World Academy of Science, Engineering and Technology
International Journal of Computer and Systems Engineering
Vol:2, No:1, 2008

# Sensitivity Analysis of Real-Time Systems

Benjamin Gorry, Andrew Ireland, and Peter King

***Abstract*—**Verification of real-time software systems can be expensive in terms of time and resources. Testing is the main method of proving correctness but has been shown to be a long and time consuming process. Everyday engineers are usually unwilling to adopt formal approaches to correctness because of the overhead associated with developing their knowledge of such techniques. Performance modelling techniques allow systems to be evaluated with respect to timing constraints. This paper describes PARTES, a framework which guides the extraction of performance models from programs written in an annotated subset of C.

***Keywords*—**Performance Modelling, Real-time, Sensitivity Analysis.

Open Science Index, Computer and Systems Engineering Vol:2, No:1, 2008 publications.waset.org/8128.pdf

## I. Introduction

REAL-TIME systems are everywhere in modern day society [1], such systems are usually part of a self-contained product and are known as embedded real-time systems [2]. The range of application domains where a potential error could be disastrous makes the construction of a fault-free dependable real-time system of the highest priority [3]. Time as a continuous factor can prove to be a difficult element to account for. Continuous real-time software systems vary in the granularity of the models of time they use from coarse to finely-grained representations. Time must be regarded as an intrinsic item in modern day safety-critical system development as a potentially catastrophic fault may occur at any given epoch [2, 4].

The addition of real-time values leads to increased difficulty when we verify the correctness levels of the system which has been developed. Testing techniques have traditionally been used to prove correctness of real-time software systems but are regarded as being very time consuming and laborious when trying to achieve adequate test coverage [5, 6]. This paper describes in detail how the mechanisms that underpin PARTES [7] (Performance Analysis of Real-Time Embedded Systems), a framework where performance modelling technology is used to facilitate the verification of real-time software systems, can assist in the assurance of real-time embedded systems. For a brief 2-page overview of PARTES please see [7].

The remainder of this paper has the following structure; §II introduces the real-time dining philosophers example which will be used to illustrate our approach, §III presents the background behind the formal analysis approach used by PARTES, and §IV discusses how PARTES develops formal models for analysis. §V discusses how PARTES can be used via a series of example case studies and §VI discusses what PARTES provides us with. Our conclusions are listed in §VII.

## II. Real-Time Dining Philosophers

The real-time dining philosophers example has been adapted from Dijkstra's example as cited by Hoare [8]. It involves *n* philosophers, where each philosopher can be thought of as a generic system component. Each philosopher has in their possession one fork and must have the use of two forks to eat. Each philosopher thinks for a period of time then attempts to gain control of their own fork and their neighbours fork. If a philosopher has obtained both forks they then eat for a period of time and then relinquish control of their own fork and then their neighbours fork. If a philosopher cannot obtain both forks at the first attempt they repeatedly attempt to obtain both forks until they do so. The real-time areas in this example are:

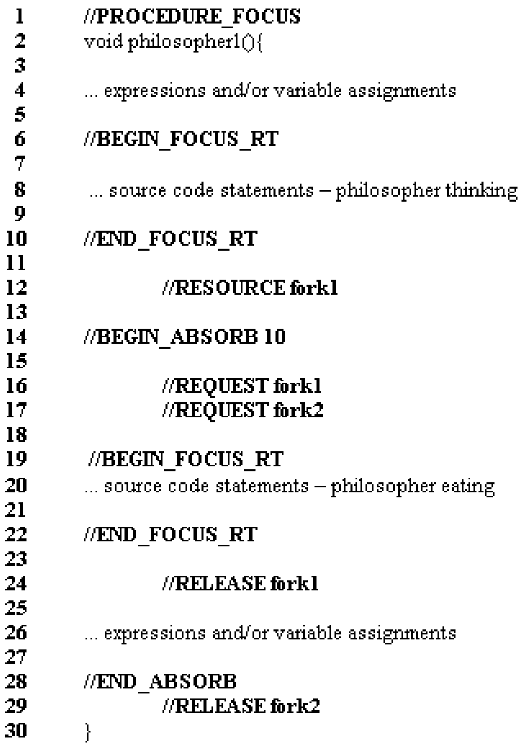
- Each philosopher takes a random time to think; with a maximum value of 5 seconds.
- Each philosopher takes a random time to eat; with a maximum value of 5 seconds.

These values are set as timing restrictions placed on the source code for the purpose of this case study; they are not restrictions placed on the models which are extracted for analysis. Each philosopher is to be checked for the timing constraint *“when a philosopher has completed thinking they must have, from this point in time, obtained access to both their fork and their neighbours fork and completed eating within 10 seconds”*. If this was not in place a philosopher may continue eating forever, this would cause the other philosopher to starve to death. Real-time examples such as the dining philosophers problem may involve several distributed components competing for a number of shared resources. The system timing constraints may be stringent, therefore there is a requirement to identify any areas of timing concern which may cause these timing constraints to be violated. This example has been modelled in the language ANSI-C and the method of communication which has been used is the User Datagram Protocol. The reduced source code for philosopher 1 is listed in Fig. 1. The correctness of a real-time system depends not only on the logical result of the computation performed, but also on the time at which the

B. Gorry is with BAE Systems Rapid Engineering, Military Air Solutions, Warton Aerodrome, Preston, Lancashire, England.

A. Ireland and P. King are with the School of Mathematical and Computer Sciences, Heriot-Watt University, Riccarton, Edinburgh, Scotland.

results are produced [4]. Since it is estimated that testing consumes at least half of the labour expended to produce a working program [5, 6] it is desirable that we reduce this consumption of resources. If we are to use testing:



```
//PROCEDURE_FOCUS
void philosopher1(){

... expressions and/or variable assignments

//BEGIN_FOCUS_RT

 ... source code statements – philosopher thinking

//END_FOCUS_RT

          //RESOURCE fork1

//BEGIN_ABSORB 10

          //REQUEST fork1
          //REQUEST fork2

 //BEGIN_FOCUS_RT
... source code statements – philosopher eating

//END_FOCUS_RT

          //RELEASE fork1

... expressions and/or variable assignments

//END_ABSORB
          //RELEASE fork2
}
```

Fig. 1 Source Code with Annotations for a Real-Time Dining Philosopher

> *"our goal should be to provide enough testing to ensure that the probability of failure due to hibernating bugs is low enough to accept"* [5].

If we could focus our areas of testing, our costs in terms of time and labour could be reduced and we could spend more time testing potentially problematic areas of our system.

## III. Background

### A. Performance Modelling

*Performance Modelling* techniques can provide us with enough information to allow us to determine if the design or system we are analysing will perform as we intend it to. This analysis often takes the form of probabilistic analysis and can involve the use of a process algebra such as CSP [8]. There are a number of tools available which use these algebras to carry out probabilistic analysis, such as PEPA [9]. A widely used performance analysis tool is SPNP (Stochastic Petri Net Package) [10, 11] which analyses models written in CSPL (C-based Stochastic Petri Net language) form, a language akin in syntax to C. Testing often proves to be expensive in terms of money, time, and manpower. To eradicate this problem we can use performance modelling techniques; by representing the intrinsic characteristics of the timing elements of the system we can evaluate how the system will perform under a range of possible conditions [12].

SPNP offers a wide variety of user-definable options on a range of model-types, the most widely used of which are *Generalised Stochastic Petri Nets* (GSPNs). For our purposes, we require analysis which can represent both timed and un-timed sections of the system, GSPNs deliver this. SPNP offers an *analytic-numeric solution* technique. Analytic-numeric solution allows us to specify a stochastic reward net as a Continuous Time Markov Chain (CTMC). The CTMC eliminates the need for analysis in discrete steps. With analytic-numeric solution, we can perform *sensitivity analysis* on a CSPL model.

The problem with performance modelling tools is that the models have to be constructed manually. If we could mechanise the construction of these models we could increase the accessibility of performance modelling techniques to the software community.

### B. Sensitivity Analysis

Performance modelling offers an approach called *sensitivity analysis*. Sensitivity analysis allows us to analyse how variations of system parameters affect the performance of a model and is defined as:

> *"the study of how variation in the output of a model (numerical or otherwise) can be apportioned, qualitatively or quantitatively, to different sources of variation, and of how the given approach depends upon the information fed into it" [13].*

Sensitivity analysis allows us to observe how the individual sections of a system will behave by generating partial derivatives of these measures. These partial derivatives provide us with an indication of the likelihood of a section or sections of a system being a contributing factor to causing the timing constraint to be violated. From this, we can then investigate these areas of concern. A common sensitivity measure is the MTTA (Mean Time To Absorption). The MTTA provides a measure of when an absorbing state will be reached, if an absorbing state is reached the system will progress no further (it will deadlock). In PARTES, each partial derivative which is generated is a partial derivative of

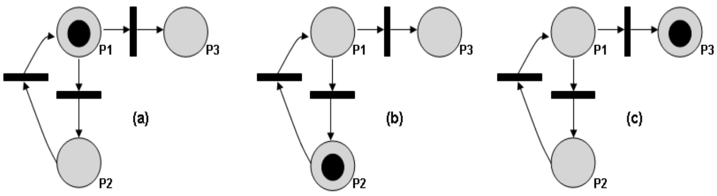


Fig. 2 An Example of Absorption (Deadlock)

the MTTA with respect to an area of timing importance in the model (those areas which represent sections of program source code). If the value of a partial derivative:

- *Increases* - a section of source code is likely to cause a system timing constraint to be violated.
- *Decreases* - a section of source code is unlikely to cause a system timing constraint to be violated.
- *Remains constant* - a section of source code is not directly related to the system timing constraint being analysed.

The concept of an absorbing state is illustrated in Fig. 2. As we see in Fig. 2 (a), the token is initially in place P1. From P1, the token could travel either to place P2 (as shown in Fig. 2 (b)) or place P3 (as shown in Fig. 2 (c)). If the token moves to place P2 it will then return to place P1. However, if the token moves to place P3 it has nowhere else to go and is "absorbed" into place P3. From this, we see that the model will not progress any further, for this reason we can say that when a state of absorption is reached the model has reached a point of "deadlock".

## IV. PARTES

PARTES links performance modelling techniques to the source code level. If any real-time areas of concern are identified this information can be used to inform traditional testing techniques by allowing testing efforts to be focused on specific areas of concern. By using a set of annotations which can be placed directly in the ANSI-C source code a set of structured formal models which represent faithful abstractions of the real-time system can be extracted. After formal analysis has been performed, the engineer can relate the results of any errors produced back to the sections of program source code which the model represents. We now discuss the development and analysis of PARTES performance models.

### A. *PARTES Performance Models*

PARTES extracts the structure of a performance model from the program source code in the form of a performance test harness. To produce CSPL models, this information is then combined with the results of analysing the pertinent real-time sections of the program source code to gather *actual* mean system execution times. Fig. 1 shows the placement of annotations in the ANSI-C code for a philosopher, each annotation begins with the // symbol. To begin with, the user must identify which procedures within the source code are of importance. To identify the pertinent procedures the user places the annotation listed on line 1 of Fig. 1 prior to the beginning of each procedure which they wish to include as part of the performance test harness.

The annotation on line 12 of Fig. 1 indicates that a shared binary resource 'fork1' has been created. To request access to 'fork1' or release control of 'fork1' the annotations are listed on lines 16 and 24 of Fig. 1 respectively. By using annotations we can indicate exactly where an area of real-time concern exists. Annotations which delimit an area of timing concern within a procedure are, for the start and end point respectively listed on lines 6 and 10 of Fig. 1 to indicate when a philosopher is thinking and on lines 19 and 22 of Fig. 1 to indicate when a philosopher is eating. To add a real time correctness constraint to a procedure the annotations which delimit the start and end point are listed on lines 14 and 28 of Fig. 1. Line 14 indicates that the value of the timing constraint for a philosopher is 10 seconds. Using these annotations, a performance test harness which represents the structure of the program source code in Petri net form is automatically extracted. This test harness represents a number of Petri net components; these are illustrated alongside the corresponding source code sections/PARTES annotations from Fig. 1 in Fig. 3. From the annotations in Fig. 1 the Petri net components in Fig. 3 are plugged together to construct CSPL models. To begin with, an initial place indicating a starting point in the model is created. This place is linked to another place via a timed transition; this is illustrated as Fig. 3 (a). Line 12 of Fig. 1 states that the shared resource "fork1" has been created. This is represented as the structure in Fig. 3 (d). A sequence of variable assignments or expressions is represented as a place linked to another place via an immediate transition. This is represented as the structure in Fig. 3 (b). Lines 16 and 24 of Fig. 1 represent access and release requests to shared resource "fork1". These are represented as structures (e) and (f) in Fig. 3, respectively. The annotations on lines 6 and 10, along with the annotations on lines 19 and 22 of Fig. 1 delimit areas of timing concern. These are represented as a place linked to another place via a timed transition. This is represented as structure (g) in Fig. 3.

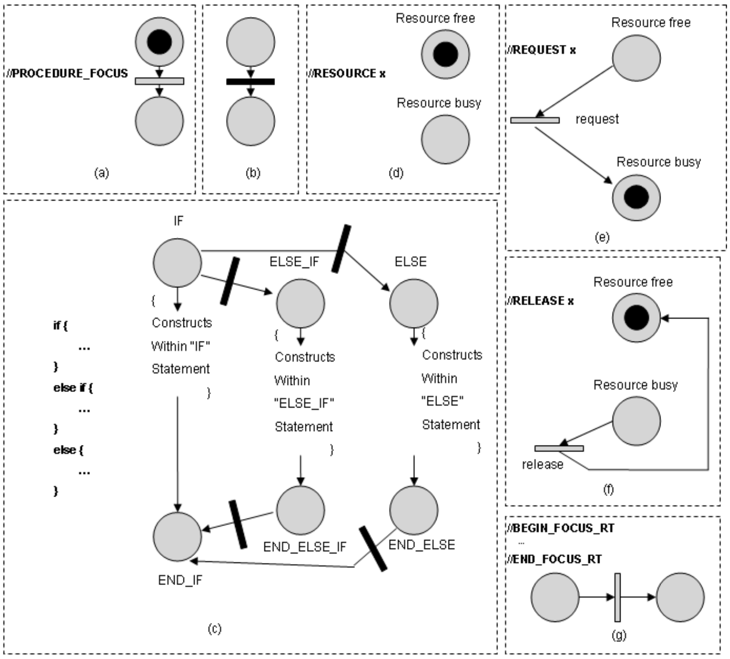


Fig. 3 PARTES Petri net Components

The development of the structure of performance models in PARTES is a generic process which uses the constructs outlined in Figs. 3 and 5. By using annotations we can indicate, at any point in the source code within a procedure, exactly where an area of real-time concern exists. When annotations which delimit an area of timing importance are encountered any ANSI-C constructs are ignored since it is the area of timing importance which is important, not the ANSI-C source code structures contained within it. The two areas identified in Fig, 1 represent a philosophers thinking and eating times.

For structure (g) we require times to model as transition firing rates. Gathering the *actual* system timing values can be problematic. During the first stage of PARTES the user places paired annotations in the ANSI-C source code. These annotations are then automatically numbered to indicate the position of pairs. For an area of timing importance, annotations which delimit the boundaries take the form //BEGIN_FOCUS_RT t and //END_FOCUS_RT t where "t" is the number assigned to a particular pair of annotations. In PARTES these annotations act as timing hooks which are replaced by additional sections of source code which are used to gather timing information. The source code which has been augmented with the additional code is executed on the *actual* system hardware to allow us to gather timing values which represent the *actual* system execution times.

To capture timing values the *clock()* function from the C library <time.h> is used. This provides the elapsed processor time used by the program. This value is then divided by CLOCKS_PER_SEC, the number of processor clock ticks per second, to provide us with the value of the total program execution time in seconds. This calculation is performed at the point of each annotation and from this the value of the first calculation is subtracted from the second calculation to provide us with a timing value in seconds. Since the system is run on the actual hardware which it is intended to run on, the times which we gather are as accurate as possible a representation of what the system timing values will be when the system is executed without any adverse interruptions to performance. Two mean execution times were gathered for each philosopher, the mean time a philosopher spends thinking and the mean time a philosopher spends eating. The mean time philosopher 1 spent thinking was 3 seconds while the mean time philosopher 1 spent eating was 3 seconds. These values are placed as parameters at the relevant points in the model which are represented as structure (g) in Fig. 3. The mean time philosopher 2 spent thinking was 4 seconds while the mean time philosopher 2 spent eating was 2 seconds. Up to this point only two philosophers are included, both are generic in structure. Later in this paper we will discuss the real-time dining philosophers example for several philosophers. In addition to this, an example illustrating the use of PARTES on a system containing bespoke components is discussed.

Since we are measuring the execution time of the program relevant to the hardware it is expected to run on, there may be occasions when the program execution times vary. To account for the possible spurious nature of some timing results we have used confidence intervals. Confidence intervals provide us with a degree of confidence that the mean value, in this case the mean timing value for a section of source code, lies within a pair of numbers known as confidence limits. For the purposes of timing analysis confidence intervals set at 95% have been used. A confidence interval of 95% states that on 95% of occasions, the mean value will fall within the interval bounded by these limits. In PARTES the mean value is taken as the mean value between these confidence intervals. After a sample size of 30+ execution runs the mean value is calculated and if it lies within these confidence limits we take this mean value to be an accurate representation of the timing value.

For the real-time dining philosophers example, a diagrammatic representation of a model of a generic philosopher is presented in Fig. 4. The only Petri net construct in Fig. 4 which is not illustrated in Fig. 3 are the timed transitions which link the end point of a model to the beginning and the conclusion of a correctness timing constraint. The timed transition which links the end point of a model to the beginning is used to close the loop of the model to ensure that it does not halt when it reaches the end place. Fig. 5 illustrates the Petri net notation which corresponds to a system timing constraint. The black bars in Fig. 5 represent immediate transitions, these indicate the points in the model where the timing constraint is linked into the model, these link points may also be timed transitions. The upper timed transition represents a link point which corresponds to the annotation on line 14 of Fig. 1. The end point of the area

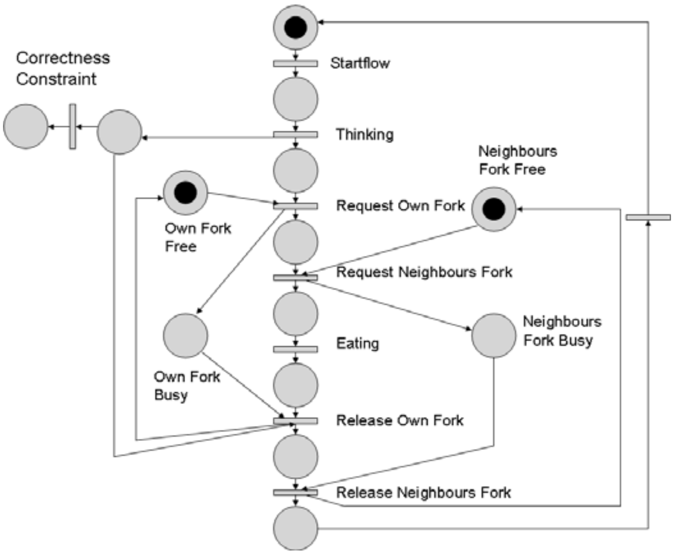


Fig. 4 Petri Net Of Real-Time Dining Philosophers Example

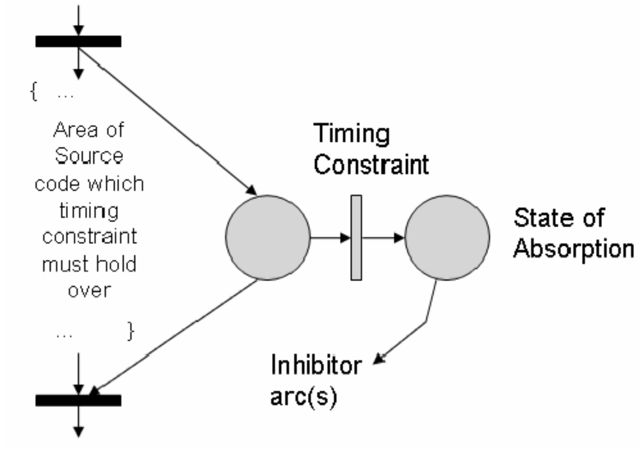


Fig. 5 Structure of a Correctness Timing Constraint

which the timing constraint must hold over is represented by the annotation on line 28 of Fig. 1. The upper transition link point leads to a place which then links to the lower transition link point. The value of the timing constraint is determined by the number 10 as part of the annotation on line 14 of Fig. 1.

If the total time taken to execute the sections of source code which the timing constraint must hold over is less than the value of the timing constraint, the timed transition which represents the timing constraint will not fire and therefore the state (place) of absorption will not be reached. This would indicate that it is likely that the sections of source code which the timing constraint must hold over will execute in the desired time. If the place of absorption is reached, inhibitor arcs which are linked from the absorbing place to each of the timed transitions in the model will halt any further progress through the model. For the real-time dining philosophers example sensitivity analysis showed that if the eating times for both philosophers summed to 10 seconds or greater it is likely that the timing constraint will be violated. The average eating times were 3 seconds for philosopher 1 and 2 seconds for philosopher 2. We continually increased the eating time for philosopher 1, while keeping the eating time for philosopher 2 constant at 2 seconds, and found that a state of absorption was approached when philosopher 1 took 8 seconds to eat. This can be seen on the graphs in Fig. 6. The graphs in Fig. 6 illustrate how the sensitivity of the eating time of philosopher 1 (Fig. 6 (a)) and the eating time of philosopher 2 (Fig. 6 (b)) contribute to causing a state of absorption to be reached as the eating time of philosopher 1 varies between 1 and 20 seconds. The graph in Fig. 6 (a) shows that the differences between the partial derivatives increases linearly up until the point in time at 8 seconds when they begin to decrease in difference. Conversely, the graph in Fig. 6 (b) shows that the differences between the partial derivatives decrease rapidly up until the point in time at 8 seconds where it appears that there is very little difference when comparing the values of the partial derivatives from this point onwards. These graphs indicate that it is likely that a state of absorption has been approached when the philosopher eating times sum to 10 seconds and that the eating time of philosopher 1, shown in Fig. 6 (a), is more likely to be the cause of this absorption than the eating time of philosopher 2, shown in Fig. 6 (b).

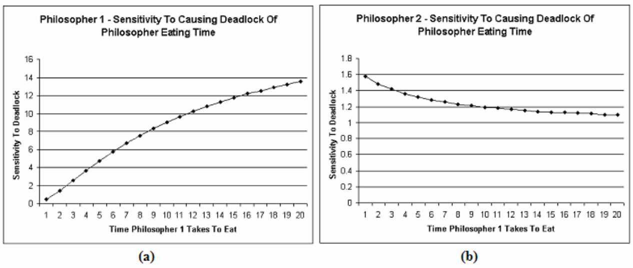


Fig. 6 Philosopher 1 and 2 Eating – Sensitivity to Causing Deadlock

Since a philosopher is allowed up to 5 seconds to eat and the average eating times were 3 seconds and 2 seconds, respectively, for philosopher 1 and philosopher 2, they appear to be acceptable and will not cause the system to reach a state of absorption. Performance analysis has shown that it is likely that this system will not violate its timing constraints while providing an indication of when the system may violate its timing constraints. This analysis has provided the engineer with a degree of assurance that their system is correct for the current timing values while providing an indication of which timing values they should remain within the bounds of to prevent a possible violation of the system timing constraint. A full description of this case study can be found in [14].

TABLE I
EIGHT DINING PHILOSOPHERS WITH RESULTS FROM ANALYSIS

| Philosopher | Thinking Time | Eating Time |
|---|---|---|
| 1 | 3 | 3 |
| 2 | 4 | 2 |
| 3 | 5 | 5 |
| 4 | 3 | 1 |
| 5 | 2 | 4 |
| 6 | 1 | 8 |
| 7 | 2 | 2 |
| 8 | 4 | 6 |

(a)

| Philosopher Number | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| 1 | | | | | | X | | |
| 2 | | | | | | X | | |
| 3 | | | | | | X | | X |
| 4 | | | | | | | | |
| 5 | | | | | | X | | X |
| 6 | X | X | X | | X | X | X | X |
| 7 | | | | | | X | | |
| 8 | | | X | | X | X | | |

(b)

Following the analysis of two philosophers, combinations from a selection of 8 philosophers, that have different thinking and eating times, were analysed. A table listing the details of these eight philosophers is presented as Table I (a). Each philosopher was paired with each other philosopher and sensitivity analysis was performed. The results from this analysis are listed in Table I (b). The presence of an `X' shows that a particular pairing of philosophers will cause a correctness timing constraint to be violated. These results show that philosopher 4 is the only philosopher that can be safely paired with every other philosopher. Further analysis involved combinations of three philosophers. Results from this showed that safe combinations of philosophers are 1, 2 and 4 or 1, 2, and 5.

## V. CASE STUDIES

To illustrate how PARTES can be used we have developed two other case studies which show what value PARTES can deliver when identifying possible areas of real-time concern. These are now discussed. Further details can be found in [14]. These case studies differ in:

- The placement of annotations.
- The number of areas of timing importance.
- The range of timing values used.
- Whether the system components are bespoke or generic.
- The number of shared resources in use.

### A. Shared Train Track Example

The shared train track example involves two trains which run on parallel tracks and contend for the use of a single stretch of track. This single stretch of track allows only one train at a time to cross a bridge. Each train approaches the bridge then requests access to the bridge. If the bridge is not in use by another train the train is granted access to the bridge. If the bridge is in use by another train the train waits until the bridge becomes available. When the train has access to the bridge the train crosses the bridge. The bridge is then free to be used. The real-time values in this example are:

- Each train takes a random time to approach the bridge; with a maximum value of 10 seconds.
- Each train will take a random time to cross the bridge; with a maximum value of 10 seconds.

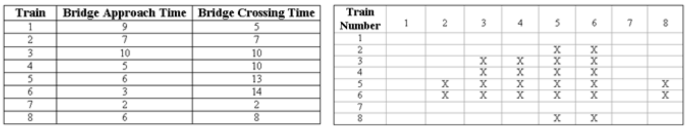
These values are set as timing restrictions placed on the source code for the purpose of this case study; they are not restrictions placed on the models which are extracted for analysis. The embedded software which controls each train is to be checked for the timing constraint:

> *"when a train has completed its approach to the bridge it must have, from this point in time, obtained access to the bridge and completed its crossing of the bridge within 20 seconds."*

For train 1, the mean time it took the train to approach the bridge was 9 seconds while the mean time it took to cross the bridge was 5 seconds. For train 2, the mean time it took the train to approach the bridge was 7 seconds while the mean time it took to cross the bridge was 7 seconds.

Results showed that the system begins to reach a state of absorption at a time when the sum of the bridge crossing times for both trains exceeds 20 seconds. Since a train is allowed up to 10 seconds to cross the bridge and the average bridge crossing times were 5 seconds and 7 seconds, respectively, for train 1 and train 2, they appear to be acceptable. Therefore these times will not cause the system to reach a state of absorption. Performance analysis of the shared train track example has shown that it is likely that this system will not violate its timing constraints. After establishing that there appeared to be no areas of concern in the shared train track example, further sensitivity test were performed on the example. Trains may vary in the time they take to approach the bridge and the time they take to cross the bridge. The times for eight different trains are listed in Table II (a).

TABLE II
TIMING VALUES AND PAIRING RESULTS FOR EIGHT TRAINS



| Train | Bridge Approach Time | Bridge Crossing Time |
|---|---|---|
| 1 | 9 | 5 |
| 2 | 7 | 7 |
| 3 | 10 | 10 |
| 4 | 5 | 10 |
| 5 | 6 | 13 |
| 6 | 3 | 14 |
| 7 | 2 | 2 |
| 8 | 6 | 8 |

(a)

| Train Number | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| 1 | | | | | | | | |
| 2 | | | | | X | X | | |
| 3 | | | X | X | X | X | | |
| 4 | | | X | X | X | X | | |
| 5 | | X | X | X | X | X | | X |
| 6 | | X | X | X | X | X | | X |
| 7 | | | | | | | | |
| 8 | | | | | X | X | | |

(b)

Each of these trains was paired with each other train and sensitivity analysis was performed on each system. This allows us to determine if previous conclusions which were based on the first two trains listed in Table II (a) are correct or not. Table II (b) lists eight pairings for each of the eight trains. An `X' symbol indicates that a pairing is not possible since the paring of a particular pair of trains will cause the timing constraint to be violated. These sensitivity tests confirmed that if the sum of the train bridge crossing times exceeds 20 seconds the system begins to reach a state of deadlock. In addition to this, train bridge crossing times which exceed 10 seconds have been included and, if paired with a train that brings the combined sum of the bridge crossing times to less than 20 seconds, the system timing constraint will not be violated. In this situation, sensitivity analysis shows that even though the combined sum of the bridge crossing times of two trains may not exceed 20 seconds, the train which exceeds the maximum crossing time of 10 seconds will be identified as being likely to cause a violation of the system timing constraint. Even though a system timing constraint has not been violated a possible area of concern has been identified. As well as a possible area of timing concern, this identifies that additional system time is available since one train has exceeded the maximum crossing time without violating any system timing constraint. This time could then be allocated elsewhere in the system.

### B. Embedded Control System

The embedded control system example is taken from a paper by Muppala, Ciardo, and Trivedi [15]. It consists of an input processor which is connected to three sensors, an output processor which is connected to two actuators (one active actuator and one spare actuator), a main processor, and a communications bus. The input processor takes readings from its three sensors, performs some computation on these readings and passes the result to the main processor. The main processor takes this result and converts it into commands which are passed to the output processor which in turn passes these commands to an actuator. This system could be used in a variety of environments including the chemical and avionics industries [15]. The real-time values in this example are:

- The input processor takes a random time to check sensor values; with a maximum value of 5 seconds.
- The main processor takes a random time to convert input sensor results into output sensor commands; with a maximum value of 5 seconds.

- The output processor takes a random time to communicate commands to its actuator; with a maximum value of 5 seconds.

These values are set as timing restrictions placed on the source code for the purpose of this case study; they are not restrictions placed on the models which are extracted for analysis. The embedded control system is to be checked for the timing constraint:

*"during a complete system cycle the system must have performed each of the tasks listed above within 20 seconds."*

The input and output processors each have timing constraints of 10 seconds while the main processor has a timing constraint of 20 seconds. The mean time the input processor spent gathering sensor values was 4 seconds, the mean time the main processor spent converting sensor values into actuator commands was 3 seconds, and the mean time the output processor spent issuing the actuator commands was 4 seconds.

Results showed that when the MTTA was observed for the sample sensitivity analysis experiments it showed that the model appeared to reach a state of absorption when the input processor took 10 seconds or more to check the input sensors, even though the time that the main processor spent converting sensor values into actuator commands was 3 seconds and the time the output processor spent issuing the actuator commands was 4 seconds (less than 20 seconds in total). At this point it was observed that the main processor looked sensitive to being the cause of absorption. This was a consequence of the relationship between the input processor and the main processor. From the results we can say that for a system to avoid causing a state of deadlock the timing values for either the input or output processors should be no greater than 10 seconds. It appears that the system begins to reach a state of deadlock at a time when either of the three system timing constraints have been violated. Even though the main processor has a constraint value of 20 seconds, sensitivity analysis has shown how this can be effected by the links the main processor has with the input and output processors. Since the actual times were 4 seconds, 3 seconds, and 4 seconds for the input processor, main processor, and output processor, respectively, they appear to be acceptable and will not cause the system to reach a state of absorption. Performance analysis has shown that this system will not violate its timing constraints.

The CSPL models generated by PARTES use a notation so that the areas of timing concern can be related directly back to the original source code which has been annotated. Since we are able to identify potential problem areas, we can then focus the attention of further analysis techniques on these areas and potentially guide the application of fault tolerant approaches.

## VI. Discussion

We have illustrated a framework which supports the mechanisation of the extraction of a test harness which allows existing technology to be utilised while extracting performance models which can be used to evaluate real-time correctness properties. Once the source code has been annotated any minimal changes to future revisions will make the generation of a test harness relatively simple.

By adopting an annotation-driven approach to abstraction, based at the level of the program source code, we minimize the amount of knowledge of formal techniques that an engineer requires while allowing them to benefit from using approaches such as model-checking and performance modelling. If engineers at the levels of testing and verification employed a technique similar to that discussed in this paper, it may allow them to use verification technology to focus on potentially problematic areas of the software system under analysis.

## VII. Conclusion

We have presented a framework which builds on an existing technique to help aid in the verification of real-time correctness properties of software systems. This approach has been discussed via example case studies which highlight its flexibility and strength. A discussion of the benefits of such an approach has been provided. We aim to further investigate the range of performance modelling technologies and features which are available so that we can maximize their use and provide a varied range of options for use by the engineer. A long-term aim will be to investigate other modelling technologies such as [9] and [16] along with their related specification languages to investigate whether a set of tools could be developed.

## References

[1] Albert M. K. Cheng, *Real-Time Systems - Scheduling, Analysis, and Verification*, John Wiley & Sons, Inc, 2002.

[2] H. Kopetz, *Real-Time Systems, Design principles for Distributed Embedded Applications*, Kluwer Academic Publishers, 1997.

[3] J-C Laprie, *Dependability: Basic Concepts and Terminology*, Springer-Verlag Wien New York, 1991.

[4] A. Burns and A. Wellings, *Real-Time Systems and Programming Languages*, Pearson Education Limited, 2001.

[5] B. Beizer, *Software Testing Techniques*, International Thomson Computer Press, 1990.

[6] E. Kit, *Software Testing In The Real World*, ACM Press, 1995.

[7] B. Gorry, A. Ireland and P. King, *PARTES : Performance Analysis of Real-Time Embedded Systems*, Proceedings of the 4th International Conference on the Quantitative Evaluation of SysTems (QEST), pg 271-272, 2007.

[8] C. A. R. Hoare, *Communicating Sequential Processes*, Prentice Hall International (UK) Ltd, 1985.

[9] S. Gillmore and J. Hillston, *The PEPA Workbench: A Tool to Support a Process Algebra-based Approach to Performance Modelling*, Proceedings 7th Int. Conference on Modelling Techniques and Tools for Computer Performance Evaluation, 353-368, 1994.

[10] G. Ciardo, J. Muppala and K. Trivedi, *SPNP: Stochastic Petri Net Package*, Department of Computer Science, Duke University, Durham, 1989.

[11] K. Trivedi, *SPNP User's Manual Version 6.0*, Center for Advanced Computing and Communication (CACC) Department of electrical and Computer Engineering, Duke University, 1999.

[12] P. J. B. King., *Computer and Communication Systems Performance Modelling*, Prentice Hall International (UK) Ltd, 1990.

[13] A. Saltelli, K. Chan and E. M. Scott, *Sensitivity Analysis*, John Wiley & Sons, Ltd., 2004.

[14] B. Gorry, A. Ireland and P. King, *Performance Analysis Of Real-Time Embedded Systems*, Heriot-Watt University, School Of Mathematical And Computer Sciences, Departmental Technical Report HW-MACS-TR-0040, 2006.

[15] J. K. Muppala, G. Ciardo and K. S. Trivedi, *Stochastic Reward Nets for Reliability Prediction, Communications In Reliability, Maintainability, and Serviceability*, 1, 9-20, 1994.

[16] M. Kwiatkowska, *Probabilistic Symbolic Model Checking with PRISM: A Hybrid Approach*, 8th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS`02), 2002.