequate. Specifically, after indicating its intention to enter its critical section, a worker process P needs to wait "long enough" to ensure that all other workers are either (a) back at the start state, or (b) have already followed P, usurped Ps place, and have sent P back to the start state.

One approach to modelling the timed behaviour of a Fischer protocol system is to equip each worker with its own local clock, and predicate certain transitions on clock values. This is the Timed Automata approach, described for example in [12].

Applying our methodology, we express the "workers wait long enough in state C" condition purely in terms of the sequences of events allowed (or disallowed) by timing interval restrictions. The condition that worker P waits long enough for worker Q can be enforced by the requirement that the interval between the *qab* and *qbc* event be longer than the interval from *qab* to any *pcd* event. In other words, once a *qab* event has occurred, a *pcd* event may not occur (i.e. P must wait) until *qbc* has occurred. A process that enforces this constraint is shown in Figure 8.

The process shown in Figure 8 is an instance of a family of processes which have the effect of disallowing a specific sequence of actions. In this case the process disallows the subsequence $qab \rightarrow pcd$ in the set of all sequences of events drawn from $\{qab, qbc, pcd\}$. Constraints based on disallowing longer sequences of events can easily be generated, using an algorithm based on the Knuth-Morris-Pratt string searching algorithm [11].

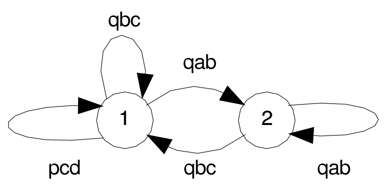

Fig. 8. Fischer protocol timing constraint between processes P and Q.

This process expresses the constraint that requires P to wait for worker Q. To fully express the timing constraints for the whole system a constraint process is needed for every ordered pair of distinct workers, so $n(n-1)$ constraint processes are required for an n worker system. [2] For our two-worker example, the two instantiated constraint processes are:

```
TimingPQ <- TimingConstraint(pcd,qab,qbc)
TimingQP <- TimingConstraint(qcd,pab,pbc)
```

Applying these timing constraints to our untimed system yields the process:

```
TimedFischer <- FischerSystem * TimingPQ * Timing QP
```

The relative timing interval constraint technique employed here is more generic and less concrete than the use of clock variables in timed automata. Unlike clock variables, relative timing interval constraints do not directly suggest an implementation in terms of local clocks used by concurrent processes. It is interesting to note that the nature of the CIRCAL composition operator allows the timing interval constraint processes given in this subsection to be replaced by an alternative set of processes which express the necessary timing constraints in another idiom – as discrete local clocks for each process for example – without requiring modification to either the worker processes or the correctness property process (described in the next section).

3.4 Process Models of Behavioural Properties

The mutual exclusion property says that only one process may be in its critical section at a time. In our model, this property can be expressed in terms of the events that mark each worker process entering (*cd* events) and leaving (*da* events) its critical section. For a system of n worker processes, a simple $n + 1$ state property process indicates what sequences of events are compatible with the mutual exclusion property. The two-worker version of this property process is show in Figure 9.

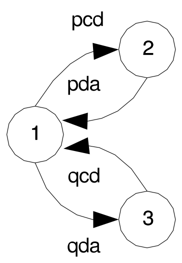

Fig. 9. Mutual-exclusion property for two processes P and Q.

[2] By using slightly more complex processes, this can be reduced to n constraint processes for an n worker system. There are a number of different constraint processes that correctly enforce the Fischer protocol's timing requirements; the constraint process used here is one of the simplest.

3.5 Verification

The behaviour of a protocol system (including the shared register and timing constraint processes) for two or three workers is simple enough that the mutual exclusion property can be verified by printing out the critical section behaviour and inspecting it. Figure 10 shows the complete behaviour of a two-worker protocol system, with all actions except critical section actions hidden by the abstraction operator. It clearly conforms to the two-worker mutual exclusion property (the two being in fact identical).

```
Start State     Transition Label      End State
-----------     ----------------      ---------
     1               ["Pcd"]       ->      2
     1               ["Qcd"]       ->      3
     2               ["Pda"]       ->      1
     3               ["Qda"]       ->      1
```

Fig. 10. Critical section behaviour of worker, register and timing constraint processes.

Larger systems can be verified by using the technique described in section 2. Treating the mutual exclusion property as a safety condition (it expresses the *allowable* behaviours for a correct system), our correctness condition is

$$\texttt{TimedFischer * MutexProperty} \cong \texttt{TimedFischer}$$

where $TimedFischer$ is the system model process (including timing constraint processes) and $MutexProperty$ is the mutual exclusion property for the appropriate number of workers. Using the current generation of CircalSystem tools we have successfully performed this verification for systems of at most 5 workers.

4 Discussion

The ability of a modelling formalism to accurately represent timing information is becoming increasingly significant when designing a range of complex, concurrent systems such as asynchronous digital logic circuits [7, 20, 4] and network communication protocols [3].

In this paper we present a practical modelling and verification methodology which exploits the characteristics of a specific process algebraic composition operator. This approach differs from existing methodologies.

Rather than augment an automata model with clocks and timed transitions, temporal constraints are expressed as relative timing interval constraint processes. The primary requirements for use of the interval timing constraints technique are that (a) the critical states and time intervals in the system are cleanly delimited by actions, and that (b) timing constraints can be expressed as relationships between these intervals. For the example in this paper the constraint relationship takes the form a relative differences in interval duration for two intervals that start at the same moment. Other timing properties known to be amenable to expression as interval constraints include intervals required to be overlapping (or non-overlapping); and intervals required to be entirely contained within other intervals. Cowie [6] describes a methodology for translating a class of constraints normally expressed in an interval algebra to constraint processes.

Our methodology contrasts with previously described methodologies for modelling and verifying timing-dependent systems. Timed Automata [1] are formal automata models which include a real-valued local clock value for each process, and allow transitions to be predicated on clock values. The Uppaal and TVS systems are toolsets that include model-checkers for Timed Automata (Fischer protocol verification examples for each are reported in [12] and [2]).

Timed process algebra [17] extend untimed process algebra (such as CCS, CSP or ACP) with operators for expressing the possibility that transitions may be delayed a certain period after they become active. [21] describes the Fischer protocol in terms of a discrete-time and a real-time process algebra.

A third approach to modelling and analysing timed systems is to introduce timing components (e.g. clock processes and "clock tick" actions) into an untimed framework such as an untimed process algebra. The result is a system somewhat similar to discrete-time process algebra, but where the timing constraints are expressed as model components rather than in language of the surrounding formalism. An example of this is given in [3].

The methodology described in this paper contributes to the state of the art of formal methods by providing (1) an alternative technique for defining constraints in timing-critical systems: separate constraint processes which define relationships between critical timing intervals; and (2) an alternative technique for verifying properties in such systems: the composition-based verification technique, which does not require the introduction of temporal logic, model-checking or refinement relation checking.

This elegant approach does not introduce any additional mathematical concepts, and capitalises on a concept already very familiar to engineers: processes described by state transition diagrams. The methodology presented in this paper thus presents a lower barrier of entry to design engineers that would otherwise be unlikely to adopt formal methods techniques, and provides an additional set of tools for the experienced formal methods practitioner.

We see this methodology being used, as in the Fischer protocol example, to analyse complex systems in terms of sequences of critical events. Experimentation and modelling at this level can be used to develop correct algorithms and protocols.

We would like to acknowledge that this research has been funded in part by the Australian Research Council.

References

1. Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.
2. Marcel Ammerlaan, Ronald Lutje Spelberg, and Hans Toetenel. XTG - an engineering approach to modelling and analysis of real-time systems. In *10th Euromicro Workshop on Real Time Systems*, pages 88–97. IEEE Computer Society Press, June 1998.

3. A. Cerone, A. J. Cowie, G. J. Milne, and P. A. Moseley. Modelling a time-dependant protocol using the Circal process algebra. *Lecture Notes in Computer Science*, 2102:124–138, 1997.
4. A. Cerone, D. A. Kearney, and G. J. Milne. Integrating the verification of timing, performance and correctness properties of concurrent systems. In *The International Conference on Application of Concurrency to System Design*, pages 109–119. IEEE Computer Society Press, 1998.
5. A. Cerone and G. J. Milne. A methodology for the formal analysis of asynchronous micropipelines. In *International Conference on Formal Methods in Computer-Aided Design (FMCAD'00)*, number 1954 in Lecture Notes in Computer Science, pages 246–262. Springer-Verlag, 2000.
6. Alex Cowie. *The Modelling of Temporal Properties in a Process Algebra Framework.* PhD thesis, University of South Australia, 1999.
7. S. B. Furber and P. Day. Four-phase micropipeline latch control circuit. *IEEE Transactions on VLSI Systems*, 4(2):247–253, June 1996.
8. C. A. R. Hoare. *Communicating Sequential Processes.* International Series in Computer Science. Prentice Hall, 1985.
9. Joel Kelso. Proof of the soundness of the concurrent composition property checking technique. Technical Report Report-05-NNN, School of Computer Science and Software Engineering, Univeristy of Western Australia, 2005.
10. Joel Kelso and George Milne. The prototype Haskell CIRCAL system. `http://www.csse.uwa.edu.au/FormalSpecification/HaskellCircal/`, 2003.
11. D.E. Knuth, J.H. Morris, and V.R. Pratt. Fast pattern matching in strings. *SIAM Journal on Computing*, 6(1):323–350, 1997.
12. K. J. Kristoffersen, F. Laroussinie, K. G. Larsen, P. Pettersson, and Wang Yi. A composition proof of a real-time mutual exclusion protocol. Technical Report RS-96-55, Aalborg University, Denmark, 1996.
13. Leslie Lamport. A fast mutual exclusion algorithm. *ACM Transactions on Computer Systems*, 5(1):1–11, 1987.
14. George J. Milne. CIRCAL and the representation of communication, concurrency and time. *ACM Transactions on Programming Languages and Systems*, 7(2):270–298, 1985.
15. George J. Milne. *Formal Specification and Verification of Digital Systems.* McGraw-Hill, 1994.
16. Robin Milner. *Communication and Concurrency.* International Series in Computer Science. Prentice Hall, 1989.
17. Xavier Nicollin and Joseph Sifakis. An overview and synthesis on timed process algebras. In Kim Guldstrand Larsen and Arne Skou, editors, *Computer Aided Verification, 3rd International Workshop, CAV '91*, volume 575 of *Lecture Notes in Computer Science*, pages 376–398. Springer, 1992.
18. G. Plotkin. Structural operational semantics. Technical Report DAIMI FN-19, Aahus University, 1981 (reprinted in 1991).
19. A. W. Roscoe. *The Theory and Practice of Concurrency.* Prentice Hall, 1997.
20. Ivan E. Sutherland. Micropipelines. *Communications of the ACM*, 32(6):720–738, 1989.
21. J. Vereijken. Fischer's protocol in timed process algebra. Technical Report CSR 94/32, Eindhoven University of Technology, Computing Science Department, 1994.