TECHNICAL REPORT

COMPUTER SCIENCE

A COMPARISON OF TIMED CSP AND TIME BASIC NETS FOR THE SPECIFICATION OF TIME CONSTRAINTS

Maria Kutar

Report No 311

May 1998

ABSTRACT

A level crossing case study was specified in timed CSP. The same case study was then specified using Time Basic Nets. To provide an element of control a second case study, the secure room, was specified first using Time Basic Nets and then timed CSP. Neither Timed CSP nor Time Basic Nets were adequate for the specification of the time constraints contained within the level crossing case study, although both techniques proved suitable for the specification of the secure room. The report concludes with a critique of the two techniques, both as 'ordinary' specification languages and in relation to their suitability for the specification of time constraints.

A COMPARISON OF TIMED CSP AND TIME BASIC NETS FOR THE SPECIFICATION OF TIME CONSTRAINTS

INTRODUCTION

The report is set out as follows:

1 The case studies used. *(p2)*

2 The timed CSP specifications.*(p6)*

3 The Time Basic Nets specifications.*(p17)*

4 Comparisons and conclusions.*(p24)*

It is assumed that readers of this report have a basic knowledge of both Communicating Sequential Processes (CSP) and Petri Nets. This report includes introductions to the timed extension to CSP*(p5)* and to Time Basic Nets*(p15)*, although it should be noted that the introduction to timed CSP covers only the part of timed CSP used in this report and is therefore not a comprehensive guide.

1 CASE STUDIES

Two case studies were used in this study of timed CSP and Time Basic Nets (TBN). Firstly, a level crossing case study [BAI91], originally taken from [GOR87] was adapted to incorporate several response time constraints. This was specified using CSP and timed CSP, (a full account of which is available at [KUT97]) and then using TBN. When the TBN specification was produced it was felt that the previous use of the case study had distorted the re-analysis. It became difficult to consider the level crossing case study in terms of the system itself rather than in terms of the CSP specification and so a second case study, the secure room [SEL96] was specified to provide an element of control, with the order reversed so that the secure room was

specified first in TBN and then in timed CSP. The basic requirements for each case study are laid out below.

1.1 THE LEVEL CROSSING

The system described below is a simple level crossing, where a single train or car may pass through during any one operation of the system. The description and diagram (Fig. 1) show the system where a train passes from right to left, and a car from top to bottom. It may be assumed that trains and cars passing in the opposite direction behave in an identical manner. Fig. 1 shows the layout of the crossing.

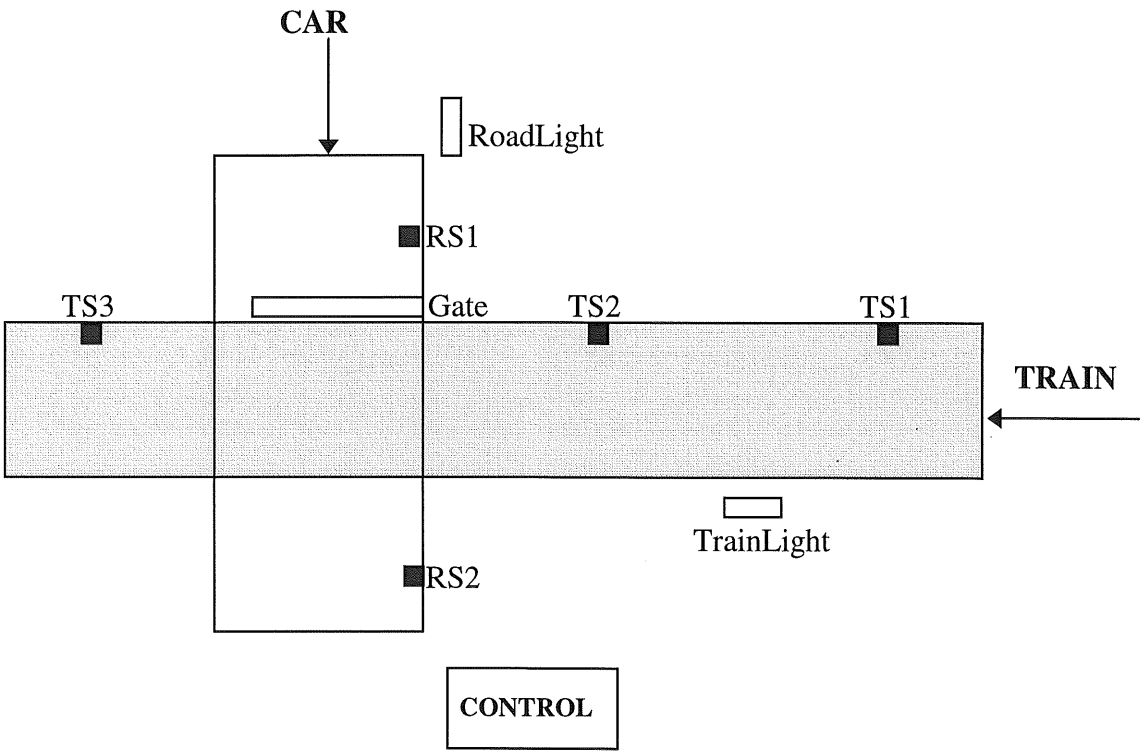


Fig. 1

- The driver will approach the TrainLight which should be showing red. As he does so the train will necessarily pass over the sensor TS1, alerting control that a train is approaching.

- Control will ensure that the RoadLight is turned to red, the road gates are closed and the TrainLight turned to green within 4 minutes of the train passing over the sensor TS1, allowing time for Control to clear the crossing of cars which are counted in and out of the crossing using the road sensor. The road sensor is the combination of sensors rs1 and rs2.

- Once the TrainLight has turned to green the driver will proceed through the crossing and in doing so, the train will pass over the sensors TS2 and TS3. Control will ensure that, within 5 seconds of the train passing over TS2 the TrainLight will be turned to red, and that within 1 minute of the train passing over the sensor TS3, the gates are opened and the RoadLight turned to green.
(N.B. this assumes that the train is assumed to have passed over a sensor when the *whole* train has passed over.)

1.2 THE SECURE ROOM

The secure room is a room to which a user may only gain access after he has entered a valid user id and personal password on the keypad situated next to each entrance to the room. For valid access codes the door is unlocked for 3 seconds during which time the user may enter the room. A sample usage scenario would be as follows:

- user enters id on keypad
- user enters personal password on keypad
- system validates user id and password against central database
- for valid access codes, door is unlocked for 3 seconds during which user can enter
- after 3 seconds expire, door is locked again

Other basic scenarios which the system is required to deal with are:
unsuccessful access
abandoned access
break-in detection
(For the purposes of this report database functions of the system such as adding and removing users are ignored.)

2 TIMED CSP

Timed CSP was selected as a suitable specification technique as it is designed expressly for use in real-time systems, and therefore should be an ideal language in which to specify the timed sections of the system. It is a direct extension of the original CSP [SCH96]. For the level crossing specification the only timed CSP construct which was used was the timeout, the operation of which is described below. Readers requiring a more comprehensive introduction to timed CSP are directed to [DAV94]

TIMEOUT

The timeout construct, $P \triangleright^{t} Q$ initially behaves as process P. If an event from the alphabet of P is carried out before time *t* has elapsed, then the choice is resolved in favour of P, which will continue to operate whilst Q is discarded. If no event has been carried out before *t* has elapsed, then the choice is resolved in favour of Q as the timeout has occurred. This may be illustrated by the process:

MAKE_CALL =

$$(\text{call\_answered} \rightarrow \text{SPEAK\_TO\_CALLER}) \triangleright_{30} \text{ABANDON\_CALL}$$

where someone making a telephone call is prepared to wait for 30 seconds for the call to be answered. If the call is answered within this time period the choice is resolved in favour of the process SPEAK_TO_CALLER. If, however there is no response then after 30 seconds the timeout will occur and the person will abandon the call.

Timeouts are most commonly used in error detection - if some expected action does not occur within the given time frame then an exception will occur. Less commonly, it may be used where the expectation is that the timeout will occur, as is shown in this fragment from a wedding service:

$$(\text{speak\_now} \rightarrow \text{DISRUPTION}) \triangleright_{10} \text{FOREVER\_HOLD\_PEACE}$$

Here, it is expected that the timeout will occur, but the opportunity to prevent it is provided. (Example taken from [SCH96])

2.1 SPECIFYING THE LEVEL CROSSING TIME CONSTRAINTS IN TIMED CSP

The level-crossing case study contains three time constraints which are as follows:

1. The RoadLight must be turned to red, the road gates closed and the TrainLight turned to green within 4 minutes of sensor ts1 being triggered. For obvious reasons the events must be carried out in this order.
2. The TrainLight must be turned to red within 5 seconds of sensor ts2 being triggered.
3. The road gates must be opened and the RoadLight turned to green within 1 minute of the sensor ts3 being triggered.

An initial specification of the level crossing in the 'untimed', original form of CSP was produced. Part of this specification was the process TRAIN, shown below.

TRAIN:

α TRAIN = { trainapproach, trainin, trainout, flipred, flipgreen, wait, ts1, ts2, ts3, open, close, go_red, gogreen}

TRAIN = (trainapproach $\rightarrow$ ts1 $\rightarrow$ go_red $\rightarrow$ close $\rightarrow$ flipgreen $\rightarrow$ trainin $\rightarrow$ ts2 $\rightarrow$ flipred $\rightarrow$ ts3 $\rightarrow$ trainout $\rightarrow$ open $\rightarrow$ gogreen $\rightarrow$TRAIN)

All of the time constraints in the case study can be incorporated into this particular process but a number of difficulties were encountered. The first time constraint in the case study indicates that the following events need to occur within four minutes of sensor ts1 being triggered:

-RoadLight turned to red (go_red)
-road gate closed (close)

-TrainLight turned to green (flipgreen)

This is rather problematic in relation to the timeout operator as the construct may only be based on a single event being carried out within the given time and our requirement is that three events take place within this time. There are at least two possible solutions to this problem. The first solution would be to base the timeout on the operation of the last of these events. It is a workable solution in that we may assume that if the TrainLight is turned to green, then the preceding events, namely *go_red* and *close* must have taken place. If they had not the process would not have reached the point where the TrainLight turns green.

In order to construct the specification around this solution the TRAIN process must first be split into two processes. This is so that in order for the timeout to be properly constructed, it is reliant on the first event of the process taking place, in this case the *flipgreen* event. The two processes are:

αTRAIN1 = { trainapproach, ts1, go_red, close, ✓ }
αTRAIN2 = { flipgreen, flipred, trainin, trainout, ts2, ts3, open, gogreen }

TRAIN1 = (trainapproach → ts1 → go_red → close → ✓)
TRAIN2 = (flipgreen → trainin → ts2 → flipred → ts3 → trainout → open
→ gogreen → TRAIN1)

TRAIN = TRAIN1 ; TRAIN2

The timeout would then be used to ensure that the flipgreen process occurs within the specified time, reporting an error if it does not.
(The time units used throughout the timed CSP specifications are such that (num.)s represents (num.) seconds and (num.)m represents (num.) minutes)

$$\text{TRAIN2} \rhd_{4m} \text{GO\_RED\_ERROR}$$

However, this does not specify the problem with a great deal of accuracy. The first problem is that this actually states that the event flipgreen will occur within 4 minutes of the final event of TRAIN1 occurring, namely the *close* event. In addition this will only report an error if the *flipgreen* event does not occur. Whilst it is unrealistic to expect that the failure of any single specified event will be reported as an error it is reasonable to expect that a number of key events, in this case any of those integral to the time constraint, will be detected and reported. Using the above solution, if for example the RoadLight was to turn to red but the gate did not close, then the process would simply become 'stuck' - not proceeding any further. After some consideration it was decided that this solution was unworkable if the specification was to provide the basis of a functional system and therefore a different path had to be taken.

A second solution to this problem is to treat this time constraint as three separate constraints, with the allocated time divided between the three. Again, the train process needs to be divided into separate processes, and as the remaining time constraints also require division of the same process, they are incorporated into the model at this stage. The remaining time constraints are that the TrainLight is turned to red within five seconds of the sensor ts2 being triggered, and that the gate must be opened and the RoadLight turned to green within one minute of the sensor ts3 being triggered. The final time constraint is similar to the first in that it is made up of two events, and as in the first case, it would be a more useful system if a failure could be reported for the specific event. For this reason the final time constraint is modelled as two constraints, with the time divided between the two.

The time constraints were divided thus:
First constraint - that the RoadLight is turned to red, the road gate closed and the TrainLight turned to green within 4 minutes of ts1 being triggered.

- The RoadLight will be turned to red within two minutes of sensor ts1 being triggered.
- The road gate will be closed within one minute of the RoadLight being turned to red.
- The TrainLight will be turned to green within one minute of the road gate being closed.

Final constraint - that the road gate will be opened and the RoadLight turned to green within one minute of the sensor ts3 being triggered.

- The road gate will be opened within 50 seconds of sensor ts2 being triggered.
- The RoadLight will be turned to green within 10 seconds of the road gate opening.

(As the second time constraint was dependent on only one event it did not need to be adapted.)

The key events which need to occur within the specified time, to avoid the timeout from occurring are :

First time constraint : go_red, close, flipgreen

Second time constraint: flipred

Third time constraint : open, gogreen

The train process was divided into these processes:

TRAIN1 = (trainapproach → ts1 → ✓)

TRAIN2 = (go_red → ✓)

TRAIN3 = (close → ✓)

TRAIN4 = (flipgreen → trainin → ts2 → ✓)

TRAIN5 = (flipred → ts3 → trainout → ✓)

TRAIN6 = (open → ✓)

TRAIN7 = (gogreen → TRAIN1)

The corresponding alphabets for these processes :

αTRAIN1 = { trainapproach, ts1, ✓ }

αTRAIN2 = { go_red , ✓}

αTRAIN3 = { close, ✓ }

αTRAIN4 = { flipgreen, trainin, ts2, ✓ }

αTRAIN5 = { flipred, ts3, trainout , ✓}

αTRAIN6 = { open, ✓ }

αTRAIN7 = { gogreen }

These processes may then be combined using the sequential composition operator. At the same time the timeout operator is introduced, showing where the timeouts occur, and indicating an error should the constraints not be met.

TRAIN = (TRAIN1 ; TRAIN2 $\triangleright_{2m}$ GO_RED_ERROR ; TRAIN3 $\triangleright_{1m}$ CLOSE_ERROR ; TRAIN4 $\triangleright_{1m}$ FLIPGREEN_ERROR ; TRAIN5 $\triangleright_{5s}$ FLIPRED_ERROR ; TRAIN6 $\triangleright_{50s}$ OPEN_ERROR ; TRAIN7 $\triangleright_{10s}$ GOGREEN_ERROR)

Obviously the newly included processes which occur when the timeout operator is triggered must be defined. This is shown in full in [KUT97] but all new 'error' processes allow for the system to attempt a single retry of the failed event, and should this also be unsuccessful an 'alert' process is triggered to notify the system operator of the failure. Control at this stage then passes from the system to the operator and therefore remains undefined in the specification. An example of the error process is the GO_RED_ERROR:

GO_RED_ERROR = RETRY_GO_RED $\triangleright_{2m}$ GO_RED_ALERT
αRETRY_GO_RED = { go_red }
RETRY_GO_RED = (go_red $\rightarrow$ TRAIN3)

Thus if the retry is successful, the relevant TRAIN process is picked up and operation of the system may continue as normal.

Modelling the time constraints contained in the level crossing system initially seemed rather cumbersome. Although one of the operators appeared to closely match the requirements, it was only through adaptation of the case study that a specification could be produced. Therefore it must be noted that none of the timed operators allowed specification of the system outlined in the description of the case study.

The adaptation of the case study gives a system where the initial time constraints may still be met, but it has to some extent distorted the system. The first time constraint was that within four minutes of the depression of sensor ts1, the road light would be turned to red, the road gates closed and the TrainLight turned to green. The adaptation ensures that the road light is turned to green within two minutes of the sensor ts1 being triggered. The road gates will be closed within one minute of the road light being turned to red and the TrainLight will be turned to green within one minute of the road gates being closed. Although most operations of the system would probably execute in accordance with the original requirements, there are a number of circumstances where operations of the system which would be acceptable under the requirements would not be accepted by the system specified. This is because we have had to allocate the time allowed for the three combined events separately for each event. Under the required system, it is possible, for example, for no events to take place for the first three minutes and the overall time constraint to still be met. This would cause at the very least an error to occur in the specified system. The same problems apply to the final time constraint in the requirements which has had to be divided into two.

Despite the fact that timed CSP could not be used to specify the original requirements it is still possible to evaluate the operator that was used. The key role of the timeout operator is in error detection and this is mirrored in the way that the specification of the time constraints developed. It is obviously an error if time constraints integral to the system are not met and the time constraints are critical to the operation of the system. A level crossing system which might keep cars or trains waiting for long periods of time is of no use to anyone. A retry is included in order to make the system more efficient. This in itself showed a shortcoming of CSP. Although CSP includes a method to restart a process, this restarts the process from its first event. There is no way of simply retrying a failed event, (without adapting the processes as we have done above) which would be particularly useful. Thus although the operator is aimed at use in error detection, the language gives no assistance in specifying how we might recover from those errors. An operator which provides assistance in recovery from errors as well as in error detection would be a useful

addition. However, it must be said that the timeout operator allows clear specification of a time constraint based on a single event and allows us to specify, albeit in a rather roundabout way, what the system should do if an error occurs.

One of the key difficulties encountered when specifying the time constraints was the problem of incomplete requirements. Whilst the requirements lay out the required behaviour of the system they do not extend to the requirements of system behaviour when an error occurs. Because the boundaries of the system are unclear it was difficult to assess what is required of the system in terms of responding to errors. It seems clear that the system should not be required to prevent further trains from approaching the crossing, but it would not be unacceptable to require that it communicates either with signalling personnel or even with the signalling system. The 'alert' processes remained unspecified in this study mainly as a result of incomplete information about the boundaries of the system.

It would have been possible, by allocating a time to each event, representing the time it takes to be performed, and running traces on the specification, to prove that the system meets the time constraints as laid out in the requirements. This route was not chosen, largely because if it had been there would have been no way of specifying how the system should act if the constraints are not met. In addition this approach is rather like working backwards - we may design a system and then prove that it meets the requirements but this is rather different to constructing a system *to* those requirements. Using this method does not actually model the time constraint in the same way that the timeout operator does, it merely gives us a system that we can show works under the conditions imposed. An additional problem with this approach is that we could only arbitrarily allocate times that each event takes to occur, and these times would have to include estimated time between the execution of events. This conflicts with the fundamental assumption in CSP that every event occurs instantaneously - this approach would then in fact be an estimation of time spent between events and cannot be considered to be a model of the time constraints.

2.2 SPECIFYING THE SECURE ROOM IN TIMED CSP

The specification of the single timing requirement of the secure room in CSP is a straightforward task. If the room were to be specified as a single CSP process we would have the following:

αSECURE_ROOM = {enter_id, enter_password, validate, unlock_door, lock_door}

SECURE_ROOM =
(enter_id $\rightarrow$ enter_password $\rightarrow$ validate $\rightarrow$ unlock_door $\rightarrow$ lock_door $\rightarrow$ SECURE_ROOM)

The timing requirement could be added using the timeout. This would require the process to be separated at the time constraint to give:

SECURE_ROOM1 = (enter_id $\rightarrow$ enter_password $\rightarrow$ validate $\rightarrow$ unlock_door $\rightarrow$ $\checkmark$)

SECURE_ROOM2 - (lock_door $\rightarrow$ SECURE_ROOM1)

An additional process would be required to form the second part of the timeout, let us call it simply ALERT and leave it undefined. These processes would then be put together thus:

SECURE_ROOM = (SECURE_ROOM1 ; SECURE_ROOM2 $\triangleright_{3s}$ ALERT)

Timed CSP is a particularly suitable notation for the specification of the secure room - the timeout operator is perfectly adequate for specification of the timing requirement. The nature of CSP itself means that it is possible to model the secure room very concisely, without ambiguity. A simple choice operator would allow unsuccessful access to be modelled. The nondeterministic nature of the internal choice

operator means that external choice would need to be used (external choice is choice made externally to the system, by the environment). It is debatable whether this would be an entirely accurate depiction of the reality, although it could be argued that the choice is essentially made by the user in entering the id and password.

Addition of the requirement that the system may deal with abandoned access may be achieved through the use of further timeouts, although this will require the process to be split so that each event requiring action from the user forms the first event of a process. Break-in detection could be incorporated through the use of a conditional [HOA87] which would allow, for example, only three unsuccessful access attempts before raising the alarm.

Much of this ease of use must be attributed to the fact that the secure room case study is a particularly straightforward case study. The only time constraints included are of a single type, namely response time, and this holds true even when further time constraints are included to incorporate abandoned access.

3 TIME BASIC NETS

Time Basic Nets (TBN) is a temporal extension to general Petri Net theory and involves the following:

- Associating certain relative time values, referred to as *time offsets*, with each transition as its time parameters.
- *Time-stamping* of tokens for determining the enabling times of transitions.
- Associating an interval of clock times with each transition, giving the permitted, or the mandatory firing times of transitions. This interval is defined as a function of time stamps on tokens at input places and time offsets mentioned above.

[NIS97]

3.1 INTRODUCTION TO TIME BASIC NETS

For a full description of TBN readers should see [NIS97]. This introduction will however give the reader sufficient knowledge to follow the rest of the report. In TBN tokens are 'time-stamped'. A time stamp is a record carried by each token of the firing time of the transition which created it. The allowable firing times of transitions form the *time condition*, which is a set of clock times or absolute time values. The time condition is formed in terms of the time stamps of its input places and certain temporal parameters associated with the transition. The temporal parameters form the *time offsets* of the transition and are usually specified as lengths of time, i.e. relative times. The time offsets are fixed parameters of the transitions and therefore do not change with time.

A transition is said to be enabled if, and only if, the following hold:

- each input place contains the necessary number of tokens. (This condition is the same as that in untimed Petri nets)

-there are time values in the time condition which are greater than or equal to the largest time stamp on the input tokens.

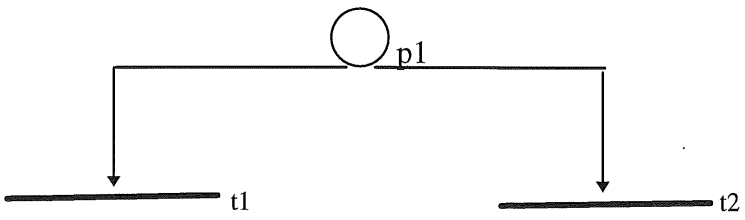
The *firing period* of a transition is defined as the clock time interval whose values are drawn from the time condition of the transition, but are greater than or equal to its enabling time.

The execution of transitions follows these rules:

- only enabled transitions may fire. The firing of a given transition must take place within its firing period.
- As a transition fires, the enabling tuple is removed from the input places. (This is the same in untimed Petri Nets.)
- Firing of a transition delivers to its outputs the appropriate number of tokens, each time-stamped with its firing time and according to the arc weight following the rules of untimed Petri Nets.
- The remaining rules of untimed Petri Nets apply.

EXAMPLE:

If we take the following Petri net fragment:



and put it together with the table below, showing the TBN timing aspects required of the system we have what is considered to be a timed model of the system.

Transition	Time Offsets	Input Places	Time Condition
t1	t_x	p1	(time(p1), time(p1) + t_x)
t2	t_x	p1	(time(p1) + t_x, ∞)

The table should be read in conjunction with the Petri Net as follows:

Transition t1, whose input place is p1, (and whose token will therefore be time stamped time (p1)), may fire between the times, time(p1), and time(p1) + t_x (this is,

therefore, the firing period of transition p1). If this time is exceeded then transition t1 may not fire but transition t2 may, as the condition for t2 to fire is that time is greater than time(p1) + t_x - i.e. satisfies the time condition (time(p1) + $t_{x, \infty}$).

3.2 TBN AND THE LEVEL CROSSING CASE STUDY

The level crossing case study was not fully specified in TBN as on beginning the specification it became apparent that the previous analysis of the case study had influenced the approach taken. However, it is possible to see how TBN could have been used from parts of the specification. The 'train' part of the specification which is roughly equivalent to the TRAIN process in the CSP specification is shown overleaf. The level crossing diagram is reproduced below.

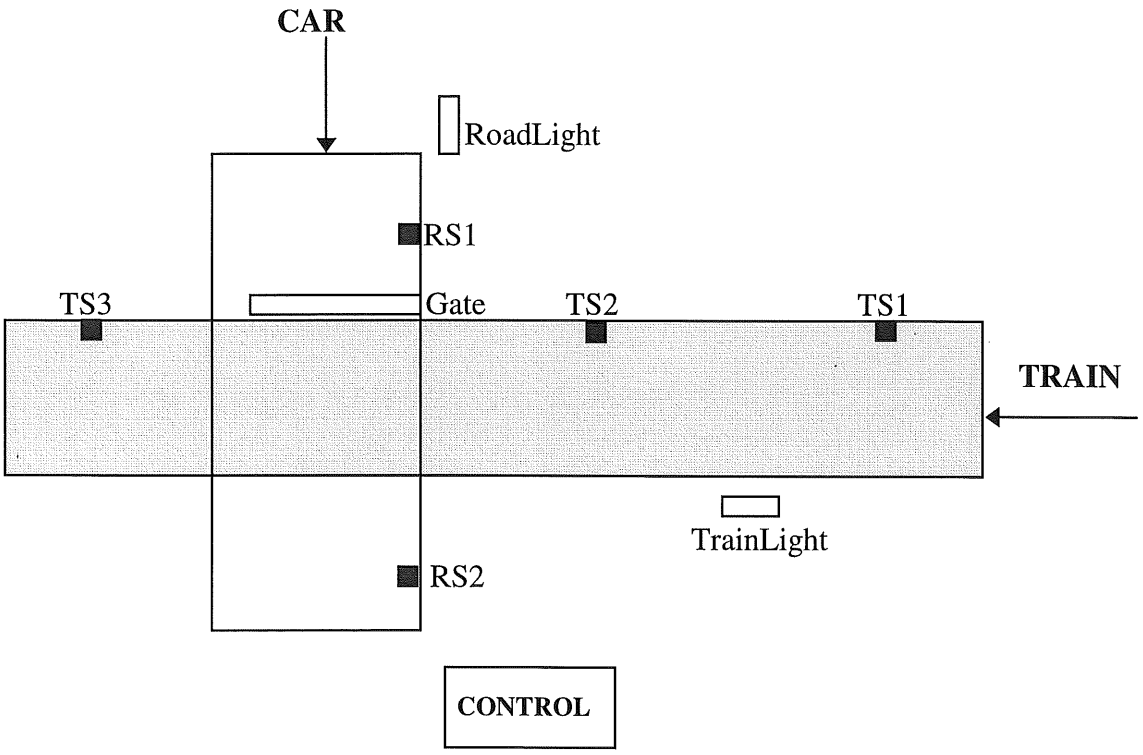
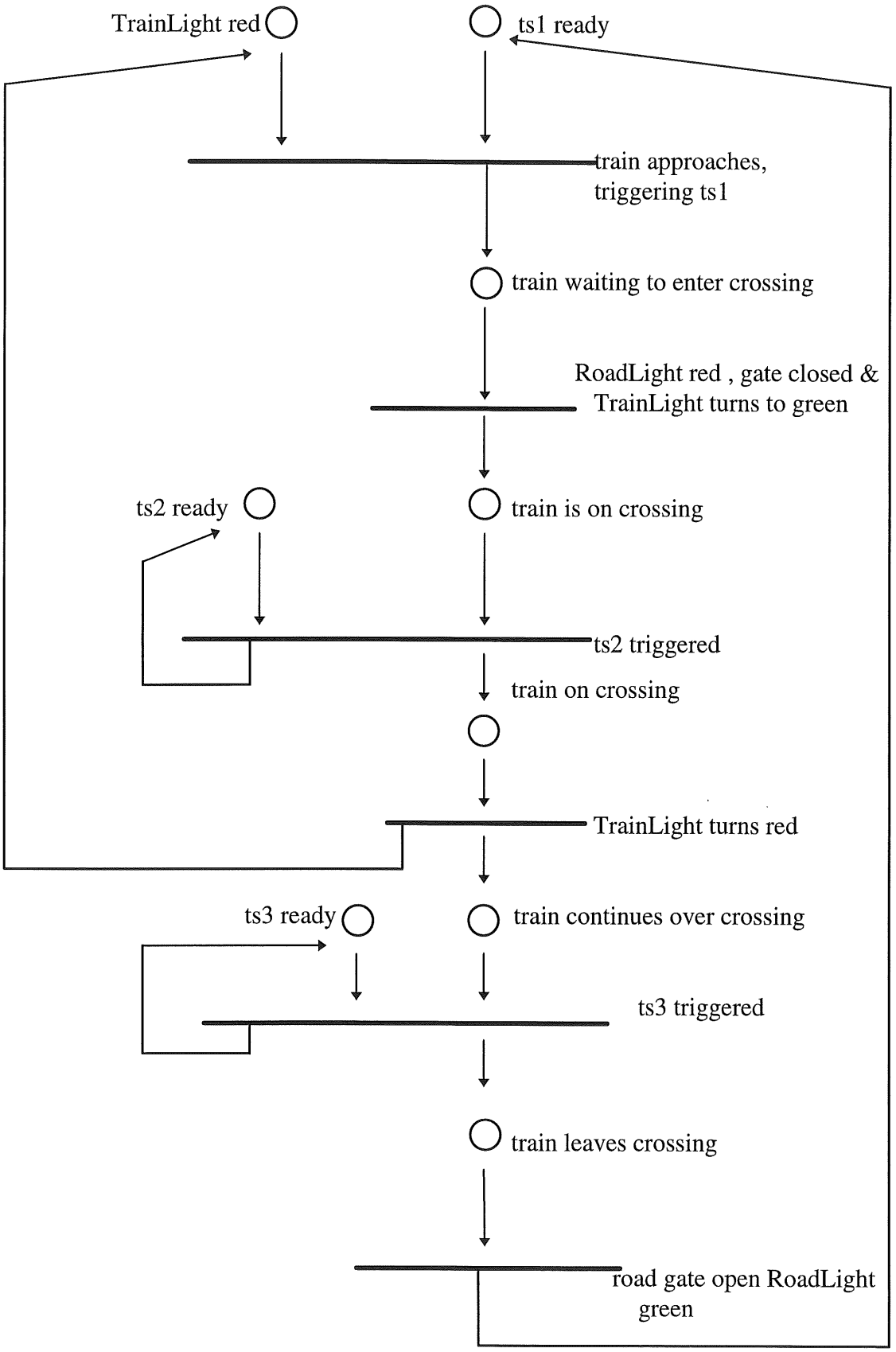


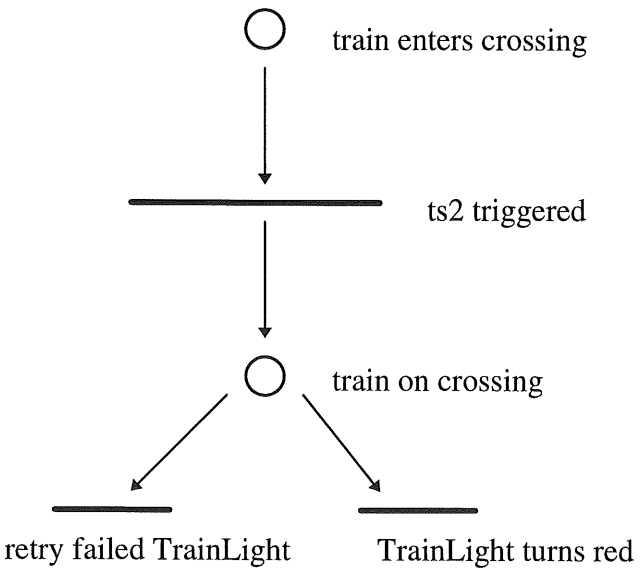
Fig. 1


TrainLight red
ts1 ready
train approaches, triggering ts1
train waiting to enter crossing
RoadLight red , gate closed & TrainLight turns to green
ts2 ready
train is on crossing
ts2 triggered
train on crossing
TrainLight turns red
ts3 ready
train continues over crossing
ts3 triggered
train leaves crossing
road gate open RoadLight green

This fragment may be read in conjunction with the TBN timing table to give a model of the timed parts of the system.

Transition	Time Offsets	Input places	Time Condition
RoadLight red, gate closed & TrainLight turns green	t_{4m}	train waiting to enter crossing (p1)	(time(p1), time(p1)+ t_{4m})
TrainLight turns red	t_{5s}	train on crossing (p2)	(time(p2), time(p2)+ t_{5s})
road gate open & RoadLight green	t_{1m}	train leaves crossing (p3)	(time(p3), time(p3)+ t_{1m})

As with timed CSP the time units used are such that (num.)s represents (num.) seconds and (num.)m represents (num.) minutes. The time offsets used are a simple representation of the real-time requirement. The input places have been allocated a symbolic name to allow for easier interpretation of the time condition. The fragment above does not, however, show the alternative transitions which would need to be included to allow, for example a retry as in the CSP specification. An example of how this could be done is shown in this fragment:



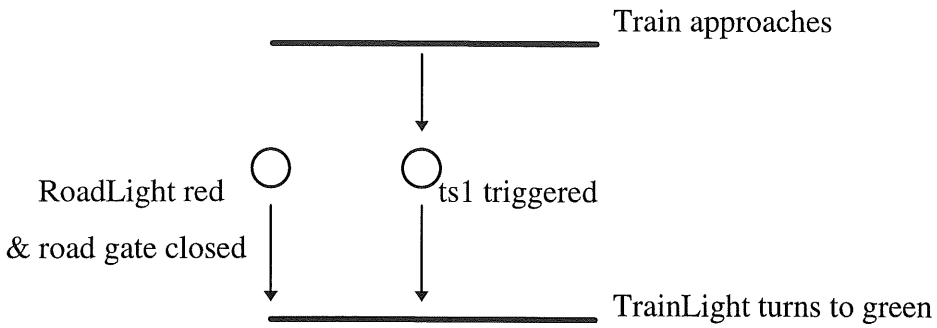
This may be put in conjunction with the following TBN table:

Transition	Time Offsets	Input places	Time condition
TrainLight turns red	t_{5s}	train on crossing (p1)	(time (p1), time (p1) + t_{5s})
retry failed TrainLight	t_{5s}	train on crossing (p1)	(time (p1) + t_{5s} , ∞)

Therefore if the TrainLight fails to turn red within t_{5s} of ts2 being triggered (which is the input transition to the place 'train on crossing', meaning that it is the firing time of this transition which will be time stamped on the token at 'train on crossing'), the transition 'retry failed TrainLight' will fire. This could be extended to include a third potential transition which would alert control to the failure, allocating periods of time for each retry and the alert.

It would initially appear that the timed sections of the level crossing could be specified in TBN. However, the 'train' fragment above includes transitions which include more than one single event. Whilst this would appear to be allowed under the rules of Petri nets [PET81] it does not show the reality of the system in enough detail to be of any real effect. The level crossing system requires a notation which allows the specification of the individual events alongside the ability to base a time constraint on multiple events, and this is not provided for in TBN.

A serious difficulty which was encountered with the use of TBN was the need to construct the basic Petri Nets with TBN in mind. The level crossing was initially modelled in ordinary Petri Nets but when it came to introducing the time conditions much of this had to be remodelled. This was largely due to the fact that the ordinary Petri Nets had been constructed such that events external to the system, over which the system has no control, were modelled as places rather than events. For example, the fragment below required alteration in order to allow a time constraint to be based on the triggering of sensor ts1.



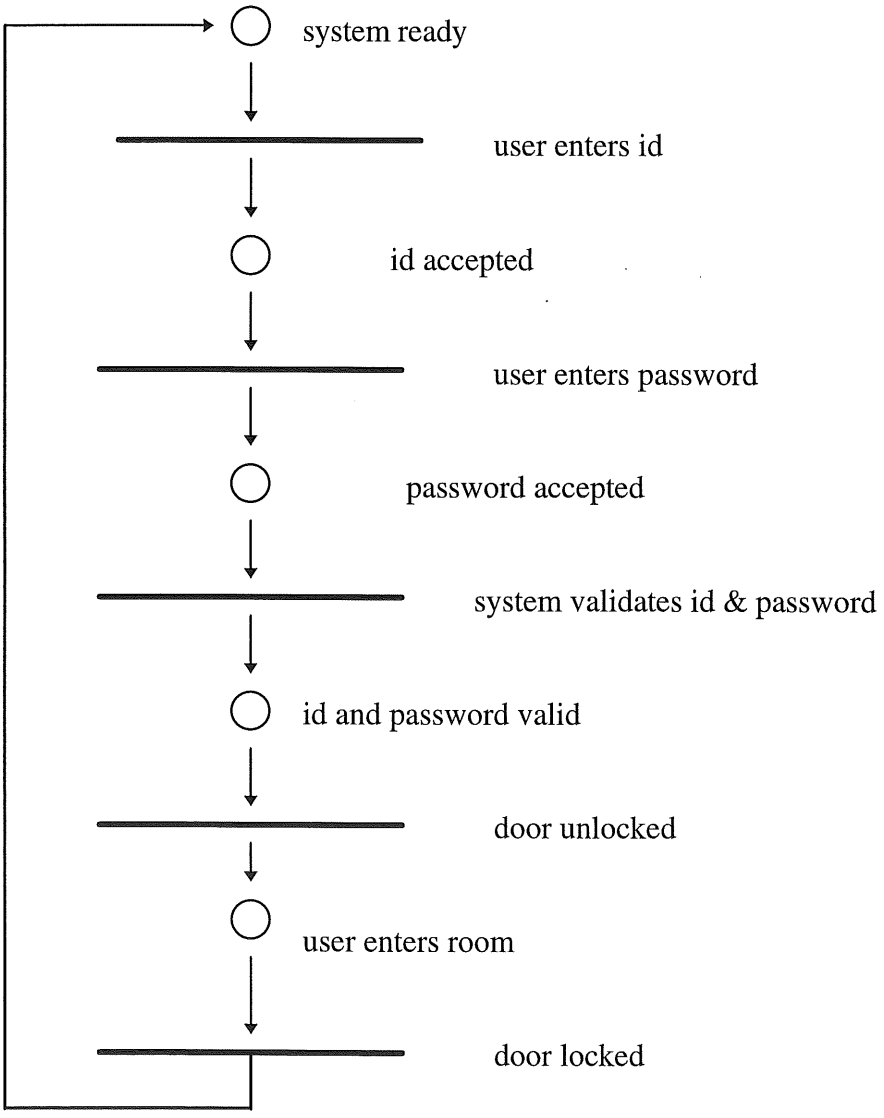
The nature of the level crossing is such that a train approaching the crossing must necessarily pass over the sensor ts1 and therefore this would appear to be a natural postcondition of the event 'train approaches'. However, the time stamp the token receives is the time of the *transition* that creates it and therefore this model could not be used in TBN.

Petri Nets also require the addition of a large degree of artificiality in order that the places may be modelled as conditions and this does distort the model to a certain extent. For example, as can be seen from the Petri Net on page 18, the model now includes places such as 'ts1 ready'. This is implicit in a model of a system and the need to explicitly incorporate it results in unnecessary detail being included for no real purpose. This clutters the Petri Net graph thus reducing clarity. Whilst the graphical nature of Petri Nets aids understanding of a straightforward net, there is a tendency for them to become very cluttered if there is any amount of complexity in the system modelled and the clarity is lost.

Overall therefore, TBN has not proved to be a particularly useful notation in the modelling of the level crossing. There was a certain amount of difficulty in producing a clear and concise model of the system and the time constraints could not be modelled with any real accuracy. As with CSP it would have been possible to do this had the constraints been altered to include only a single event each but the unaltered requirement could not be specified.

3.3 TBN AND THE SECURE ROOM

Producing the TBN model of the secure room was a much more straightforward process than with the level crossing. This must be at least partially attributed to the fact that the secure room case study is much smaller and more straightforward than the level crossing, and also to the fact that the initial analysis was made with TBN in mind. The previous experience of using TBN meant that this Petri Net was constructed such that the event on which the time constraint was based was included as a transition rather than a place, thus avoiding the problems encountered with the level crossing. The Petri Net below shows the secure room without reference to the requirements regarding abandoned or unsuccessful access and break-in detection. These are discussed below.



The net shows a sample usage scenario for the most straightforward case where an authorised user gains access to the room. There is only one timing requirement, shown below.

Transition	Time Offsets	Input places	Time condition
Door locked	t_{3s}	user enters room (p1)	(time(p1), time(p1)+t_{3s})

As with the level crossing, additional transitions would be required to show what the system will do if the time constraint is not met. The inclusion of the additional requirements similarly requires further transitions. Abandoned access can be catered for by the inclusion of an additional time constraint at each stage where the system is dependent on the user, for example where the password is entered. Unsuccessful access may be included by simply including an extra transition alongside 'system validates id & password' which would allow the system to follow the appropriate route through the net.

4 COMPARISONS & CONCLUSIONS

4.1 AS A GENERAL SPECIFICATION LANGUAGE

4.1.1 CSP

CSP was a straightforward language to learn to use. In addition it appears to force a thorough analysis of the problem - in order to form the processes each event must be considered at an early stage and therefore it was harder to forget small details. For example, the construction of the TRAIN process ensured that events such as the triggering of the sensors were included. Additionally, any alterations such as the addition of extra events may be done without too much difficulty as events slot straight into a process. This is in marked contrast to Petri Nets where the addition of a single event would entail the inclusion of both extra transitions and extra places, and quite possibly the adaptation of existing places.

By considering the system as a group of processes the system appears to naturally fall into a set of subsystems, allowing each component part to be considered separately. These parts are then quite straightforwardly brought together using the parallel and sequential operators, giving a model which naturally reflects the system. There was never the feeling that the system had been distorted in order to fit with the language, which can happen with other languages. The only initial drawback is that the non-graphical nature of the language means that a CSP specification is not accessible to someone with no knowledge of the language. However it would be easy to translate into a graphical form resembling the Petri Net diagram if necessary. Although this is not known to be a common approach it would increase the ease of understanding of a CSP specification for someone with no knowledge of the language. Tool support is available and the specification is verifiable through the use of traces.

A good specification language not only allows for the intended behaviour of the system to be clearly laid out, but should encourage a well organised, systematic software development environment. The specification produced should be precise, relatively concise, and take the system under development to the stage where implementation may begin. CSP processes are constructed such that they may be implemented without alteration (for detail see [HOA85]) and the specification of the

level crossing was both unambiguous and concise, containing all the information needed but nothing more. The experience of CSP when specifying the level crossing was that it appeared to satisfy all of these criteria. In addition inconsistencies become apparent at an early stage and changes are easily made where required. Overall CSP is a very useful language for the production of general specifications.

4.1.2 PETRI NETS

Initially Petri Nets appeared to be a straightforward and logical language to use, being based on the use of pre- and post-conditions. However, this feature made the language more difficult to use, and resulted in a certain amount of artificiality having to be included, as was discussed above (p21). Rather than simply concentrating on the events in the system a great deal of attention had to be paid to the conditions which would link them together. Thus even two seemingly straightforward sequential events, such as the entering of an id followed by the entering of a password in the secure room, must be separated by a condition. There was a feeling that the Petri Nets produced did not really reflect the system, which had to a certain extent been distorted by the need for conditions to be included merely to link the events. This difficulty extended to the need to include changes - unlike CSP an extra event cannot just be inserted where necessary - there is the need to both make an extra event fit with existing conditions and also to introduce new conditions thus increasing the artificiality of the model. The nature of the language means that it works very well where the problem can be easily considered in terms of events and pre- and post-conditions (which parts of the case studies could be) but that it becomes very difficult to use where the problem is not suited to the language. Obviously this reduces the range of applications where the language might be considered to be a useful one to use as most systems are likely to be similar to the level crossing in that whilst parts of the system may be suited to Petri Nets not the entire system is. Essentially, it would appear that Petri nets are best used when each event has a visible consequence which may be modelled as a postcondition which then forms the precondition to the next event. However where a system is made up of a series of sequential events, where

perhaps the only visible consequences occur after several events have occurred then Petri nets become less suitable.

A second difficulty found with Petri nets was that their graphical nature, although useful in the reading of a completed specification, had a tendency to quickly become too complex. This may be overcome to a certain extent through the use of sub-nets but this in itself makes a specification more difficult to follow. Whilst the graphical Petri Net may be represented as a four-tuple this further reduces readability of the specification making the language less accessible.

4.2 SPECIFICATION OF THE TIMED ELEMENTS OF THE SYSTEM

4.2.1 TIMED CSP

As was shown above, timed CSP was not particularly suitable for the specification of the time constraints in the level crossing system. It was not possible to specify the constraints in their original form and the result is a system which does not behave exactly as required. In conclusion, timed CSP cannot be said to have assisted with specifying the required system. The use of the timeout operator allowed us to model a similar system to that which was required but there are some additions to the language which would be particularly useful. These are a timeout operator that can take into account a group of events and a retry operator that may be applied to a specific operator.

The use of timed CSP in the secure room case study was much more successful. The timeout operator allowed the system to be specified without any alteration to the time constraint and the result is a neat and concise specification. If a system contains time constraints which are all based on a single event then timed CSP will allow the system to be specified neatly and, if required, verified through the use of traces.

4.2.2 TBN

The applicability of TBN in the two case studies mirrored that of timed CSP. If a time constraint is based on a single event or transition then it is a straightforward matter to include it in the specification. If, however it is based on more than one event as in the level crossing then it cannot be accurately specified. The Petri Nets shown above have attempted to get around this by including more than one event in a single transition. This is not, however a particularly acceptable solution because there is no way to accurately detect errors if this route is taken and error detection is essential in a real-time system. One serious difficulty encountered with the use of TBN was that the initial Petri Net specification needs to be produced with TBN in mind. The initial Petri Net specification which was produced had to be scrapped and a new one produced which would allow for inclusion of the time constraints. This is a serious disadvantage in comparison with timed CSP where the only changes required were the breakdown of the original CSP processes. Thus whilst an ordinary CSP specification may quite easily be adapted to incorporate the timed elements of a system, the same is not true for Petri Nets and TBN.

5.3 COMPARISONS AND FINAL CONCLUSIONS

In terms of their ability to specify time constraints there is little to choose between the two languages. Both were able to specify the time constraint in the secure room without any difficulty, but both were unable to deal with the timing requirements of the level crossing system. This leads to the obvious conclusion that it could be that the nature of the time constraints contained within the level crossing case study place unrealistic expectations on the timed elements of a specification language. However, further research is required to establish whether or not this is the case and the possibility that the two languages simply contain the same shortcoming cannot be discounted.

As an 'untimed' specification language, CSP would appear to be both easier to use and a superior language in terms of system development and readability. CSP also confers the advantage that timing requirements may be incorporated into a specification without difficulty, although this advantage is only beneficial in those cases where timed CSP is able to handle the timing requirements of the system. Therefore in final conclusion it seems that if time constraints based on no more than one event are to be specified CSP would be the better choice of language to use. Neither language, however, is suitable for the specification of time constraints based on more than one event.

REFERENCES

[BAI91] Baillie, J. *A CCS case study: a safety critical system.* Software Engineering Journal 1991 *pp159-167*

[DAV94] Davies, J. and Schneider, S. *A Brief History of Timed CSP* Programming Research Group Monograph. Oxford University 1994

[GOR87] Gorski, J. *Formal Support for Development of Safety Related Systems* Safety and Reliability Symposium, Altrincham, UK (Elsevier Applied Sciences)

[HOA87] Hoare, C.A.R. *Communicating Sequential Processes* (Prentice Hall International 1987)

[KUT97] Kutar, M.S. *A Study of Communicating Sequential Processes in Relation to Real-Time Systems* MSc Computer Science Project Report University of Hertfordshire 1997

[NIS97] Nimal, Dr S. *Realtime Systems* (Prentice Hall Series in Computer Science 1997)

[PET81] Peterson, J.L. *Petri Net Theory and the Modelling of Systems* (Prentice Hall 1981)

[SCH96] Schneider, S. *Real-Time Systems - Specification, Verification and Analysis.* (Edited by Mathai Joseph) Prentice Hall International 1996

[SEL96] Selic, B. *Real-Time Object-Oriented Modelling* Tutorial 54. Conference on Object-Oriented Programming Systems, Languages and Applications. *(OOPSLA)* San Jose, California, October 1996