Models' Validation for Complex Real-Time Systems

Irobi Ijeoma Sandra

Masters' Degree thesis
June 2004

Department of Computer Science and Engineering
Mälardalen University
Västerås Sweden

Abstract

Models' validation poses an interesting area of discussion in computer science and engineering as in other disciplines, which cannot easily be overlooked.
No model of a complex system can be said to be absolutely correct, or a perfect description of the system, as such models are, and should be an abstraction of the actual system.
This situation becomes even more glaring for real-time computer systems where stringent conditions are imposed on the system, for instance, timing and hence predictability constraints.

In this thesis we shall explore different perspective on models' validation and techniques in computer science and their attendant problems especially for complex real-time systems. Major open and philosophical issues on the correctness criteria for models validations that seek to answer such questions as: *Can models be validated? Can 'valid' models yield information? Do correct models contain truth?* and more shall be investigated as well. We will discuss the implications of our answers with respect to computer science and real-time systems in particular arguing vehemently that models can yield information.

We shall briefly investigate validation of models in computational physics, and seek to draw some parallels between its techniques and methodologies with those in computer science. Finally, we will present results on our validation experiments for a robotic electric control motor running on VxWorks operating system and from our work deduce and propose guidelines for engineers and empirical scientists working on models. These guidelines suggest simple formats of determining the number of criteria and queries that could be used in validation, given different levels of abstraction and model confidence.

*To the **Third Person** in the Trinity….The custodian of my dreams;*
And to my parents, for setting my feet firmly on the right foundation.

Acknowledgements

I am profoundly grateful to my supervisor Johan Andersson for his valuable help, advice and insights on this thesis, together with my examiner Anders Wall, whose PhD thesis gave me good inspiration to venture into this very interesting area of investigation.
I remain thankful to Christer Norström for coming up with this thesis and giving me the opportunity to work with his research team in this area and to Hans Hansson for introducing me to this program in the department in the very first place.

I would like to thank Gordana Dodig-Crnkovic - for her great understanding and useful answers to some of my questions in Computational Physics and Gerhard Fohler for providing me with initial challenges and training in research work. Ivan Christoff (Uppsala University), Torsten Soderstrom (Uppsala University), Cedric Linder (Uppsala University), Jafar Mahmoudi, Corporate Research department at Outokumpu Västerås for their immense help and comments that have improved the quality of this work.

I also wish to thank all members of my family especially Edward, Nkechi and Beulah for their support and goodwill during this period.
Finally many thanks to the ABB Robotics in Västerås for kindly lending us a system running VxWorks for the experiments and all staff of computer science and engineering department in MdH, for being such a wonderful bunch of encouragement in everyway.

Table of contents

1 Introduction

Simulation models have gained increasing grounds in being used in problem solving and as decision aids in several fields of study. It is the favorite tool of the engineer who intends to test and investigate new ideas while reducing industrial risks to the barest minimum; it is often a surrogate for experimentation with the actual system (whether it exists or still proposed), which is not cost-effective, disruptive, or just impossible. Simulation is used to evaluate the design of process components and separate unit operations. Model specifications are re-used in the design evaluation of large integrated processes and control systems.
In addition, the models once made are used to support operator training. Computerized dynamic simulation models are useful for verification of both conceptual and detailed process designs.
They make in-house pre-testing of automation systems, user interfaces, and operational procedures possible, as well. They are used for generic teaching and learning of basic principles, detailed pre-training of new personnel, and re-training of experienced operators.
Simulation models are extensively used in a situation when the real system cannot be used for experiments. This is the case for example when:

- The real system does not yet exist.
- The experiments would involve high economic risks.
- The experiments would be dangerous.
- The experiments cannot be controlled or carried out.
- The experimentation with the real system is expensive

1.1 Defining validation: Issues of terminology

There is a wide range of definitions and meanings given to verification and validation in different technical disciplines. For example, the meaning given to those terms by the Institute of Electrical and Electronics Engineers (IEEE) and the Software quality assurance community differ from that used in the Department of Defence Modelling and Simulation community (US), which has been the leader in the development of fundamental concepts and terminology for verification and validation.

The validation of a model is referred to as a process of ensuring that a given model is adequately accurate for a given purpose [1]. It is defined as the substantiation that a computerized model within its domain of applicability possesses a satisfactory range of accuracy consistent with the intended application of the model [2]. Generally, it is usually time consuming and expensive to establish absolute validity of a model over the complete domain in which it is applicable. Therefore tests and evaluations are conducted till a point of sufficient confidence is reached when the model could be considered valid for its intended application.

This is usually done by comparing between observations of the system's behavior and predictions made by analyzing the model when both the model and the system are driven under identical input conditions. Any model can only be validated with respect to the specific purpose for which it was made; therefore there is no such thing as a 100% accurate model. A model could be said to be valid for a particular set of conditions only when its accuracy parameters lie within a specified range. Hence, the aim of validation is to show that a given model is adequately accurate for its purpose, the conditions of which would normally be agreed upon right from the inception of the model development.

In order to expedite future usage of a model, it should be relatively easy to keep the model and the system consistent as the system evolves. Also the effort needed to adjust the model to reflect the impact of, for instance, a maintenance operation, should not be the same as building the initial model. The change required to update the model should be intuitive and similar to the change in the system.
Model validation provides a systematic framework to include model error and uncertainty in the decision-making process. Quite importantly, the inferences made in the validation domain need to be extrapolated to the untested region (area) where the actual application takes place.

1.2 Real-time systems

Our main thrust in considering validity for general models would be to consider validity criteria in particular for models of real-time software systems.

A real-time system could be referred to as *any system in which the time at which output is produced is significant. This is usually because the input corresponds to some movement in the physical world, and the output has to relate to the same movement. The lag from input time to output time must be sufficiently small for acceptable timeliness [3].*

Real-time computing systems have different requirements from general computing systems. For correctness purposes, the general computing systems only have to be functionally correct, which implies a situation where if given a specific input, they should yield the desired output. In this case, the timeliness of the output relative to the input is important.
In a multitasking environment, where more than one task is scheduled in the central processing unit (CPU), we could employ a scheduling algorithm in order to create a sequence in which the different tasks are run so as to achieve such timeliness. In that case one could consider factors like minimal average response times and maximum average system utilization to select an algorithm for use.

Real time systems are characterized by their temporal requirement. Hence apart from being functionally correct, they must also exhibit temporal correctness (correct function provided at correct time). The timing requirements are expressed as a bound on the time taken to perform some computation, which is normally referred to as the *deadline.* Depending on how much interest we have on the timing requirements of such systems being met, we could have *hard* real-time systems – where we are very concerned if the system does not respond to time correctly – for instance a car air-bag in the event of an accident. There are also those categorized as the *soft* real-time systems – where, with occasional misses of the timing requirements, there would be no need to worry.

There exist some ways by which one can show the temporal correctness of a real-time system. This includes building a model of the system mathematically and using it to analyze the system's behavior, which enables us to make good assertions as long as the model is good. However, this method could be quite difficult due to the some characteristic aspects of the real-time systems. For example, non-deterministic communication times, sharing of active system resources using different scheduling algorithms, tasks arrival patterns not adequately characterized prior to execution and may be probabilistic, making some details of simplification on the model to keep it tractable in such a way that one compromises the confidence level on the model.
Another second way could be by running extensive tests on a constructed prototype of the system in question. This could be very expensive to carry out in practice.

Irrespective of the system's critically, (whether soft or hard), adequate provision needs to be made to analyze the system with respect to its temporal behaviour and resource utilization.

1.3 Problem domain

Large complex real time systems have gradually come to stay even with the advent in technology. Most times, functionalities of such systems could be altered and new features added thereby changing the temporal correctness of initial models of such systems (if they exist). Due to this alteration, the model of the system (without the newly added system features) becomes quite different from the new system and fast becomes useless.
At first, this may not constitute a big issue but as time goes on and the system evolves, analysing it becomes increasingly an uphill task to the extent that the system may need to be re-engineered to permit analyzability.
One of the major validation challenges for software models is that computer software has a non-continuous nature, which implies that there could be drastic changes encountered as a result of little variations in the system unlike what may be obtainable in models in other fields.
It is very difficult if not impossible to explicitly and implicitly attribute validity to a software model for several reasons.
Different techniques have been proposed and are in use for the validation of real-time system models (as would be shown in subsequent sections). For each of these techniques, we shall investigate their criteria for correctness and on what factors they are based. Even though computer software are not continuous (which affects is models), as is the case with other disciplines, it is our believe that useful deductions on other common grounds between these fields could be mapped and modified for use in computer science and real-time systems in particular.

1.4 Why simulation?

Several models' verification and validation analyses methods abound. Each of these techniques has inherently good and poor points. Some of such methods of analysis include:

- *Model checking methods* (for instance Uppaal and TIMES tools): This method and tools are discussed later in details. The main ideas are to check for reachability and schedulability in the modelled states representing the real system. The major low point of this method is the 'state explosion problem', which entails the inability to effectively capture all possible states for large systems. This, presently is still a hot area of research.
- *Traditional Schedulability analyses methods* (for instance the Rate monotonic method): The major disadvantage of analysis with this method especially for complex real-time systems lies with its limited modelling language, which does not capture the entirety of intended domains for representation.
- *Simulation based approach* – This approach manages the state space explosion problem much better than the model checking methods but on the other hand can result in a low level of confidence on the results from analysing the model. The reason is mainly because the simulator has the ability to only make predictions based on the simulated instances and not on unsimulated ones. Hence, while one can confidently simulate, for instance, 1000 cases and make predictions based on them, the predictions for 1001th case is unknown since the simulation does not cover it. The simulation-based approach was used in this thesis for our experiments.

We use simulations to determine the temporal properties of complex real-time systems, which do not easily yield to deterministic analysis methods. One can view simulations as an approach of compromise [4] where we can utilize higher fidelity models than would normally be possible with mathematical analysis and can complete the validation with less cost than would have been if testing and prototyping methods were used.
The down part about simulations is that the results that we get from them could be *less certain* than those we can get by using mathematical techniques. Also, more simplifications by way of levels of abstraction could be made in such a manner that the level of confidence on the model may not be as high as that of a real prototype.
Simulation could allow execution times to be expressed as distributions rather than worst case values. Analysis of the simulation results is done by defining properties of interests in the target system. For instance, such interesting properties could include the probability of missing a deadline requirement on a task. Simulation also gives one the liberty to define non-temporal related properties, for example non- empty message queues.

Section 2 of this thesis deals with related work, and shall review how validation is done for real-time systems in computer science. Section 3 handles the philosophical perspectives (open questions) on models validation while Section 4 will takes on validation of models of real-time systems.
We discuss modelling tools in Section 5 and in Section 6 we explore validation methodologies in computational physics with a brief comparison to those in computer science and engineering. Section 7 reports our experiments on validation of the model of a robotic system running on VxWorks, results and inferences. The proposed guidelines on validation for use by empirical scientists and engineers working on models' validation are presented in Section 8. We conclude in Section 9 and give hints of future works in Section 10.

2 Related Work

In this section, we shall explore works that have been done previously with respect to validation. We shall discuss general approaches of validation and their procedures. Model robustness and common sources of errors in models and how to avoid them would also be highlighted.

2.1 The validation Process

There are many scientific grounds on which simulation models are constructed and even more techniques on which their validation rests.

In his paper [2], Sargent explored three major *general* approaches used to ascertain whether a model is valid or invalid. These three approaches are:

- Decision by the development team based on the results of the various tests and evaluations conducted as part of the model process.
- The independent verification & validation approach, which uses a third independent party to decide the validity of the model, often used for model accreditation.
- Use of a scoring model, where scores are given to different parts of the validation process and a cumulative of such scores, if it supersedes a given passing overall and category score, certifies the model valid. This approach seems to be rather too objective than it really is and could pose the problem of over-confidence in the model.

A simplified version of the modelling process as shown in Figure1 is described. The problem entity depicts the real or proposed system or phenomenon to be modelled.
The conceptual model is the mathematical or logical representation, which is developed through an analysis and modelling phase, while the computerized model is an implementation of the conceptual model on a computer.

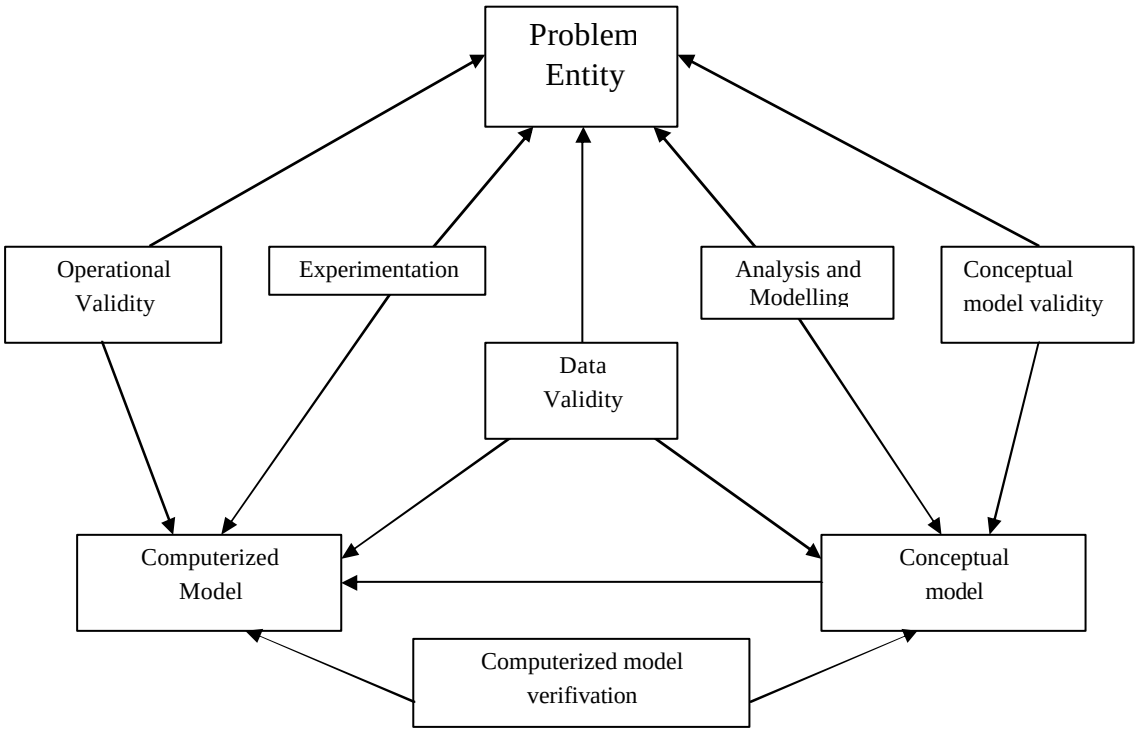

Figure 1: A simplified version of the modelling process [4].

Therefore, conceptual validity as defined by Sargent refers to the determination that the theories and assumptions on which the conceptual model is based are correct and ensures reasonableness in the model representation of the problem entity. Operational validity is the substantiation that the output behavior of the model is adequately accurate in its applicable area; while data validity is ensuring that the necessary data for model construction, testing, experiments and evaluations are sufficient and correct.

In summary, Sargent recommended that as a minimum practice in model validation procedure, the following steps should be carried out:

- Extensive testing of the assumptions and theories underlying the model
- A face validity (this means asking people who are knowledgeable about the system if the structure of the model and behavior are reasonable or not) of the conceptual model should be done
- Before developing the model, the model users, sponsors and developers must agree on the basic validation approach and techniques to be used.
- Validation results should be presented in the model documentation
- An exploration of the behavior of the model should be made in each iteration of the model using the computerized model.
- As extensively as possible, for at least two sets of experimental conditions, comparison should be made between the model and system output behavior data

For large systems involving subsystems and components, the validation information at component-level could be used to make inference at system-level where full-scale test data is not available. In [5] we see a *statistical* approach to the validation of simulation models. Also the author describes three basic approaches namely:

- *Comparing graphs of the output data:* Graphical plots of different sets of experiments with the model and system's output behavior data are compared so as to substantiate if the model's output behaviour has enough accuracy for its given purpose. The graphs are utilized for validation in three ways:

  - Experts could use them in face validity technique to make a subjective judgment, as to whether or not the model has enough accuracy for its given purpose.
  - The developers in the development process to determine sufficient accuracy for validation could use the graphs
  - They can also be used in the Turing tests.

- *Hypothesis Tests*: this is used in comparing distributions, parameters and time series of the output data of a model and a system for each set of experimental conditions to ascertain whether the output behavior of the model is within an acceptable range of accuracy.

  However, two kinds of errors could be obtained here. The first type is rejecting the validity of a valid model while the second is accepting the validity of an invalid model. In validation, this second type of error is very important and must be kept as low as possible.

- *Confidence intervals:* The authors state that confidence intervals, *ci*, joint confidence interval *jci* and simultaneous confidence interval, *sci*, can be obtained. It is always desirable to build the model range of accuracy with lengths of the *ci* and *sci* and the sizes of the *jcr* as small as possible. The shorter the lengths or the smaller the sizes, the

more useful and meaningful the specification range of the model range of accuracy will usually be.

Static system behaviors presents less surprises and therefore generally does not require simulation, however, in view of dynamic process behaviors, that is, events occurring with the passage of time, there are several interesting observations that need to be considered.

Thus in their work [6], J. F. Hetet et al made strong points about the problems inherent in *continuous discretized models and digital simulations.* It is their opinion that since the digital computer is discontinuous in its operation mode, it will be unable to properly evaluate the variables that must simulate natural phenomena, which are mainly continuous. Hence it can only give periodic samples of these variables.
For a particular machine, the time needed to calculate these variables depends on how complex the computation (activity) may be. For instance, if results are requested every 10ms, the computer could require 2ms at some point and another 3ms at some other point (example context switch time), such that the real-time essence or notion is lost! Therefore most times digital simulations could run slower than the real process with no given possibility of mastering the running speed. This is quite peculiar with real-time programs, where computation of the *worst-case execution time* is often a nightmare.

They further observed that this distortion with respect to the continuous model could alter the course of the simulation considerably, especially if the sampling is not properly done. For instance in the case that simulation changes in each state is a different task for the digital computer, and in order to simulate the model relation, the tasks have to continually communicate with each other, which take place only at sampling times. In between two of these times, the computer could consider that the variables are constant.
Several other interesting schools of thought abound concerning validation.

2.2 General validation techniques

In [7], Balci identifies the following general models' validation techniques, which can be used either subjectively or objectively. These techniques are used both to validate sub-models and overall models and also used often in computerized verification. These include:

- *Comparison with other models:* Several outputs of the model that is being validated are compared with results of the other 'valid' models.
- *Degenerate tests:* The degeneracy of the model is tested by removing portions of the model or through appropriately selecting the values of the input parameters.
- *Face validity:* this means asking people who are knowledgeable about the system if the structure of the model and behavior are reasonable or not.
- *Event validity:* The events of occurrences of the simulation model are compared with those of the real system to determine if they are similar.
- *Extreme conditions test:* Here the structure of the model and output should be plausible for any extreme and unlikely combination of levels of factors in the system.
- *Historical data validation:* If data are collected from the system before constructing the model, some of this data will be used to build the model while the rest would be used to ascertain if the model behaves as the system does.
- *Internal validity:* Various replicas of a stochastic model are made to ascertain the quantity of stochastic variableness in the model. A high amount of variability (lack of consistency) may make the model's output to be questionable and, if typical of the

problem entity, may question the appropriateness of the policy or system being investigated.

- *Multistage validation:* A proposal for combining the three historical methods of rationalism, positive economics and empiricism into a multistage of validation was brought to light by Naylor and Finger in 1967. This method deals with developing the models assumptions on theory, observations and general knowledge and intuition. The models are validated where possible, through empirical testing and testing the input-output relations of the model with the real system.
- *Parameter variability (sensitivity analysis):* This has to do with varying the values of the input and internal parameters of a model and its output. The same relations should occur in the model as in the real system. The parameters which are sensitive (cause noticeable changes in the behavior of the model) need special attention in the parameter estimation phase.
- *Predictive validation:* Here the model is used to forecast the behavior of the system and comparisons are carried out to know if the behavior of the system and the model's forecast are the same.
- *Traces:* The idea here is to determine if the necessary accuracy and model's logic is correct by tracing different types of specific model behaviors.
- *Turing tests:* Opinion of knowledgeable experts on the systems are asked to see if they can discriminate between the system and model outputs.

However, in all the work mentioned above, neither specific techniques, nor processes have been duly defined specifically for real-time systems or particular inference studies done in line with the thrust of this thesis.

Conceptual model validation
The aim of conceptual model validation is ensuring that the assumptions and theories on which the conceptual model is based are correct. It also deals with the fact that the problem definition, structure of the model, logic and casual relationships are reasonable for the purpose for which the model was created.
The frequently used validation method in this case is the *face* validation and *traces* methods. In the event that errors are observed in the conceptual model, it is revised and conceptual model validation is carried out again.

Operational validation
Here one is interested to know if the model's output behavior has the accuracy required for the model's purpose over the domain of the model's intended application. It is at this phase that majority of the evaluation and testing are carried out. All the techniques above that have been mentioned are applicable to operational validity.

2.3 Common sources of errors in a Model

Assumptions and approximations during modelling and simulation could induce errors in the prediction made based on the model. In software system, model building involves several activities and could be error prone at any of its stages. In [8], Johan Andersson et al discussed four potential error sources:

- *Understanding of the system*: Poor understanding of the target system could introduce modelling errors. Therefore there must be a good understanding of the structure and

behavior of the system by the modelling team, so that they can build a valid and robust model. Errors could be avoid by due consultations with experts, who should review the resulting model.

- *The modelling language*: The risks of misinterpretations and misapplication are associated with poor knowledge of tools for modelling, analysis and semantics of the modelling language. This error could be well avoided by appropriately documenting and communicating tools and modelling languages.
- *System's observation*: In order to ensure that the system behavior is captured as much as possible, an observation-based model must have observations taken at several representative situations. A good example would be comparing a static system to a dynamic system's observation.

  Also factors like the addition of probe effects have to be given due considerations if used for measurements. This is essentially so for real-time systems where probe could affect the system's temporal behavior and could cause or prevent exceptional events such as a missed deadline.
- *Levels of abstraction*: Robustness and accuracy would be greatly reduced if information on relevant details about the system's behavior were omitted. In this case, a sensitivity analysis would be helpful.

2.4 Model Robustness

A model can be said to be robust regarding a given variation in the system's implementation if, in the event that this variation is applied to the model, it gives the same effect on the predictions as it would with the observed system behavior.
This means that important system behaviors and semantics have been duly captured in the right level of abstraction [9]. It could be verified using the sensitivity analysis.

A good illustrative example of the importance of robustness is to consider the scenario in which, a system containing a binary semaphore is protecting a shared resource. Here, a timeout would occur when a task has been on the waiting queue for a semaphore for a given predefined time value. At the occurrence of the timeout, the task's time of execution will be increased as a result of error handling. This timeout previously has not had a reason to occur in reality following previous versions. If however, when modelling this system, the timeout possibility is omitted, the model could still seem to be accurate. But in the event that the timeout occurs, one will observe a divergent system behavior from that which has been predicted by the model.

3 Philosophical perspectives on models validation

Considering the questions: '*what does it mean to validate concepts? Or what are the criteria? Both philosophers and scientists have been unable to agree about the answers to them* [10]. Since a model is often taken to be an abstraction and simplification of reality (real system being modelled), but reality (the nature of measured data, environmental and human factors) in itself, has a nature of abstract complexity; a 'correct' model could at best be judged as one which is 'closest' in representation to the real system, but the question are: just exactly how close should 'closest' be to be correct? Are models true? Can truth emerge from a 'truthless' model? In essence, what do we need – a correct model or one that yields information? [11]. In this Section, we shall examine these crucial questions arising from the validation criteria of models that have been mentioned above.

3.1 Interesting views

Computer science is faced with the above difficulties more than other disciplines because of its diverse constituents, ever-changing contextual environment (technology), and relatively short life span. Validation assures that a model (or each construct in a conceptual model) contains the features imputed to it in their individual definitions or description. In other words, validity implies that it is well grounded, sound or capable of being justified.

The response of a computer science empiricist to the question "How do we validate?" could be to design *an experiment or build a prototype and test your concept or conceptual model*. But, a fundamental problem with this approach, notwithstanding the assumptions inherent in statistical experimental design, is the presupposition of the "validity" of a concept or conceptual model. That is, a belief in the notion that mere definition implies that a concept has "face validity." If simply using a "term" made it acceptable to a discipline, one would never reach an agreement on commonly held truisms or knowledge of that discipline.

Simulation models are believed across disciples to give *information* on the real system. In [12], a 21st century philosopher Luciano Floridi defines information as basically comprising 'meaningful content and truth'. In this thesis we argue that this definition to a large extent does not apply to computer science.

In philosophy, there is a huge difference between truth and correctness. While truth is an absolute, correctness is relative to the system. For example, if you read a book on the philosophy of mathematics, "truth" is not the issue because mathematics does not deal in truth but deals with provability. Maybe physics deals in truth, because the job of science and engineering is to understand the world as it is. Thus the issue for consideration here is *correctness*. Hence an important question to ask in this context would be: *can simulation models yield knowledge about the real world?*

The epistemological importance of this question is such that if the answer is *no*, then what many scientists are doing nowadays is just playing with computers, not creating new knowledge! However, considering the practical importance of that question, if no is still the answer, it means that the several policies, which are based on predictions from simulation models, would grossly be misguided. It is interesting to note however that even in the field of philosophy, varying opinions about towards whether verification and validation are possible or not.

In [13], an interesting philosophical argument ensues between Oreskes et al and Fredrik Suppe in trying to proffer solution to this seeming deadlock. Oreskes strongly argues that simulation models cannot be verified and hence Scientists cannot obtain knowledge from simulation modelling. On the contrary, Fredrik Suppe retorts that simulation models can be verified in some sense and hence knowledge could be obtained from them. Some important issues that readily comes to mind in this case would be a deep consideration of some epistemological questions such as:

- What do we learn from experience?
- What is the correct way of learning from experience?

There are several traditional philosophical views to these, which include Inductivism (enumerative induction, inference to the best explanation and Bayesianism) and falsificationism.
However, Oreskes argues the above, utilizing traditional philosophical debate over inductivism. Their criticism of the traditional view in 3 different areas stemmed from Hume's problem of induction, which says that:

1. *All inductive reasonings are based on the assumption of uniformity:*

What we have observed and what we haven't yet are basically similar. According to him, the question would be: 'why can we rely on such an assumption?' Nothing we have observed until today does not assure that the same regularity will hold tomorrow (unless we use induction --- this is a circular argument).

2. *Underdetermination:* - Given any amount of evidence, there are mutually incompatible theories, which would equally fit with the evidence, that is, when a prediction from a theory contradicts with the observation; there are various mutually incompatible ways for making the theory compatible with the evidence.

3. *Theory-ladenness of observation:*

These philosophical views presuppose that our observation is somewhat independent from our scientific theory. But what we see is strongly influenced by our background knowledge and assumptions. A common example would be asking a zoologist and a computer scientist to give interpretations to the diagram of a rabbit.

Why do we care about theory-ladenness of observation?

This is because a conflict between two incompatible theories is supposed to be settled by conducting some experiments or making observations. However, theory-ladenness can cause a serious problem with such a procedure.

Considering the Underdetermination vs. Theory-ladenness, the difference between the underdetermination thesis and theory-ladenness can be summarized as follows:

a) Underdetermination
Same evidence -> Incompatible theories

b) Theory-ladenness:
Incompatible theories -> Different evidence.

In the actual sense, arguments by Oreskes are an application of these traditional criticisms of induction to simulation models.

3.2 Degrees of certainty

An interesting categorization was projected by Oreskes in which the following distinctions were made as various degrees of certainty:

- Absolutely true (logical truth) i.e. verification
- Plausible, probable (in terms of evidence) > confirmation
- Consistent (not contradictory) > validation

They conclude from their philosophical analogies given above and deduced that:

(a) Models *cannot* be verified in that there is no logical proof that a model is true.

(b) Models *can* be validated, this means that we can prove that a model does not contain a detectable flaw and thus internally consistent. These can be evident in:

Comparisons of different solutions:

If two totally different ways of solving a same problem give the same answer, these ways of solution may be reliable.

Calibration:

Adjust initial values so that the model can accommodate known data. These procedures are far from *verifying* the model.

Models may be confirmed

Models may yield predictions that match with observation, but this means only that the model is probable, not that the model is true.

Therefore from the above analysis, Oreskes further concludes:

- That the primary value of a simulation model is heuristic, that is to give evidence to strengthen what may already have been partially established through other means for instance, sensitivity analysis, or even challenging existing formulations.
- A simulation model is a `fiction'. It is never a `real thing'. (Cartwright).

However, in contrast to the above views, Suppe assumes a less strict philosophical stance as follows:

(1) 'It is true that we cannot logically prove that a model is true. But maybe their (Orseke's) way of defining 'verify' is too strict. Do we really want that absolute certainty? That makes all empirical knowledge impossible'.

(2) Extra factors can affect the result. But still a simulation model is creating knowledge about the real world when the system is isolated or other factors are negligible.

(3) Don't take underdetermination too seriously. Often it is hard to find even one reasonable solution.

(4) Don't take assumption-ladenness of simulation models too seriously, either.

(5) An important aspect of modelling is the mapping relationship between three systems. As far as this mapping relation holds, a simulation model is a representation of that aspect of the real world, not just a heuristic tool.

With view to the above two major open and highly contestable areas, one could strike some good balance by answering the following questions:

- *What level of certainty do we want for scientific knowledge?*
- *Can simulation models provide that level of certainty?*

In [14] Khazanchi attempts to integrate notions from the philosophy of social sciences, the information systems (IS) field and its referent disciplines and sets forth a framework for the validation of IS concepts. The proposed philosophical framework for validation of concepts and conceptual models consists of a set of "criteria for validation" of concepts.
He asserts that as a concept fulfils each succeeding criteria its potential ability to have inherent "truth content" with regard to its general acceptance in the field strengthens. After all, "... concept formation and theory formation in science go hand in hand.... The better our concepts, the better the theory we can formulate with them, and in turn, the better the concepts available for the next improved theory." [15]. The following are his suggested criteria for such validation:

1. Is it plausible? A concept or conceptual model is plausible if it has face validity. Plausibility establishes that this model is more than just a belief. This criterion is useful to assess the apparent reasonableness of an idea and could be demonstrated by deduction from past research or theories, or, it could be developed on the basis of observation or induction.

2. Is it feasible? This criterion dictates that a concept or conceptual model, at the least, has the quality of being workable. Added to plausibility, a feasible concept or conceptual model would be operational in that it would be amenable to verbal, graphical, mathematical, illustrative, prototypical characterization.

3. Is it effective? This criterion deals with the question: How effectively does the model describe the phenomena under study? Also an effective concept or conceptual model has the potential of serving our scientific purposes [16]. It also guides and stimulates other scientific inquiries.

4. Is it pragmatic? The pragmatism criterion dictates that a concept or conceptual model should not be restrictive to the extent of logically excluding previously valid models. Thus, this criterion provides that concepts or conceptual models should subsume, for obviously practical reasons, any conceptual structures that previously explained related phenomenon. Hunt [1990] demonstrates this criterion with the example of Newton's law. He argues that simple pragmatism would require that any new conceptual development could not preclude Newton's laws (as in the case of Relativity, where these laws are a special case subsumed within relativity). In effect this criterion emphasizes that concepts and conceptual models should have some degree of abstract, logical self-consistency or coherence with other concepts and conceptual models in the discipline.

5. Is it empirical? Empirical content implies that a concept or conceptual model must be "empirically testable" [17]. In the same vein, Dewey affirms that although concepts can be developed without reference to direct observation, and although this logical conceptual development is indispensable to the growth of science, the ultimate test of a concept or conceptual model lies in having the ability to empirically collect data to "corroborate" it. According to Dewey [1933, p. 183], "Elaboration by reasoning may make a suggested idea very rich and very plausible, but it will not settle the validity of that idea.

6. Is it predictive? Does it explain a phenomenon that is expected to occur? We can better understand the meaning of this criterion through the words of Rashevsky (1954, p. 152-3): "A theory or theoretical concept is considered the more convenient or useful, the better it enables us to predict facts that hitherto have not been observed... The scientist constructs theories, theoretical concepts or theoretical frames of reference that are isomorphic with the world of observable phenomena. This isomorphism is never complete, never covers the whole range of observable phenomena... wider the range of isomorphism, the greater predictive value of the theory." Thus, a concept or conceptual model that is predictive would, at the least, demonstrate that given certain antecedent conditions, the corresponding phenomena were somehow expected to occur [Hunt, 1990].

7. Is it intersubjectively certifiable? Hunt [1990], Nagel [1979], and several others are of the opinion that all scientific knowledge, and in consequence, concepts or conceptual models "must be objective in the sense of being *intersubjectively certifiable*." This criterion provides that concepts or conceptual models must be "testable by different investigators (thus inter-subject)." Investigators with differing philosophical stance must be able to verify the imputed truth content of these concepts or conceptual structures through observation, logical evaluation, or experimentation.

8. Is it intermethodologically certifiable? In addition to being intersubjectively certifiable, this related criterion provides that investigators using different research methodologies must be able to test the veracity of the concept or conceptual model and predict the occurrence of the same phenomenon.

3.3 Ultimately?

Ultimately can we gain information from models?
The Cambridge Dictionary of Philosophy defines information as:

> '*an objective (mind independent) entity. It can be generated or carried by messages (words, sentences) or by other products or cognizers (interpreters). Information can be encoded and transmitted, but information would exist independent of its encoding or transmission*.

Suffice it to say that computer scientists view and define information basically *as meaningful data*. Meaningful in the sense that it can be read, sensed or perceived (physical and non-physical data). This *meaningfulness* is contextual such that it makes sense to the informee from the informer (any medium applied) and satisfies basic operating codes of understanding within its domain of applicability.
The implication of this definition is such that, for instance, what constitutes data that makes sense within the programming semantics of complex system domain and thus yields information may not necessarily be informative in other domains.
Do we really want absolute certainty? This would imply that all empirical knowledge is impossible. However since knowledge and information are essentially different in their basic forms and definitions, with knowledge of necessity made up of truth, one can say that in as much as knowledge deals with the **absolute** sense, we can gain information from models, which does not necessarily need to be true, on which basic improvements to the systems in question can be made.
Scientifically speaking, simulations models are instruments that yield information irrespective of whether the information is true or false about the real system. It is our conviction that though valid models could be 'probable' and not 'true' and therefore may not serve as sole basis for decision making, valid simulation models contain adequate results and data on which

very meaningful conclusions and inferences could be drawn; hence they yield information about the real system.

4 Validation of models of real-time systems

As have been mentioned in the sections 1.3 and 4 above, one of the major validation challenges for software models is that computer software has a non-continuous nature, which implies that there could be drastic changes encountered as a result of little variations in the system unlike what may be obtainable in models in other fields. For real-time systems, however, an even bigger challenge is the effective modelling and validation of the system's temporal properties, which assumes the centre stage especially for safety-critical real-time systems.

In this Section, we shall review some works that have been done in this area with the aim of validating models of real-time systems and useful properties that enhance time relatedness.

4.1 Hybrid method

In their work, Laurent Kaiser et al [18], presents a hybrid method of validation for real-time systems. They propose a method based on a unique representation of the complete systems that combines simulation with actual analysis restricted to critical parts. The underlying formalism is an offshoot of Timed Automata called Timed Input Output State Machine (TIOSM), (Kone et al, 1995).

For the above reason, they define four kinds of models namely:

- *Complete models*: The initial model of the whole system in terms of TIOSM.
- *Partial models*: This is a part of the complete model, which is assumed to represent a critical part of the behavior of the whole system.
- *Black box models*: The abstraction of the partial model obtained after its validation
- *Derived models*: The result of the integration of all the black boxes in the complete model.

The two main stages involved are the validation of the critical parts and then the validation of the system as a whole.

4.2 Use of model equivalence

Another proposed method of validation has to do with using model equivalences. Andersson et al present a notion of equivalence as a relation that enables the comparison between the temporal behaviors predicted during model analysis and the temporal behavior observed when executing the system. However, they argue that due to the known fact that models are abstractions of the system, the predicted behavior will also be an abstraction of the system behavior, and therefore direct comparison of the predicted behavior with observed behavior will not be feasible.

Hence they insist that *observable property equivalence* could rather be attained, in which case it is determined that the system and the model are *equal* in relation to a given set of system properties characterizing the temporal behavior of the system.

In their experimental framework, they defined system property as a probabilistic statement in relation to an aspect of the system behavior that could be directly observed or derived from observations of the system, which may not explicitly be found in the system implementation or configuration. This definition therefore excludes system properties such as tasks priority, tasks execution times and even the rate of periodic tasks, some of which could be found in the implementation and others calculated by tools. Therefore good examples of system properties for instance whether a task response times is less than a specific deadline or probability of a message queue being full or empty.

Using this method, an important task would be the selection of relevant properties for the comparison. The properties should be both relevant and varied so as to capture different aspects of the temporal behavior. Four types of timing related properties include:

- *Response time properties*: This depends both on the execution time of the task and on the temporal behavior of the other tasks.
- *Event pattern properties*: It is often possible to recognize patterns in the execution of tasks and arrival of events. This pattern of occurrence is a system property that can be used for comparison.
- *Synchronization properties*: These are related to semaphores and their effects. These kinds of properties could help state the absence of timeouts and deadlocks.
- *Message buffer properties*: Includes those properties related to message buffers, for example, length of time a task waits for a message, how often a task reads or writes messages from the buffer.

Caution should be taken such that too few types of these properties are not used, which could breed the risk of accepting an invalid model.

5 Modelling tools and analyses methods

Whereas simulation methods have been widespread in its usage, unfortunately many tools that are used for constructing simulators do not deal with real time issues. This has constituted a major deterrent in acceptability of modern scheduling and resource management algorithms in industry.

Very few simulation tools are designed for real-time systems and among these few, none addresses adequately, the issues peculiar to complex real-time systems. In this Section, therefore we shall investigate some of these existing tools.

5.1 UPPAAL

Uppaal is an integrated tool environment for modelling, simulation and verification of real-time systems [19].

It is good for systems, which can be modelled as a collection of non-deterministic processes with finite control structure and real-valued clocks that are communicating through channels or shared variables [20].

Some major areas where this is applied include real-time controllers and communication protocols especially those in which timing aspects are critical. The editor in Uppaal's graphic user interface is as shown in Figure 2.

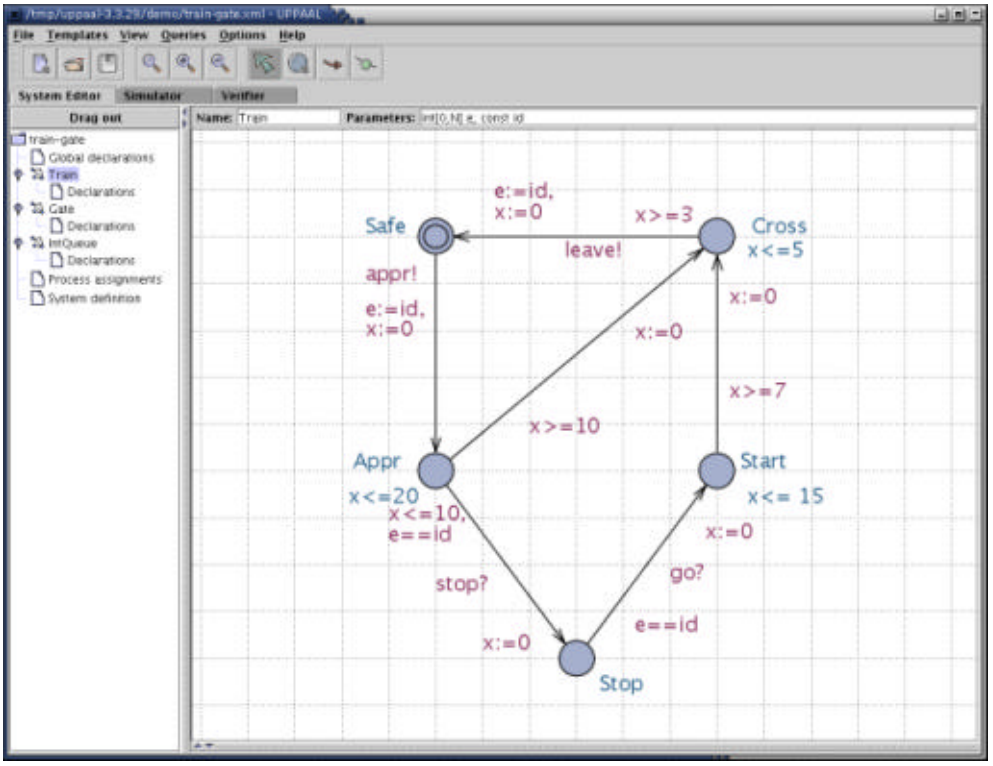

Figure 2: The editor in Uppaal GUI

Uppaal has three major parts namely:

- A descriptive language
- A simulator
- A model checker

The descriptive language is a non-deterministic guarded command language with data types such as arrays and bounded integers. This is the modelling or design language, which is used in expressing the behavior of the system as networks of automata with clock extension and data variables.
The simulator is a validation tool that enables the observation of possible dynamic executions of a system at the design on-set or modelling stages. It provides a non-expensive way to fault detection before verification by the model checker that covers a more exhaustive dynamic behavior of the system.
The model checker is responsible for checking invariants and properties for reachability by an exploration of the system state-space; this implies reachability analysis in terms of the symbolic states denoted by constraints.

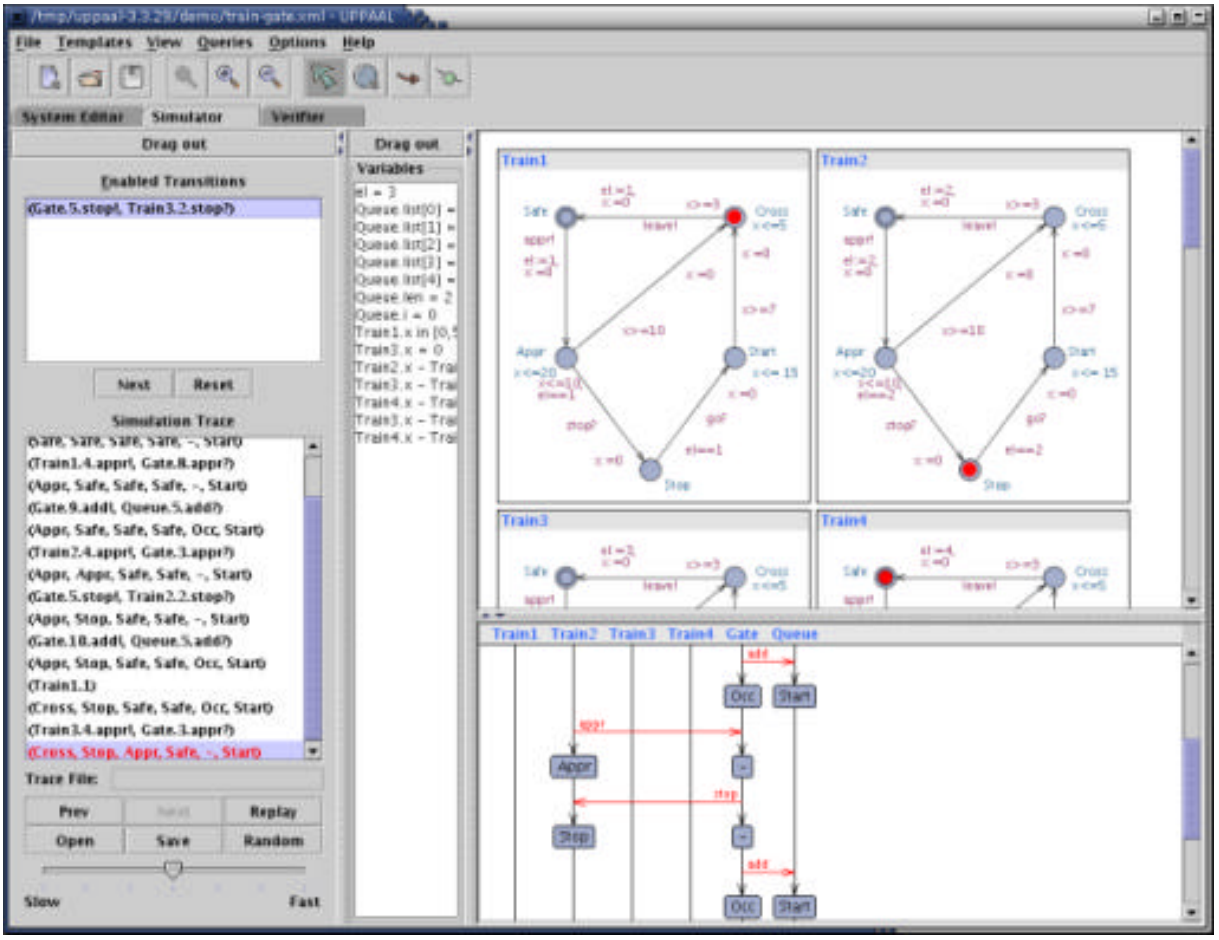

Figure 3: The simulator in the Uppaal GUI

For the Uppaal tool, there are two major design priorities namely:

- Efficiency
- Ease of usage

For efficiency, the incorporation of the *on-the-fly* searching technique has been important for the model-checker. Efficiency is enhanced by the application of a symbolic technique, which lowers the problems encountered in verification to that of efficient manipulation and solving of the constraints [21, 22, 23, 24].
In order to model or debug, the Uppaal model-checker could generate a *diagnostic trace*, which could interpret the reason for which a property is either satisfied or not satisfied through the system description. These diagnostic *traces* that are produced by the model-checker could be automatically loaded to the simulator, which could be done while investigating or visualizing the trace.

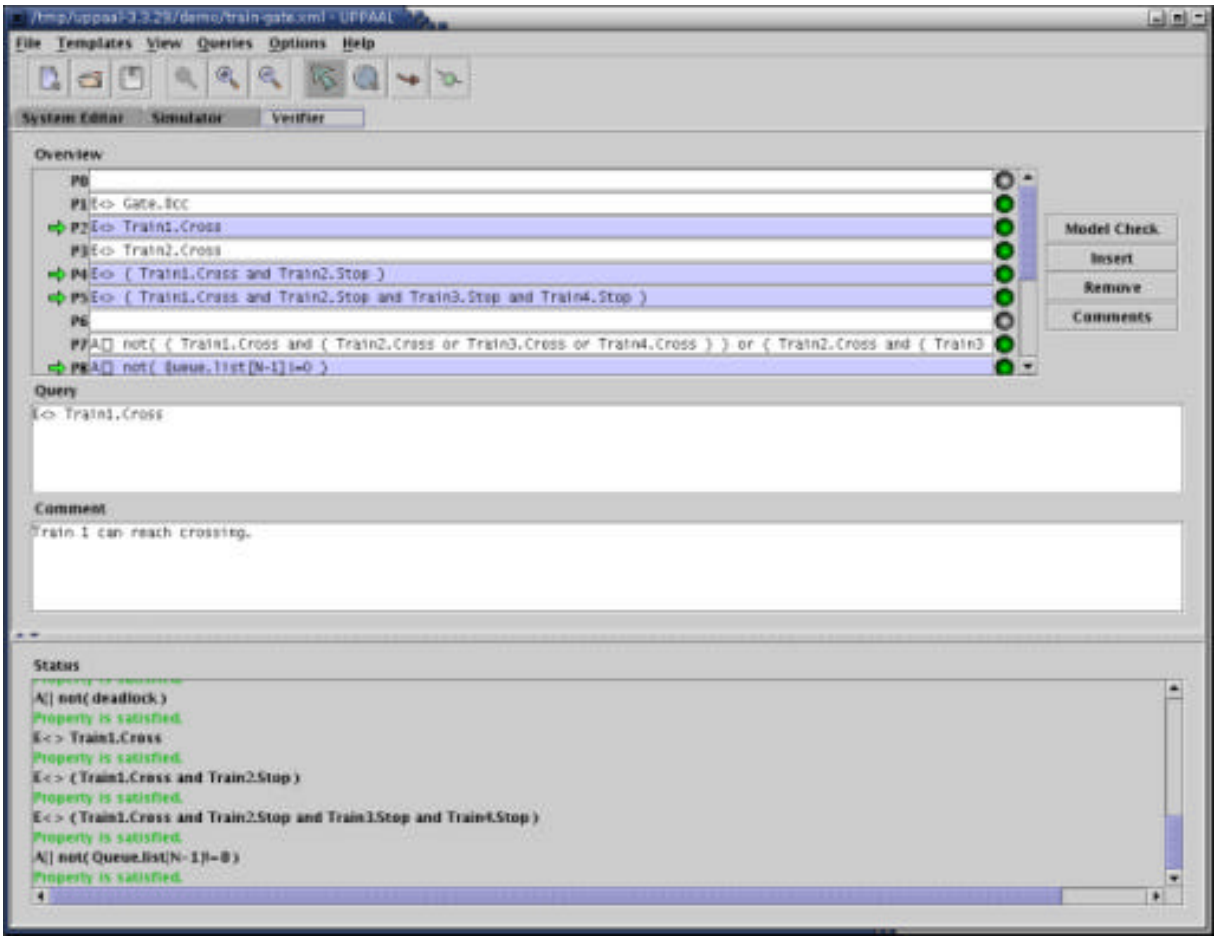

Figure 4: the uppaal verifier GUI.

Over the years, there have been improvements on Uppaal's versions, with various extended features.
In a nutshell, the major features of Uppaal2K include:

- An editor where requirements are specified, which also has a graphical user interface to the verifier.
- A graphical system editor that allows the description of the system graphically.
- A graphical simulator that provides graphical visualization and recording of the possible dynamic behaviors of a system descriptor, that is, the sequences of symbolic states of the system. This could also be used to visualize traces produced by the model-checker.

- A model-checker for an automatic verification of safety and bounded-liveness properties by reachability analysis of the symbolic state-space.
- The generation of traces in case the verification of a given real-time system fails. The diagnostic traces may be automatically loaded and graphically visualized using the simulator.

The major limitation with Uppaal is known as the state space explosion problem (inability to adequately model and simulate all states of the target system especially for very large systems).

At the moment, research works in this state explosion area are focused towards addressing the memory capacity challenges faced in using Uppaal so as to effectively capture all possible states of large systems. Also to study data properties, it becomes rather awkward (and inefficient) to model complex data structures in Uppaal.

5.2 TIMES

The word *Times* is an abbreviation of Tool for Modelling and Implementation of Embedded Systems.
Times is a modelling and schedulability analysis tool for embedded real-time systems.
It is appropriate for systems, which can be described as a set of pre-emptive (can be interrupted) or non-preemptive (cannot be interrupted) tasks that are triggered periodically or sporadically by time or external events. It provides a graphical interface for editing and simulation, and an engine for schedulability analysis.
Times have many features, mainly:

- A graphical editor for timed automata extended with tasks [25]. This allows the user to model a system and the abstract behavior of its environment.

  Also, the user could specify a set of preemptive or non-preemptive tasks with parameters like the relative deadline, execution time and priority.
- A simulator where the user could validate the dynamic system behavior to observe how the tasks execute according to the task parameters and a specified scheduling policy.

  The simulator displays a graphical picture of the generated trace showing the time points when the tasks are released, invoked, suspended, resumed and completed.
- A verifier for schedulability analysis, that is used to check if all reachable states of the complete system are schedulable, that means, all task instances meet their deadlines.
- A code generator for automatic synthesis of code from the model. If the automata model is schedulable according to the schedulability analyzer the execution of the generated code will meet all timing constraints specified in the model and the tasks.

A representation of Times is shown in Figure 5 below:
In Times, the modelling language and theoretical foundation is based on the model of timed automata with tasks.

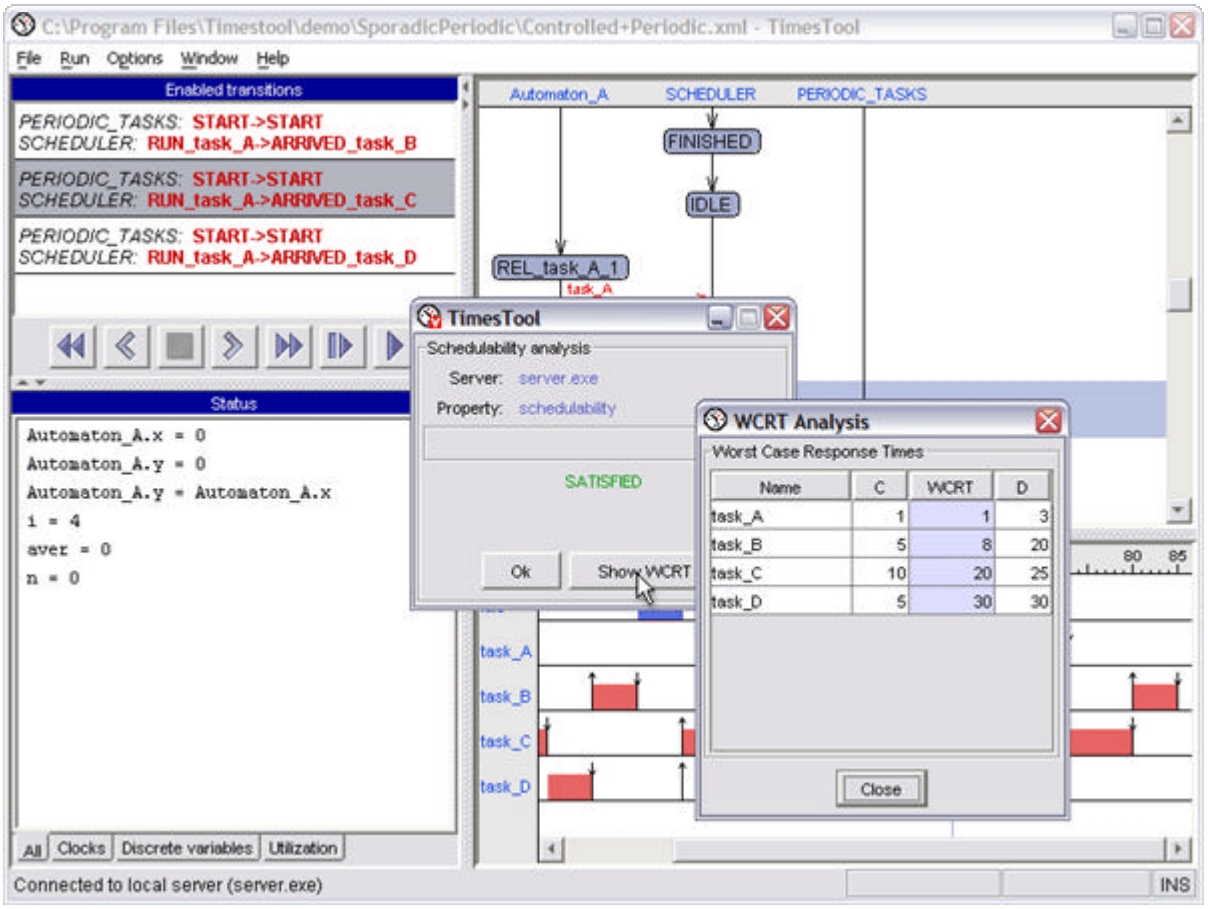

Figure 5: A representation of TIMES tool

5.2.1 The input language

The center of the input language in Times is the timed automata with real time tasks (TAT), with shared variables between the automata and the tasks.

A TAT is a timed automaton that is extended with tasks that are triggered by events. A task in this case is just an executable program with priority, deadline and worst-case execution time.

A task could update a set of variables using assignments in the form x :=E where x is a variable and E is an expression (the value of E is returned when the task is completed). These variables could be changed and tested by the automaton.
An edge leading towards a location in the automaton shows an event triggering the task, and the guard (clock constraints) on the transition specifies the possible arrival times of the event. This then helps in the description of concurrency and synchronization, and real time tasks, which may be periodic, sporadic, preemptive, and (or) non- preemptive.
Therefore, an automaton is schedulable if there exists a pre-emptive or non-preemptive scheduling strategy such that all possible sequences of events accepted by the automaton are schedulable, meaning that all associated tasks can be computed within their deadlines.

In semantics, an extended automaton could do two kinds of transitions as a standard timed automaton would, but the difference lies in the delay transitions, which corresponds to the execution of running tasks with highest priorities or earliest deadlines, and idling for the other tasks waiting to run. Discrete transitions tally with coming of new task instances. Whenever a

task is triggered, it is put in the task queue for execution (corresponding to the ready queue in operating systems).

The scheduling problem of the TAT then is to verify that all released tasks are guaranteed to always meet their deadlines when executed according to a given scheduling policy. In Times, this analysis is performed by transforming a TAT system into ordinary timed automata extended with subtraction operations on clocks, and encoding the schedulability problem to a reachability problem.

5.2.2 Tool overview

The graphical layout of TIMES is represented in Figure 6.

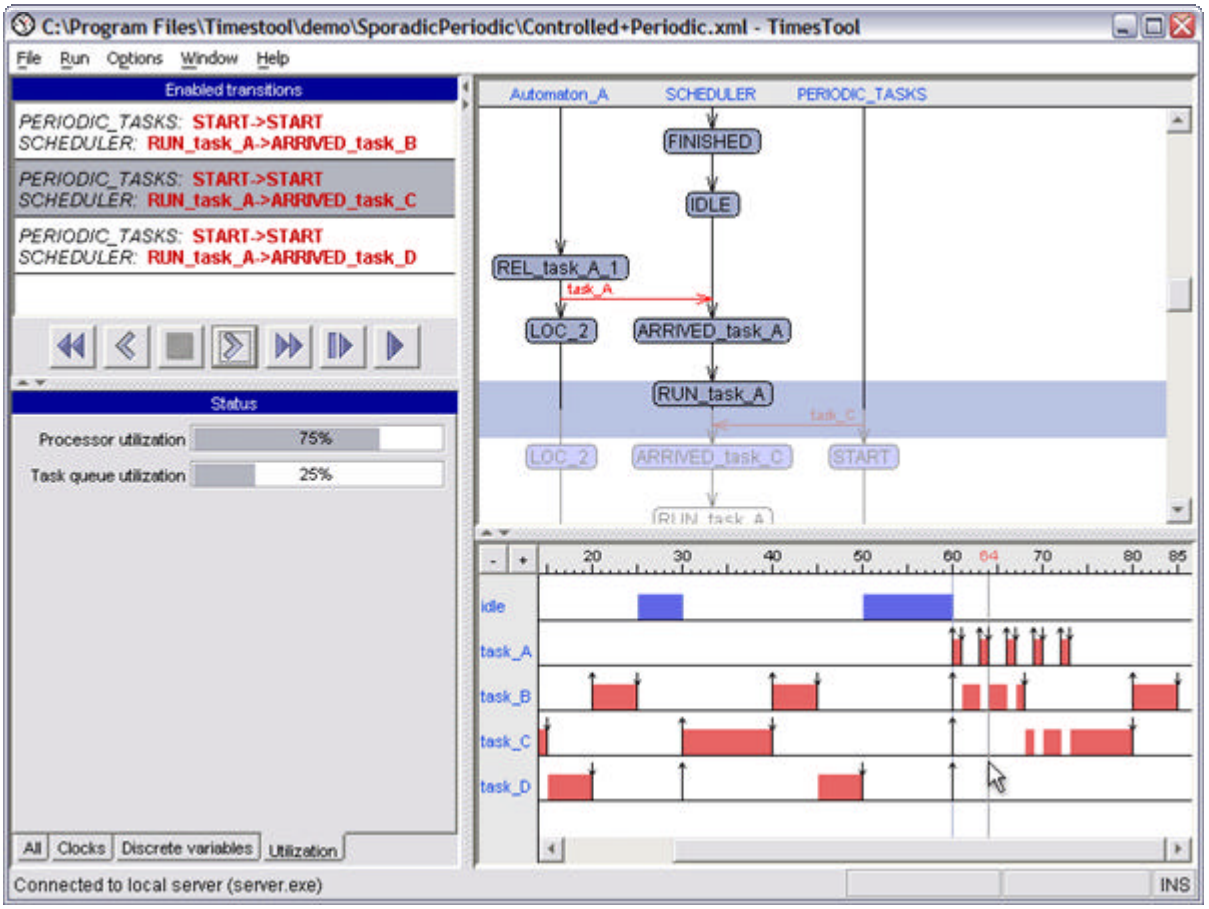

Figure 6: An overview of Times tool simulator

This tool mainly consists of three parts, which are:

- System specification
- System analysis
- Code generation

The system specification area is further sub-divided into the *control automata* which is modelled as a network of timed automata extended with tasks, *a task table* which bears

information about the released processes in the event that the control automata changes location, and a *scheduling policy*.

The system editor tool is used for drawing the control automata of the system model. It shows a table for defining the task parameters. The task parameters that are supported currently by this tool include: relative deadline, execution time, period, priority, a reference to the task code and a field indicating the task behavior. The currently supported scheduling policies are: first-come first served; fixed priority, rate monotonic, deadline monotonic, and earliest deadline first, which could all be preemptive or non-preemptive.

The outcome of the *system editor* is an XML representation of the control automata. The information from the task table and the scheduling policy are used by the *scheduling generator* to generate a scheduler automaton that is made in parallel with the controller automata to ensure that the system behaves according to the scheduling policy and the task parameters.

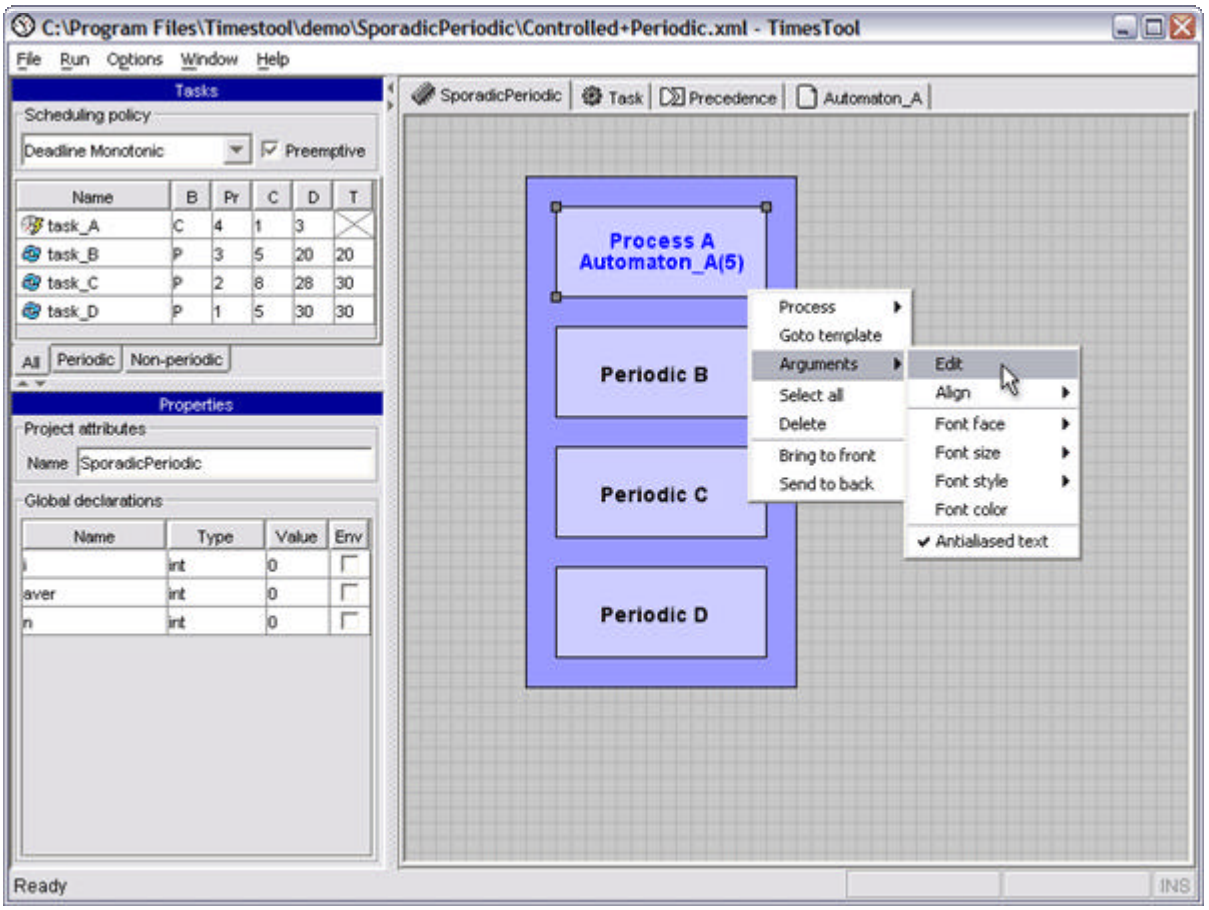

Figure 7: Times project editor

If the scheduling policy is non-preemptive, the scheduler automaton is an ordinary timed automaton. However, in the event that the scheduling policy is preemptive, the scheduler automaton is modelled as a variant of timed automata in which clock variables may be updated by subtractions. This analysis too also suffers the same state explosion problem as is in Uppaal.

5.3 The probabilistic Modelling and analysis Framework

In his work, Anders Wall described a probabilistic modelling and analysis framework, where simulations are based on the analytical model of the system made in their probabilistic modelling language ART-ML (Architecture and Real-Time Behavior Modelling Language). In using simulations, other correctness criteria than just satisfying the deadlines are duly defined. Also, execution time distributions could be used instead of assuming a worst-case scenario. The ART-ML allows the modelling of the tasks behavior on a lower level than on the software architecture. This gives rise to the creation of a more precise model as semantic relations among components can be introduced.
In the above framework, the requirement language known as the Probabilistic Property Language (PPL) is used so as to express statistical requirements, which are verified given the simulation results. The advantage of this is that there is the possibility of getting more feedback information from the analysis in probabilistic terms instead of the normal schedulable (yes) or not schedulable (no) cases.

A major reason for the probabilistic property language is to analyze the impact of changes made in a real-time system. The aim is its use to formulate probabilistic queries. Probabilistic queries here could be seen as a relational operation on two probabilities.
When modifications are made to a model and the model is run, large sizes of execution traces are created and manually gathering any information from them becomes a futile effort. Therefore the PPL comes handy with respect to the size of these traces.
Properties such as task and message queues can be extracted as probabilities of fulfilling some requirements using the PPL. For example, to know whether a task, t, always meets its deadline of 15 time units, would be ask whether the probability of the response time of task t, being less than 15 is 1. This therefore can be simply checked from the traces (values collected during simulation), whether the response time of all instances of task t, are actually less than 15.
The probability function, P ( ), forms the core of PPL. This is mainly the basis of all the probabilistic queries because any query without P would only compare constants. The first argument is the set, the task or probe, while the second argument is the condition of the query. If the set is a task, then the returned result is the probability of some instance of the set fulfilling the condition. However, if the set is a probe, then the result is the probability over time. It is important to note that since this probability is based on the observations in a trace, it is only an estimate of the true probability. It can be typified as follows:

P(<set> , <condition>)

Additionally, PPL could contain bounded variables, which could be used in feeding back values to the user. This variable could be part of the condition in a P function or as one operand in the outer relational operation of the query.
For instance, an unbounded variable used in the condition of the query could be: *what is the deadline met with a probability of at least 0.7*?

P(t(i), t(i).resp < Y) > 0.7

The unbounded variable used as the probability of the task t missing a deadline of 12:

P(t(i), t(i).resp > 12) > Y

A major advantage of the PPL over other tools with just reachability functions like Uppaal is that apart from giving a yes or no answer, it also makes available information in terms of probability of the desired reachability. On the experiments carried out in this thesis, PPL was used.

6 Validation in computational engineering and physics

As has been precisely pointed out above, validation cuts across disciplines. In this section we shall investigate validation criteria and methods in computational engineering and physics with regards to complex systems.

Note that the mention of computational physics here comprises fields of computational engineering and physics such as computational fluid dynamics, computational solid mechanics, structural dynamics, shock wave physics, and computational chemistry.

6.1 Validation in Computational Physics

To a large extent, there has been an increasing reliance on computer simulation of physical processes for design, performance, reliability and safety of engineered systems. Verification and validation of computational simulations are the primary methods of building users' confidence in such models.

While verification is the assessment of the accuracy of the solution to a computational model, validation is seen as the assessment of the accuracy of a computational simulation by comparing it with data from experiments. In verification, the major issue is **no**t the relationship of the *simulation* to the real world, whereas in validation it is the core issue.

Implications and salient factors with respect to these definitions would include the fact that both verification and validation are 'processes of determining' [26]. These could be described as on-going activities, whose adequacy or completion is normally determined by practical issues like budget, and the models' intended use.

Also the above stated definitions have to do with an on-going pattern of the processes due to the unavoidable and distressing fact that veracity, correctness and accuracy of a computerized, or computational model *cannot* be demonstrated for all possible conditions and applications except for *trivial* models. All encompassing proofs of correctness such as those developed in mathematical analysis and logic; do not exist in complex modelling and simulation.

Actually, just as one cannot prove that computer codes that are complex have errors, so also models of physics cannot be proven correct; they can only be disproved. Therefore verification and validation activities can only assess the correctness or accuracy of the specific cases that are tested.

Due to the emphasis laid on accuracy, it is assumed that a measure of correctness can be determined. In validation activities, accuracy is measured in relation to experimental data, which is the best assurance of reality. Based on the fact that all experimental data have random (statistical) and bias (systematic) errors, the issue of 'correctness' in an absolute sense therefore becomes impossible.

Hence from the point of view of engineering, 'absolute truth' is not required, but an expectation of a meaningful statistical comparison of computational results and experimental measurements.

As popularly pointed out by Roach [27] 'verification deals with mathematics; validation deals with physics'. Hence, validation handles issues of fidelity of the model to the specific conditions of the real world.

In computational physics, the conceptual model is dominated by Partial Differential Equations for mass conservation, momentum and energy. It also includes the auxiliary equations like turbulence models and constitutive models for materials as well as the initial conditions and boundary conditions of the Partial Differential Equations. The computerized model is an operational computer program that implements the conceptual model.

6.2 Methodologies:

According to R. Rebba et al [28], validation under uncertainty involves quantifying the error in the model prediction and effectively comparing the prediction with the results of the experiment when both prediction and test data are stochastic.
The fundamental strategy or method of validation here deals with:

- Identifying and quantifying the error and uncertainty in the conceptual and computational models.
- Quantifying the numerical error in the computational solution
- Making an estimate of the experimental uncertainty
- A comparison of the computational results with the experimental data

The above strategy insists that measurements in experiments are most trust-worthy reflections of the real system in terms of validation. Thus this estimation process for error and uncertainty should happen on both the mathematical physics and experiment.
With specific references to complex or larger systems, most times, a building-block approach is adopted in Computational Physics and Engineering due to the infeasibility and impracticality of carrying out true validation experiments on such systems [29, 30, 31, 32, 33].
In this approach, one divides the complex engineering system into a minimum of three tiers namely the *subsystem cases, the unit problems* and *the benchmark cases*. The aim of this division approach is to know how accurately the computational results compare with the experimental data with the quantified uncertainty estimates at different degrees of physics coupling and geometric complexity. Therefore one can see the above approach as being constructive in the sense that it recognizes the existence of hierarchy in complex systems and simulations and also recognizes that the amount and accuracy of information, which are obtained through experiments, could differ over the range of tiers.
The above description also portrays the fact that validation activities could be done at several different levels of Physics and system complexity, where each such comparisons yields inferences of validation of the tiers both above and below the tier where the comparison is made. It should be noted that the quality of the inference made at this point is dependent on how complex the tiers are below and above the comparison tier, which could be seen as a direct reflection of the strength of one's scientific knowledge about the experiments and calculations being made relative to more complex tiers.

This same problem of taking the whole system at a time (impracticability and infeasibility) is also applicable to computer science. The approach of validation in tiers is comparable to the concept of testing of computer systems where a large complex system could be divided into component, units and sub-units for testing. In this case, the test results of each of the divisions could be used to make reasonable inferences concerning the whole system.
However, this is as it concerns testing. Validation of models in Computer Science still considers modelling of the whole system most times and validating all at once instead of modelling sub-divisions of such systems and predicting overall system behaviours from them. Thus one can 'borrow' this practise from Computational Physics to sub-divide target complex systems into tiers, model each tier relative to the tiers above and below and then model the interaction between the tiers in order to make intelligent and informed decisions. This process would need that one possesses a good knowledge of the system and how it functions especially as it concerns temporal correctness in real-time systems.

With regards to validation adequacy in computational physics, two issues are important:

1. Identifying the metrics (measure) of validation
2. A specification of the magnitude of the metrics to satisfy the application requirements.

The two (the metrics and criteria for success and failure should be made clear in order to assess the expected adequacy of the outcomes of experimental-computational comparisons. The choice of one or more metrics defines the means used to measure the disparity in computational results and experimental data.
A metric should quantify both errors and uncertainties in the comparisons. The requirements for the adequacy of all specified metrics are tied to the total application requirements therefore the process of specification of the metrics for validation will work hand-in-hand with the process of specifying the application requirements.

In comparison with practises in Computer Science, choices of initial validation metrics and their magnitude specifications are obtainable. But in terms of error quantification and uncertainties, it becomes excessively difficult due to the discontinuous nature of software codes. This is because it is not easy to attribute error quantities to software codes, which should either be wrong or right. Therefore, since exhaustive testing of codes of computer systems especially for large complex systems could mean testing for several years, one cannot absolutely make certainty statements (or give safety margins) as to the percentage error inherent in a software code especially for non-tested case instances.
Potentials of systems' design with high testability in computer science, which is the probability of failures to be observed during testing when errors are present could produce more reliable systems, {34} are similar to that discussed here and thus could be applied to modelling too but not absolutely. Even while using the PIE model (determining when a bug or error causes a failure), to find the smallest error inherent in the code, which happens if and only if the location of the error is executed in the program, the execution of the error leads to an erroneous state and the erroneous state is propagated to the output; one can only determine the minimum number of test cases to find the smallest error for a certain confidence level and not the maximum and so would only yield probable instances in models as well. Moreover, testing explores the real system and not an abstraction.

7 Experiments

In this section we shall discuss the major **motivation** for our work. We will report our experiments, observations, comparison our validation process and technique and make useful conclusions on the created model's validity.
Large and complex real-time (both distributed and non-distributed) computer systems always evolve as time goes on.
During their evolving period, it is likely that increasing the functionality and hence maintaining the system becomes absolutely necessary. In cases where models of the original system have been made, it will be noticed that the temporal model of the system becomes inconsistent with the system's current implementation, in which case, one looses the possibility to analyze the effect created by the added features as they relate to the temporal behavior.
Obviously, this may not be an issue for small systems but for complex and large systems, the resulting consequences may not be foreseen immediately. It becomes important therefore not only to update existing models of the system but also to adopt feasible, flexible and realistic analyses methods that could readily validate such models.
In our experiments, the major tasks were to create a system, and to make an abstraction of the system (a model) based on some criteria. We then simulated and validated the resulting model with respect to the criteria that have been specified using the PPL.

We used the VxWorks, which is a real-time operating system and then evaluated the correctness of the model by performing a sensitivity analysis.

7.2 Modelling on different levels of abstractions

During model creation, a level of abstraction (defining and selecting relevant parameters with which to validate against) is normally chosen. It is this level of abstraction that defines, after comparisons, the accuracy and correctness of the model.
A very high level of abstraction, which entails choosing too few out of the essential properties or characteristics of the target system could result in a model not being accurate hence being of limited use in its domain of application. On the other hand, choosing a very low level of abstraction would literally mean recreating the target system, in which case there would be no need for the model.
Therefore, a good situation would be a choice in between these two levels that properly characterizes the essential properties of the system from which analysis and possible predictions could be made. Hence the model should be as simple as possible and yet a good description of the system.

7.3 System description and overview

The system we created is a control system, which is as shown in the Figure 8.

It controls an electric motor based on the data from a sensor. There are three critical tasks in the system namely, the *sensor task*, the *control task* and the *drive task*, depending on data sent through message buffers. The buffers are as small as possible so as to avoid using old sensor data, which could occur if a large sized buffer were used.

The sensor task:

- a time critical task, time triggered taking readings from the sensor every 2ms,
- it stores the value of its reading in the *message buffer* called CTRL Data Queue,

- when the buffer is full (that is, contains the maximum data readings), the oldest data is removed and,
- it has the highest priority (1).

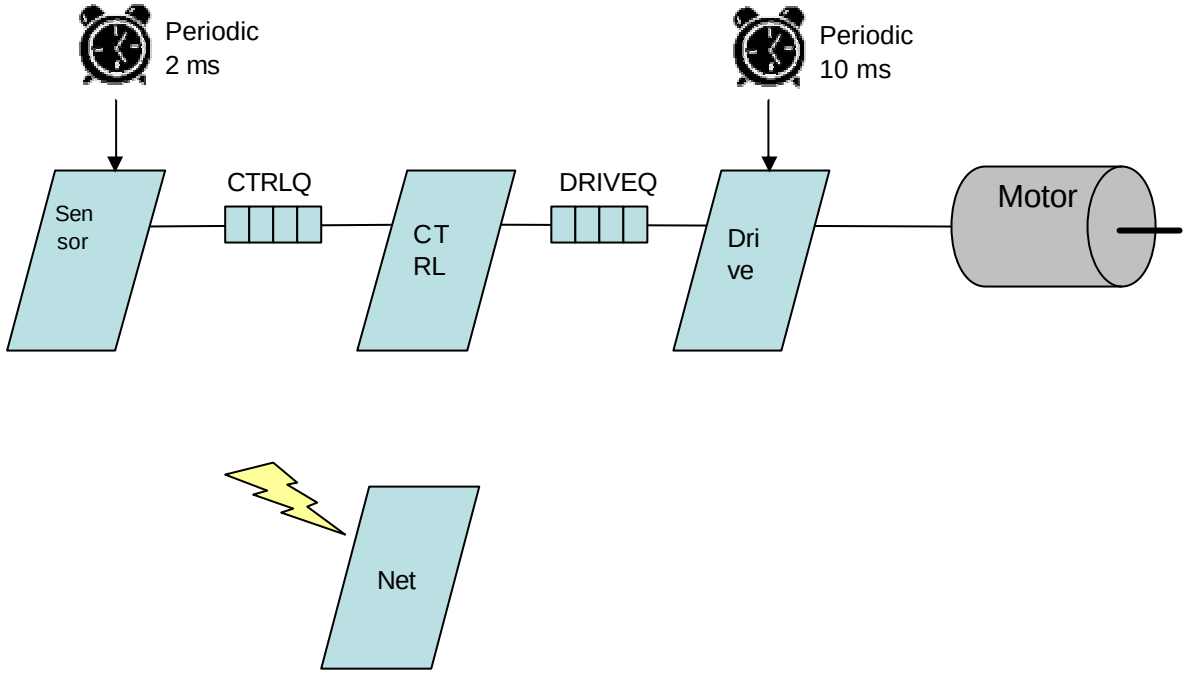

Figure 8: System overview

The CTRL task:

- Only executed when the number of sensor readings produced and stored in the message buffer is five.
- If 5 data entries are available, the CTRL consumes these five readings and calculates a motor reference, which it sends to the *Drive Queue* message buffer.
- Has a low priority of 4.

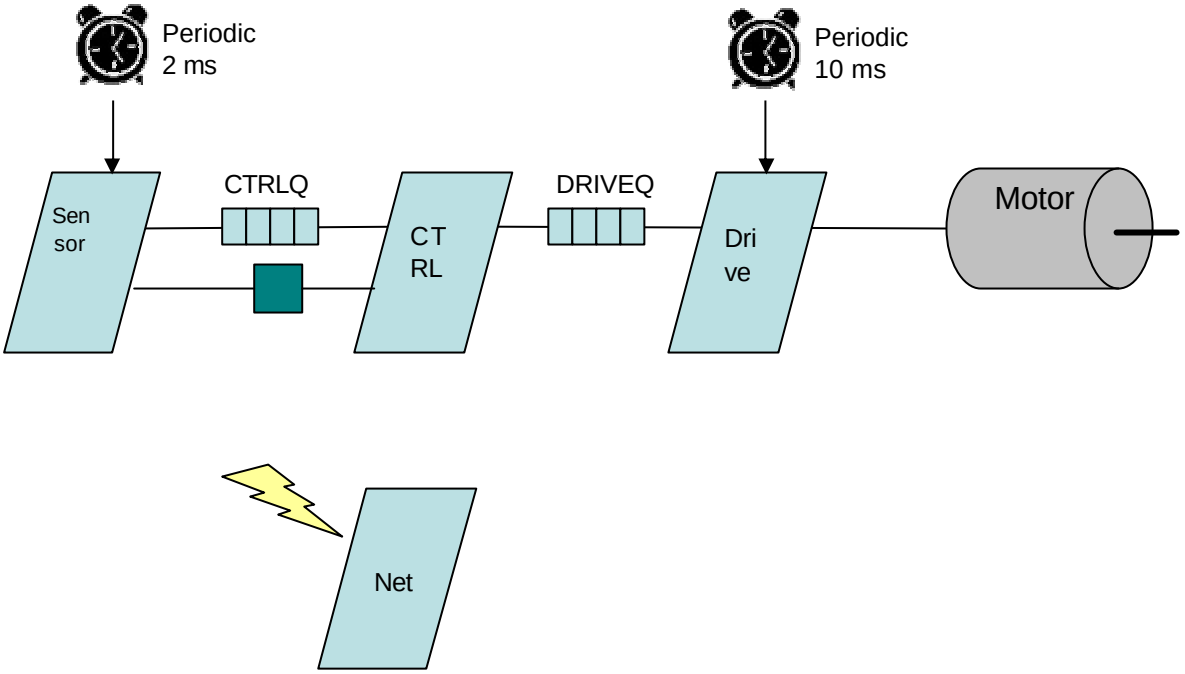

Figure 9: system with the NET task

The DRIVE task:

- Carries out periodic executions and sends data to the *motor control* every 10^{th} ms (since each reading is taking periodically every 2ms).
- Reads entries from the Drive Queue and based on that, it updates the signals to the engine control electronics.
- It has a priority of 3.
- If the Drive Queue is empty when the drive task wants to read data, it is a system failure.

In the system, there are some less critical tasks, which also communicate with the CTRL task, but through the message buffer called CTRLCMDQ.
When such messages are processed, the motor references calculation is delayed. This would likely not be a problem in the sense that such messages in essence only delays the CTRL for a relatively short time and there are enough buffered motor references in the DRIVEQ so as to avoid it being empty (starvation). At this point the CTRL is BUSY.

As can be seen from the diagrams above, two versions of the system were considered in our experiment, the first being the platform, which consists of the SENSOR, CTRL and DRIVE tasks as shown in the system overview diagram, Figure 8 above.
In the second version, a NET task was added. The NET task is used to denote ‘environmental’ variations that could form part of the input to our system. These may not directly be under specific periodic control as shown in the system overview diagram, Figure 9.

The NET task:

- Reacts to commands from the network
- Has a priority of 2
- Inter-arrival time is stochastic (between 5 – 25) ms

In order to analyze the temporal behavior of the two versions of the system, the Probabilistic property language (PPL) is used. For complex real-time systems, traditional methods used in the basic real-time systems' analyses could prove unrealistic in the sense that lots of assumptions and stringent constraints are built into them, that may not fit into real life situations. Hence tools like Uppaal with limitations as have been discussed above, may not give the best results.
However, PPL gives a probabilistic outlook. Even though it is no *silver bullet,* it gives good realistic information about the problem as against some of its contemporaries, especially with regards to the state space explosion problems, which could occur in complex systems.

A technique that is employed in the field of control theory is system identification. From measurements and observations of the relationship between the input and output of the signals in a process, one can determine a model in terms of a transfer function.
Outputs that are produced by a simulation of the model is considered to be correct if the physical processes and the simulations produce the same output data.
On models of continuous systems, one can apply residual methods, which is an observation of whether or not the residual (prediction errors) and the input signal are independent. If it happens that the errors depend on the input, it indicates that there are dynamics in the systems that are not in the model. Therefore, testing the model with different input signals and comparing the prediction with the signals produced by the actual system is good if the process is a continuous one.
However, computer systems are discontinuous in nature, which implies that the pattern of behavior could change dramatically as a result of small changes in the system.
The approach used in our experiment for validation is close to that used in system identification. The idea is that irregularities would be detected if we introduce changes in the system and corresponding changes in the model. Therefore when we compare the simulation results with the data that has been measured in the system, we would be able to ascertain the models' validity. We describe the different steps in the model creation below.

7.4 System model creation

In creating our initial model, we carried out the following activities as shown in Figure10 below:

1. Structural modelling: Here we identified and modelled the structure of the target system. This has to do with modelling the tasks, synchronization, communication and interactions among them in the system.
2. System measurement and population:
We measure the system and populate the structural model with data about the temporal behavior. Also information that would be required for the phase of validation is collected at this point, for instance the response times.
3. Tuning of the model:
In this step, we first of all simulate the initially created model with the aim of comparing them with the validation data that were collected in step 2 above. We watched to see whether the target systems' behavior has been well represented by the model accurately and if not, introduce more relevant details into the model.

A major risk that is inherent in this step is the temptation to introduce too many fine-details into the model, which only reflects the complexity of the system in question.

4. Sensitivity analysis:

In order to validate our model, we carried out sensitivity analysis. This is normally based on the foreseen potential changes in the system. When sensitivity analysis was carried out on our system, we observed:

- A change in the existing behavior of the tasks
- We added the net task to see what effect this would have
- We changed the priority of the NET task so that it has a higher priority than the CTRL task.

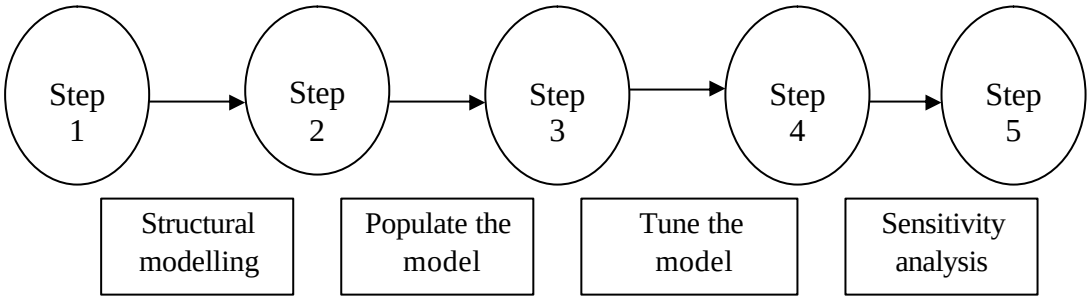

Figure 10: Process of model construction

At this point, it is important to note that the model accuracy relies on the quality of the measured data. The data measurement should affect the system as little as possible. If the probe effect on the system is too big, one could have an erroneous model, which consequently would lead to wrong predictions from the model results.

Comparison of the target system with the behavior of the model is done in a simplified manner so as to improve the model iteratively to a desired level [34]. See Figure 11 below.

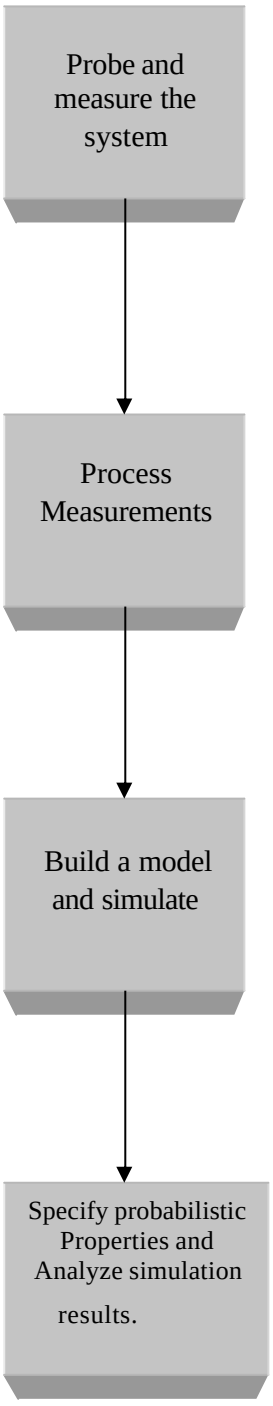

Figure 11: workflow of making an analytical model

7.5 Metrics for validation

Validation metrics could be viewed as yardsticks for comparison between measurements taken on the real system and those run by simulation.
For our system, several metrics could be used. These include:

- response time,
- scheduling properties on inter-arrival time: This deals with how the inter-arrival times differ between the tasks,
- shared resources: How resources are shared among the tasks - whether independently or involving semaphores,
- message buffers: Deals with instances of average numbers of messages in the message queue, for instance, their minimum or maximum numbers,
- communication: Here we consider instances of tasks sending and receiving messages; for instance, if one sends, does the other receive?
- execution times of the tasks: How long the tasks take to execute.

However, in the experiment, two of the above metrics were considered namely, the tasks' *execution times* and *response times*. The Figure 12 below shows the system attributes as described.

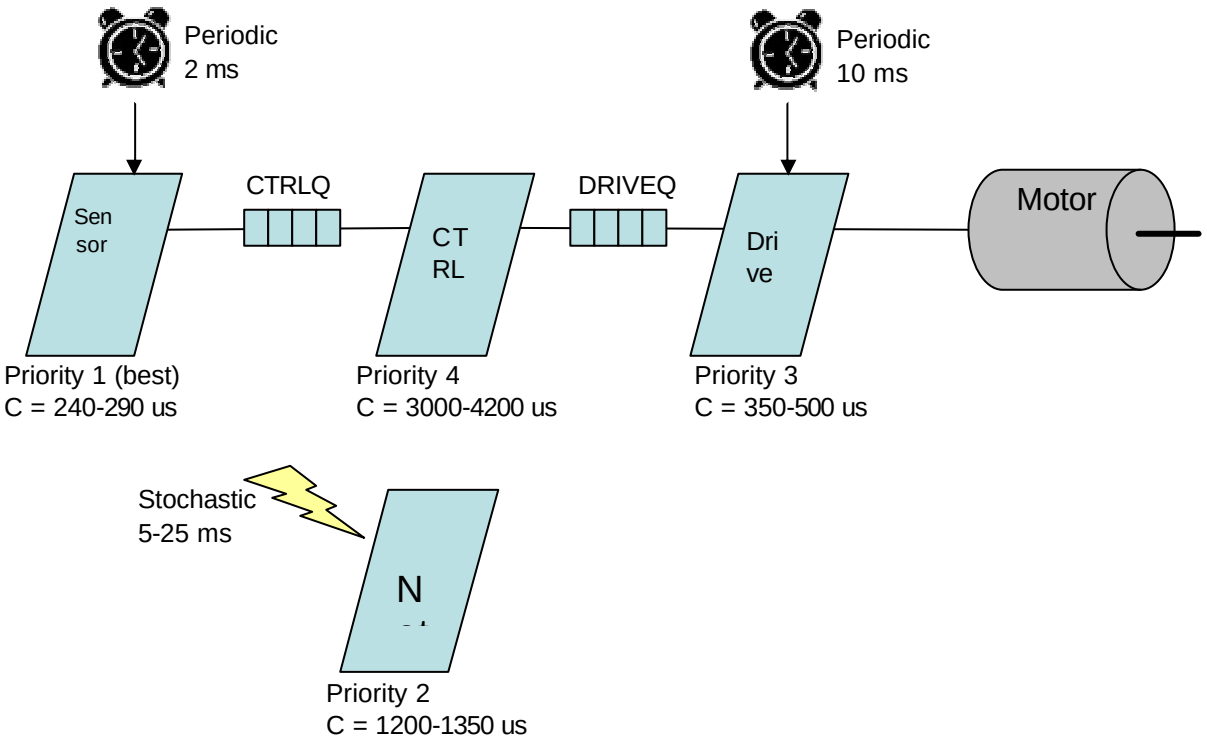

Figure 12: System's attributes

7.6 Methodologies and Systems' Measurement

Our target system runs VxWorks real-time operating system; an Intel Pentium 200MHZ from ABB Robotics. Timing was measured using a probe that is executed on each task switch. The probe was not removed but was left in the code. Constraints that could be obtained in inserting the probe include memory size and probe effect (slower running of the system, which could give an effect with negative consequences).

Test-cases:
There were two test cases in our experiment. These are:

1 Case 1 – 1: For this test case, there is no NET task. We have only the sensor, the control and the drive tasks.
2 Case 3 – 1: In this case, we introduced the NET tasks, which arrived stochastically, 5 to 25 milliseconds between instances.

In the first case, the sensor task's measured execution time was 240 – 290 microseconds. The CTRL task measured execution time was 3000 – 4200 microseconds, while that of the DRIVE task was 350 – 500 microseconds.
On the second case, the execution times of the three tasks were maintained and that of the NET task was 1200 – 1350, with an inter-arrival time of between 5 and 25 milliseconds.

7.7 Results

In showing and discussing the results, we shall first consider the target systems' measurements and then those of the model from the simulator.
A good scenario for the comparison between our model and the target system occurred with the control task as shall be shown later.

Case 1 -1
The Sensor task: Part of the measured data is given in the Table 1 in Appendix B1 that shows the measurements with respect to the execution and response times' metrics. Each plotted dot represents an instance of execution.
A plotted Excel chart of the measured values is as shown in Figure 13.

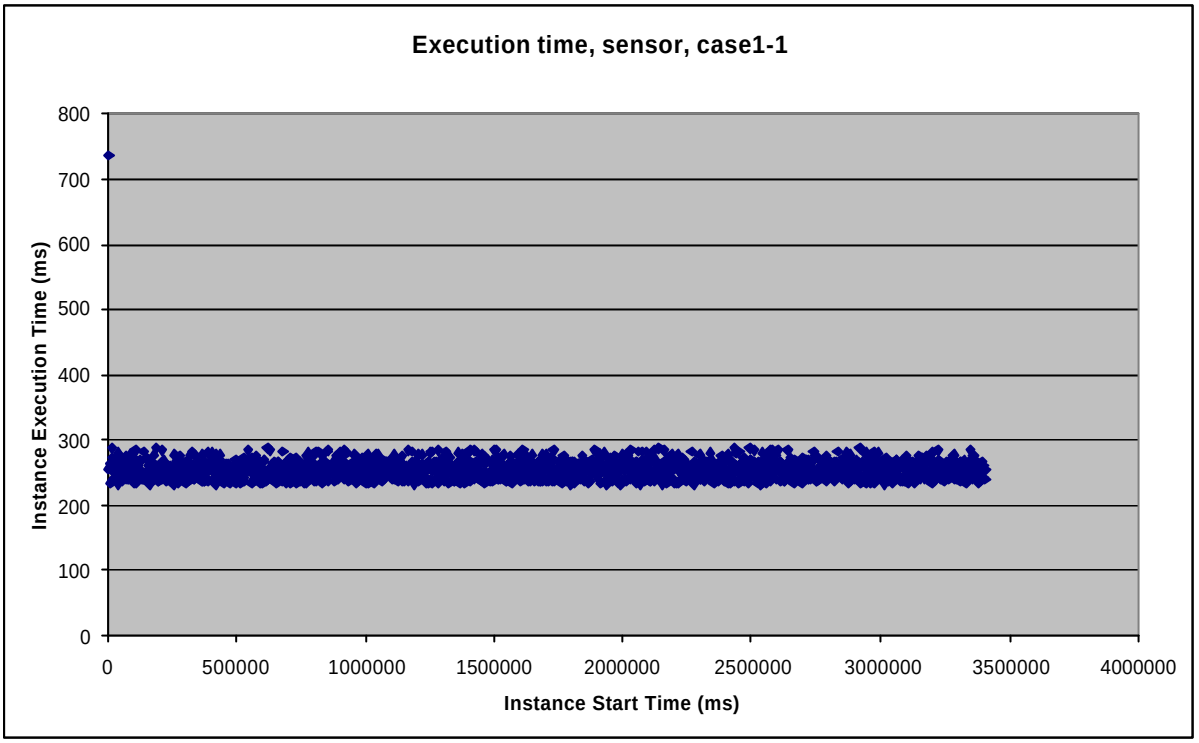

Figure 13: Execution time plot for case 1 -1 (sensor task)

Also, an Excel chart plot of the above measured response time is shown below in Figure 14:

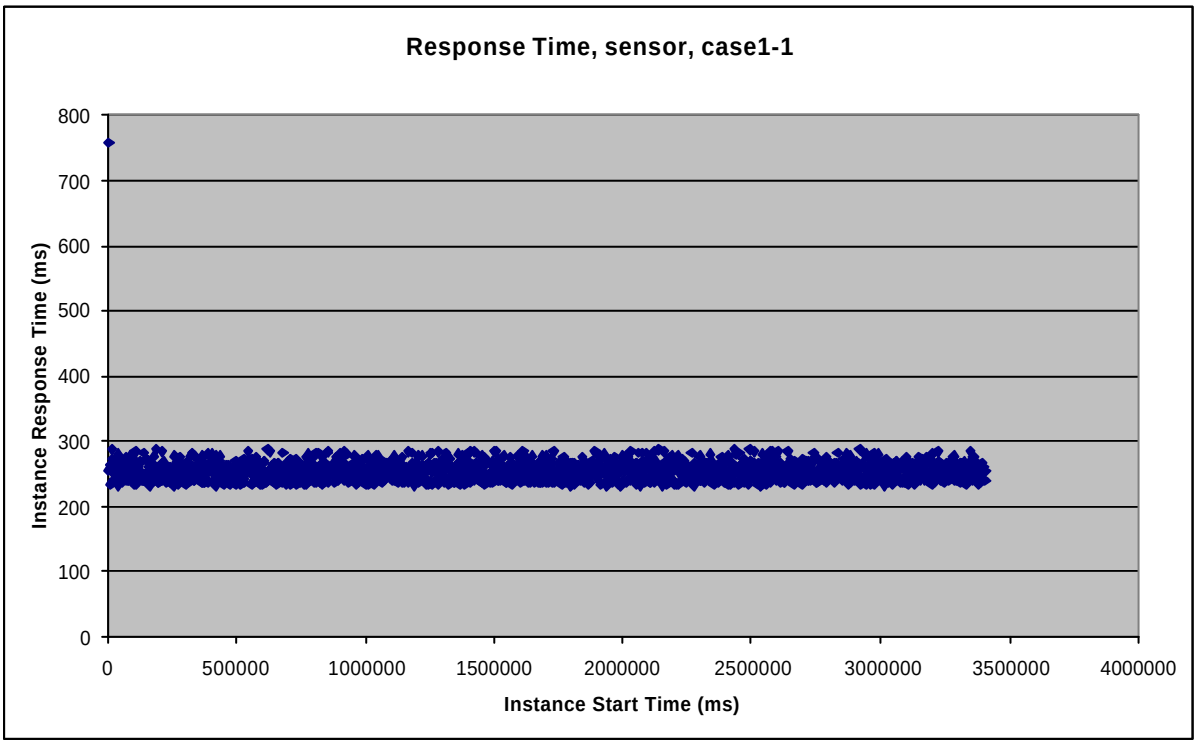

Figure 14: Excel plotted response time for case 1–1 (sensor task)

Case 3 -1 (with the NET task added)

The control task:
A cross section of the measurements for the control task's execution and response times are shown in the Table 2 in Appendix B2.

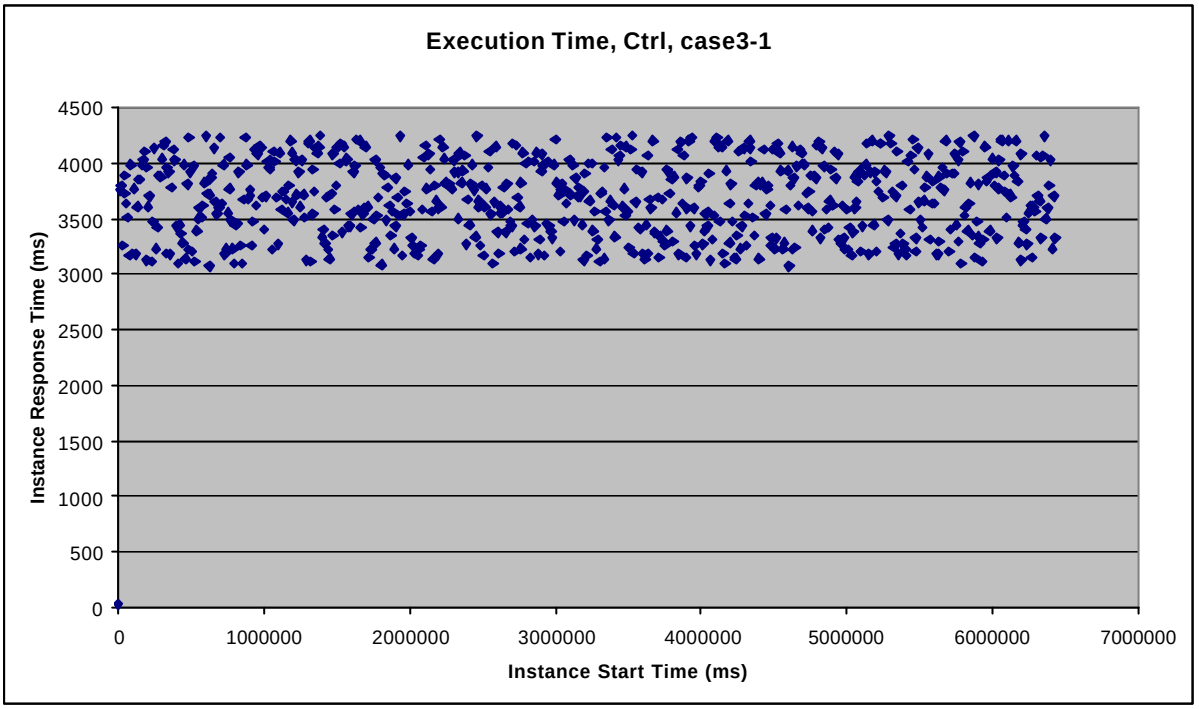

Figure 15: Execution time plot for case 3 -1 (control task)

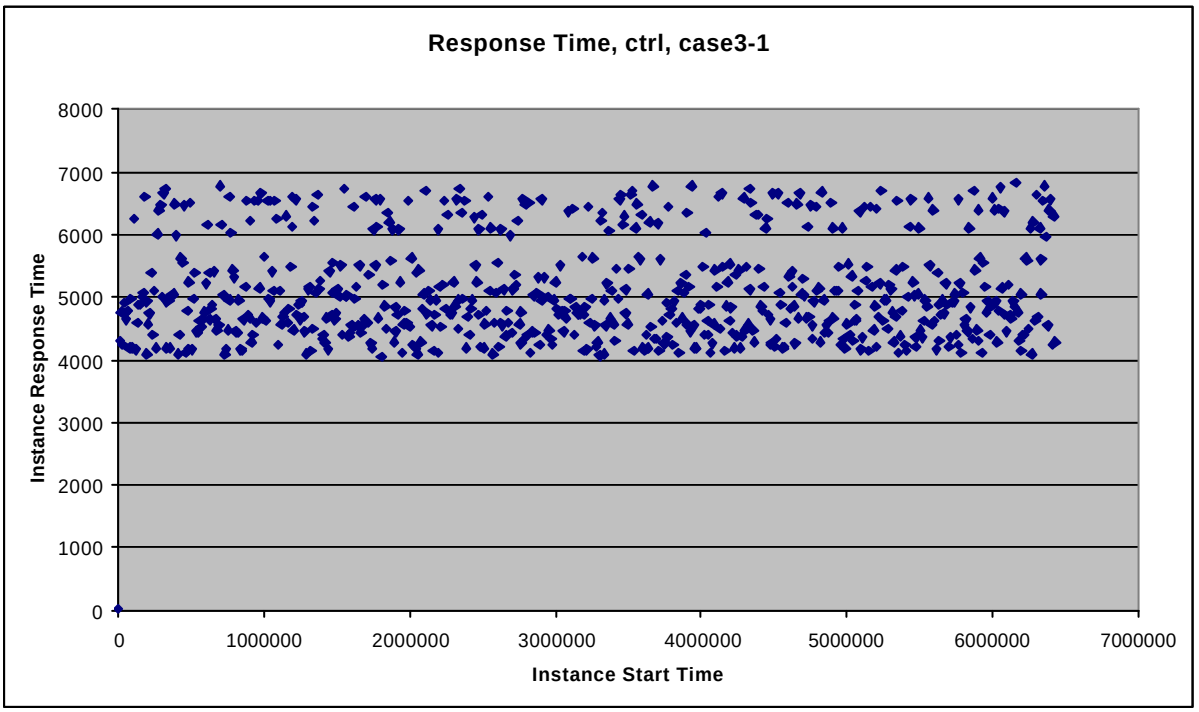

Figure 16: Response time plot for case 3 -1 (control task)

7. 7.1 Simulation models results and plots

Now we shall consider the results and data from our simulations.

Case 1-1

Sensor task: From the simulation results, a cross section of the measurement is as shown in the table 3 in the appendix B3 below.

A plotted Excel chat of the above table for the simulated execution and response times are as shown below in Figure 17.

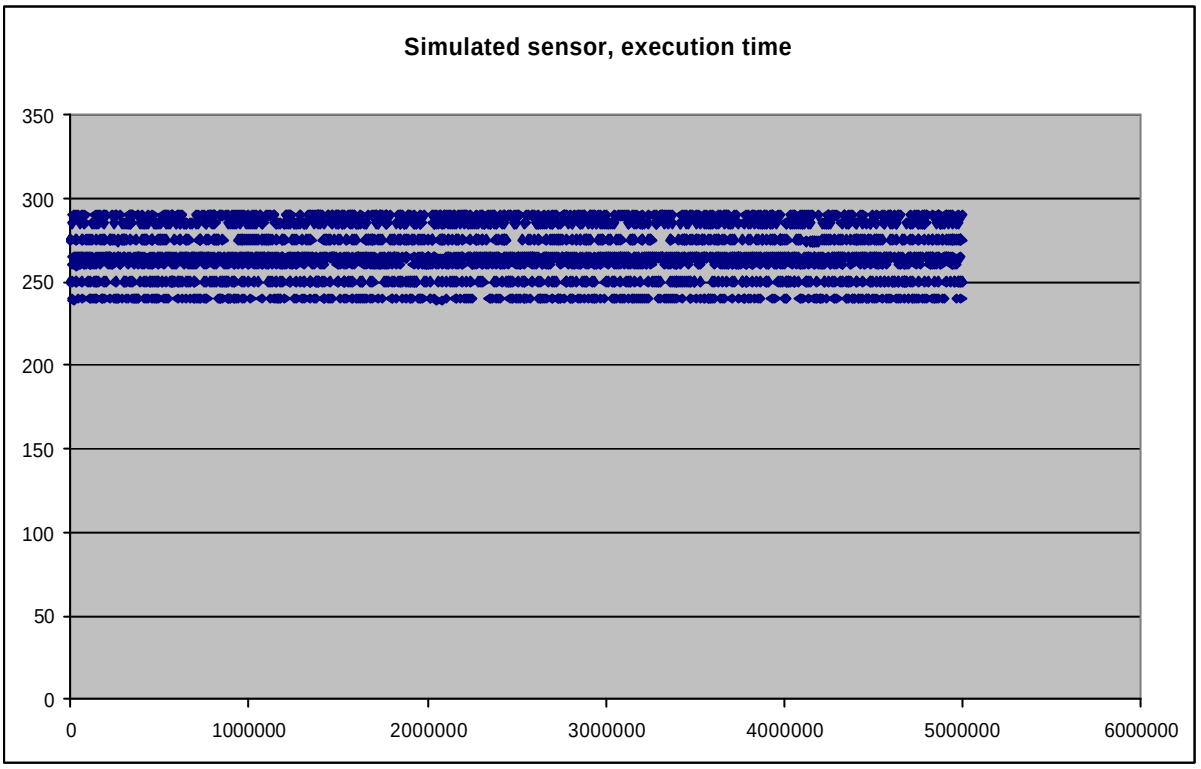

Figure 17: Excel plotted Simulation execution time for sensor task

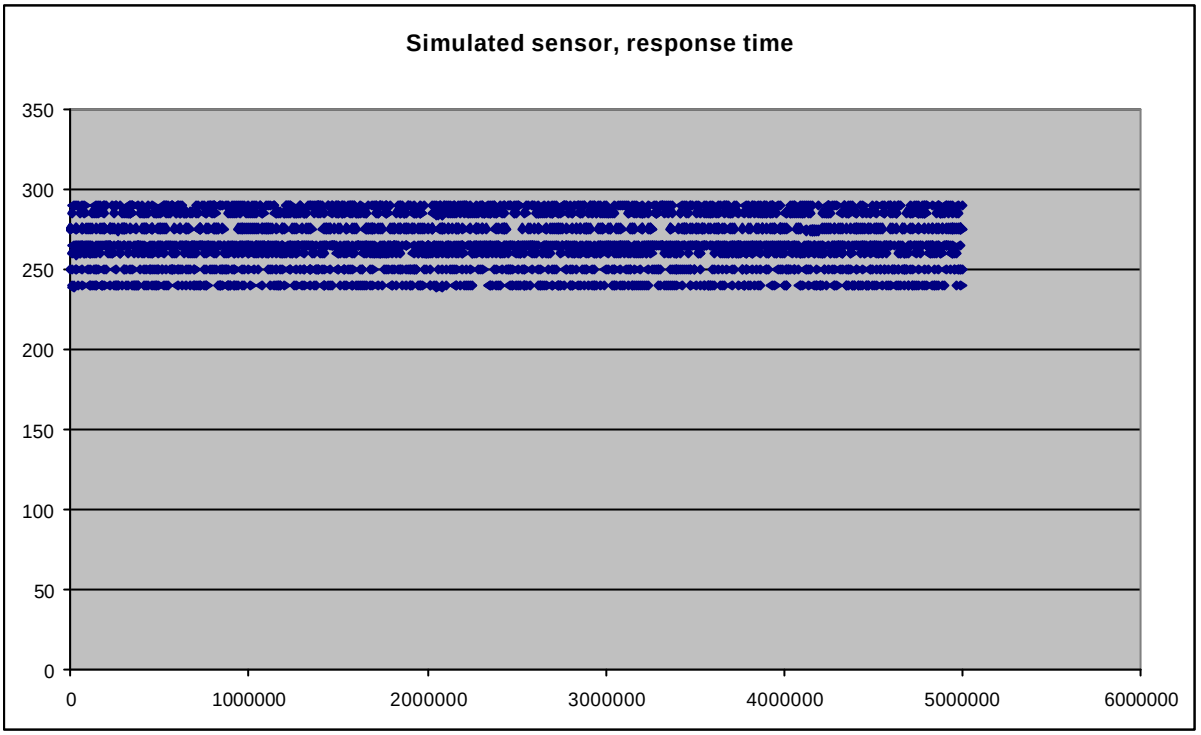

Figure 18: *Excel plotted simulated response time for sensor task*

7.7.2 Simulation measurements for Case 3-1

A cross section of our measured data for the execution and response times of the CTRL task from the simulator for case 3-1 is as shown in table 4 in Appendix B4.

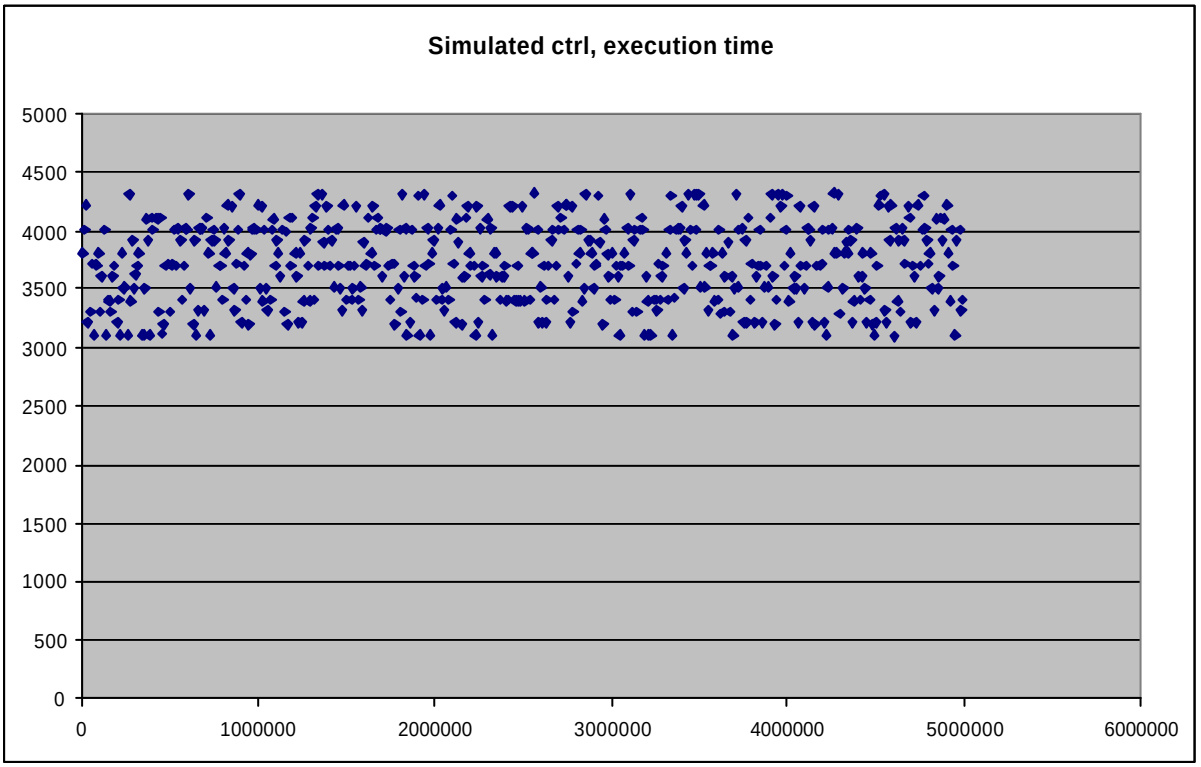

Figure 19: Simulated plots of the CTRL task execution time for case3-1

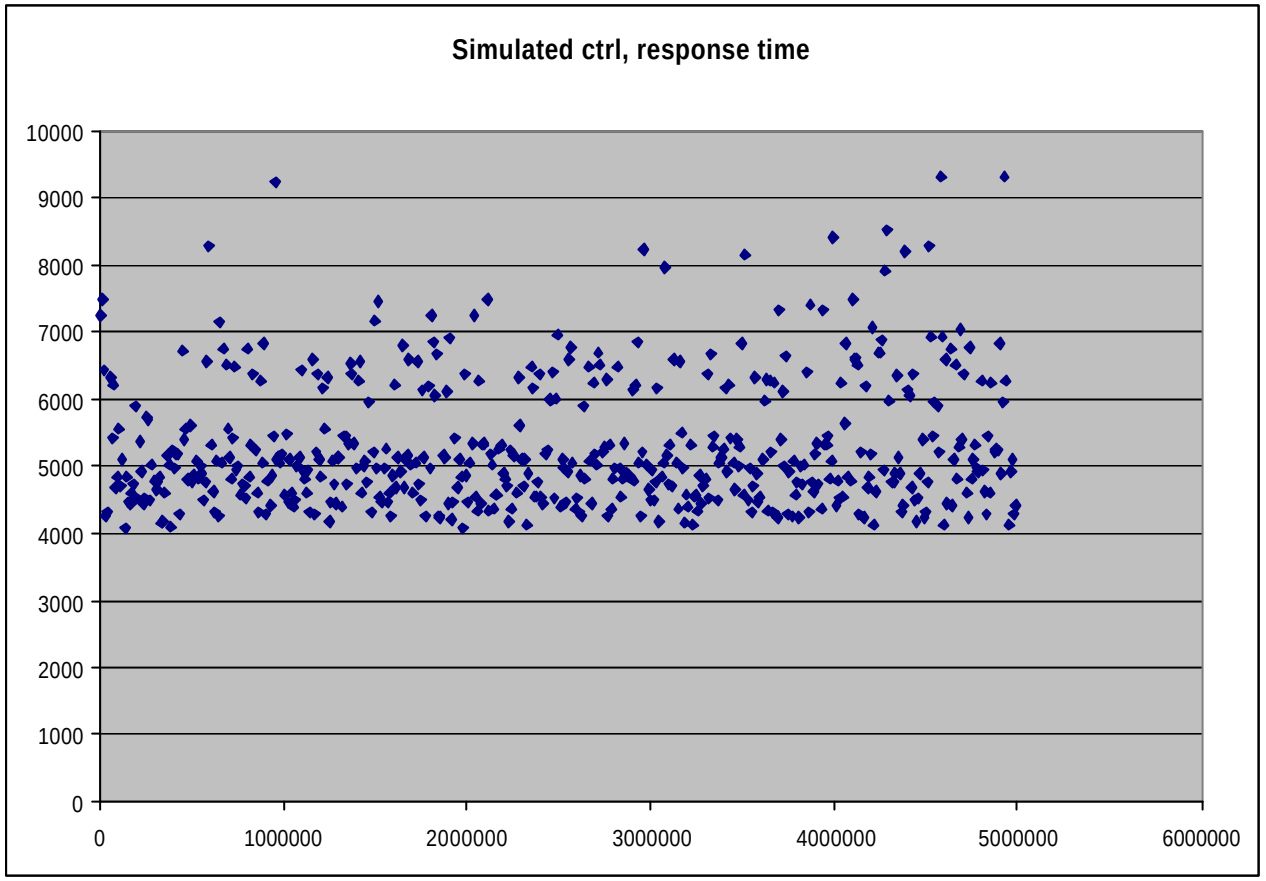

Figure 20: Simulated plots of the CTRL task response time for case3-1

7.8 Analyses and validation

For the analyses, a trace analyser and a PPL 'expert-rules' system were used.
According to what is allowed by PPL, one can specify probabilistic maximum and minimum levels of acceptance based on *apriori* knowledge of the system. However, we did not specify such maximum nor minimum values.
As can be seen in the chart in figure 8k, we specified PPL queries based on our chosen criteria, namely the response times and execution times of the tasks. Details of further PPL queries are displayed in Appendix A.

Observations and results:

- It is our observation that the number of queries used could be proportional to the level of confidence gained in the model. This is because, as the number of queries increases, so also there is an increase in the depth of the coverage of the criteria being queried.

- The number of queries introduced into the analyser determines to a large extent how much information that were obtained from the system and hence offered greater depth of comparison.

- The *exact* number of queries needed to extensively investigate a criterion could be dependent on the complexity of the system being investigated. This is to say that if one has a system of low complexity; it would take less number of queries to exhaust a criterion than it would in a highly complex system.

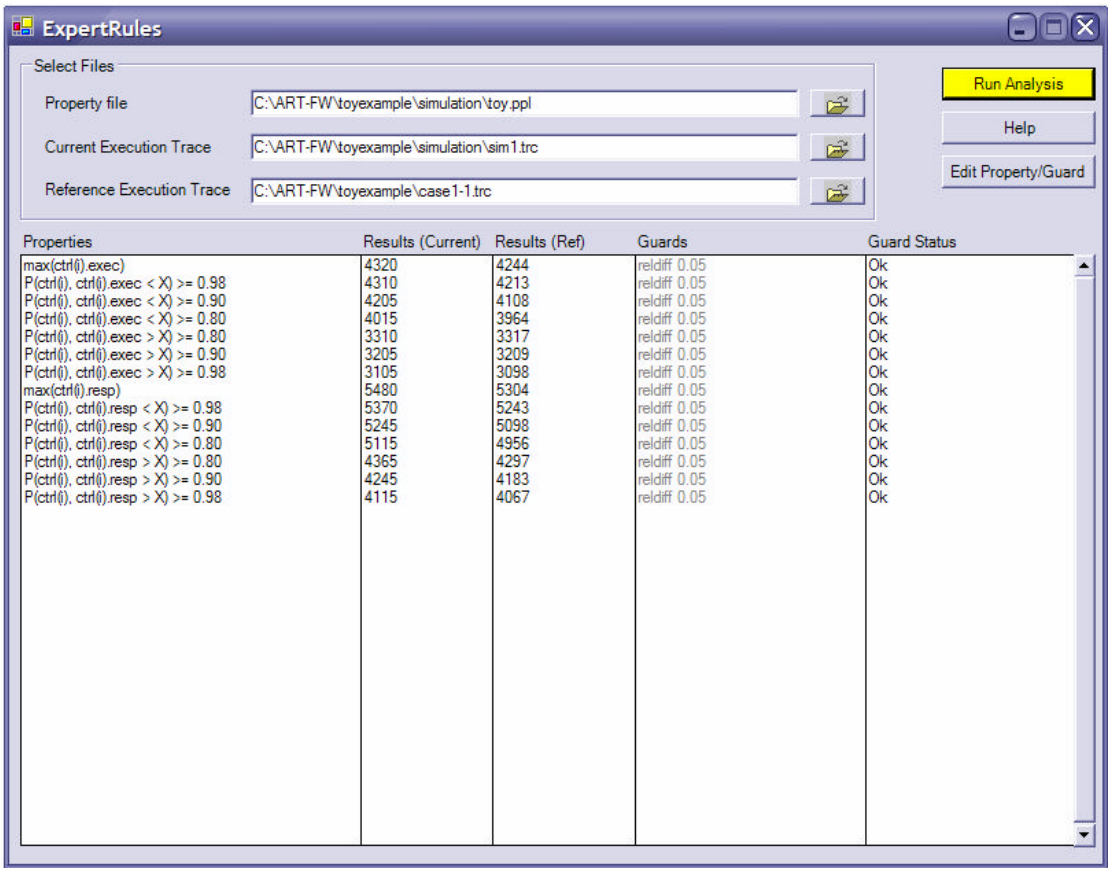

Figure 21: A snapshot of the 'expertRules' graphical user interface

Hence to boost the level of confidence in a model, the number of queries introduced into investigating a criterion is a key factor. The more relevant queries that are introduced, the more the level of confidence rise.

On comparing our target system's measurements to those of our model, for case 1-1 we observed that the model gave a good representation of the target system. In Appendix A below, we show some examples of the reports compiled and used for the comparisons using the Expertrules Tracer.
However, in case 3-1, when the NET task was introduced into the system, the simulated results from the '*Expertrules*' analyser showed that two instances of failure occurred, which means that there were major deviations between our model and the target system's values. When these values were plotted as shown in figures 19 and 20, they produced excessively random values inconsistent with those in figures 15 and 16.

Reason: This was because we had *stochastically* chosen the inter-arrival times of the NET tasks, which were not appropriately represented in our model, hence the occurrence of this variation. This would have been different if the missed detail in abstraction level was rightly considered while creating our model. This presented us with a typical instance of an invalid model.

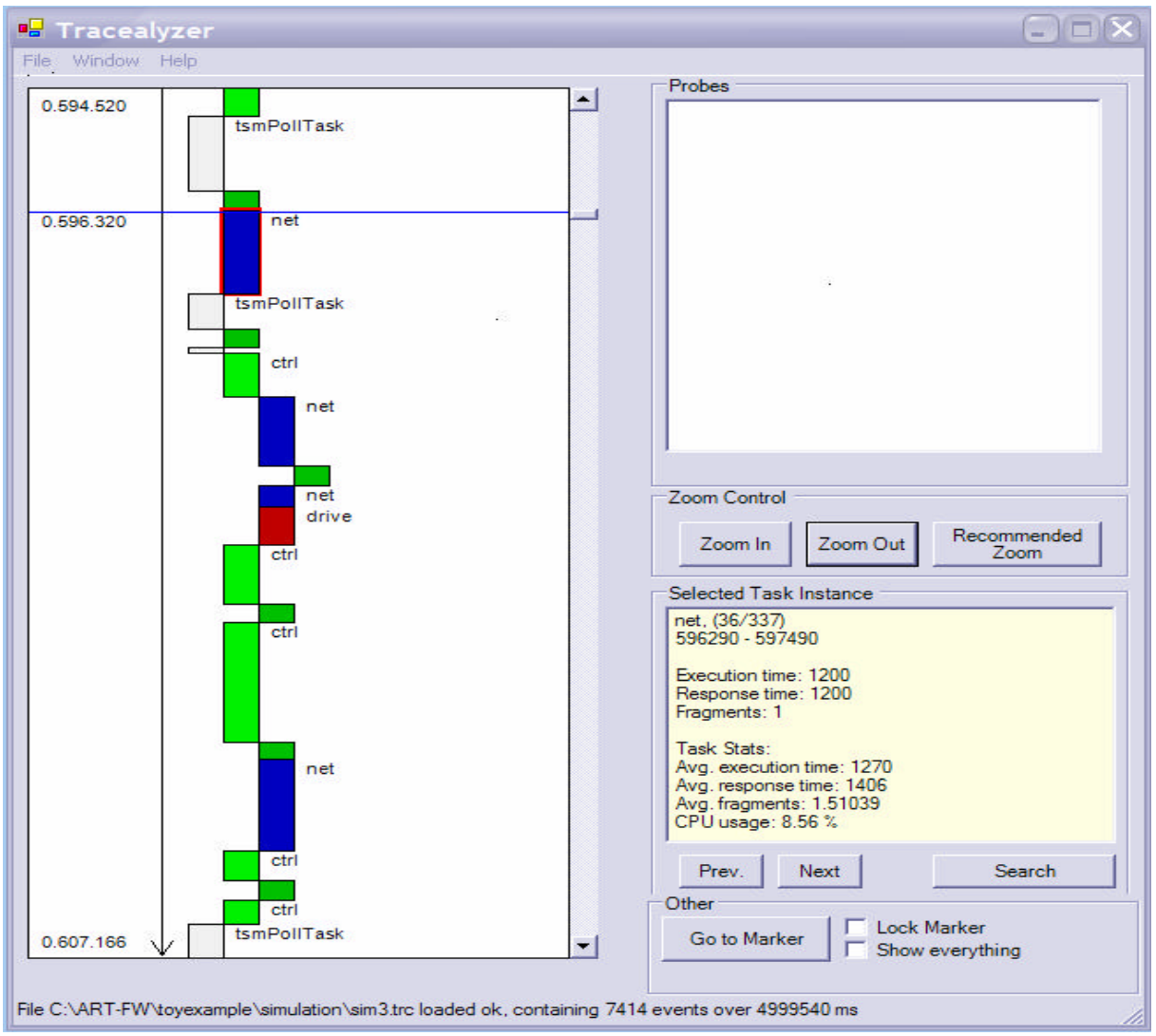

Figure 22: A snapshot of the trace analyser

8 Proposed guidelines and recommendations

From our investigations and experiments, we present two useful proposals that would help engineers and empirical scientists who work on validation of models. These proposals are explained and demonstrated below.

- **Proposal 1**: *The validation **criteria** probes into the **functionality** of the system*

  Consider a scenario where one has a system, named system 1, with n number of tasks and n random tasks, which could denote some environmental variation input source (optional) – random in the sense that the inter-arrival times of this task may or may not exactly be known.
  The above proposal suggests that if we have a second system, system 2, whose number of tasks is greater than those of the tasks in system 1, but both systems have the same functionality, then the *same number of criteria* as have been used for system 1,could be used to adequately probe system 2 to the same level of confidence that has been attained by system 1.
  Thus,

  Let number of tasks in system 1= n
  and Number of tasks in system 2 = N
  Such that: number of tasks in system 2 / tasks in system 1 = P (a whole number)

  Let the total number of validation criteria for system 1 = V1
  and the total number validation criteria for system 2 = V2

  Also let the attained confidence level for system 1 = C1
  And the desired confidence level for system 2 = C2

  Our proposal 1 implies that:

  Iff C1 = C2
  and
  N / n = P,

  Then, VI = V2.

- **Proposal 2**: *The validation **queries** probe into the **complexity** of the system*

  Still consider the above given scenario of a system, system 3 with n number of tasks and n random (optional) tasks, which are greater than those of system 1.
  Proposal two suggests that if there exists a system 3, which differs from system 1 in terms of *complexity* and not *functionality*, that is, if the functionality of both systems remains the same, then one can analyse and validate the model of system 3using the *same* number of criteria as were used in system 1, but with a total number of queries equal to the multiple of the magnitude of the ratio of the tasks in system 3 to those in system 1.
  Thus,

  Let number of tasks in system 1= n
  and Number of tasks in system 3 = N

  Such that: number of tasks in system 3 / tasks in system 1 = P (a whole number)

Let the total number of validation criteria for system 1 = V1
and the total number validation criteria for system 3 = V3

Also let the attained confidence level of system 1 = C1
and desired confidence level of system 3 = C3

If the total number of queries used for system 1 = Q1
And the required total number of queries for system 3 = Q3

Then, proposal 2 implies that:

Iff C1 = C3
and
N / n = P,

Then, VI = V3 and:

Q3 = PQ1

Basic assumptions on proposal 1 and 2:

- All tasks access the message queues by way of communication
- Communication here implies reading from or reading to a message queue and not necessarily sharing the CPU as in interleaved tasks.
- The systems have the same functionalities but different complexities.

It is very important to note that *complexity increases with functionality but functionality does not necessarily increase with complexity.* This can be better understood by looking at complexity as a matter of design and implementation and not of functionality.
Therefore a simple hot-water heating system produced by two different companies could have different levels of complexities but obviously have the same functionality. Consider the design of a simple swing example as shown in Figures 23 & 24 below. Both have the same functionalities but different design complexities.

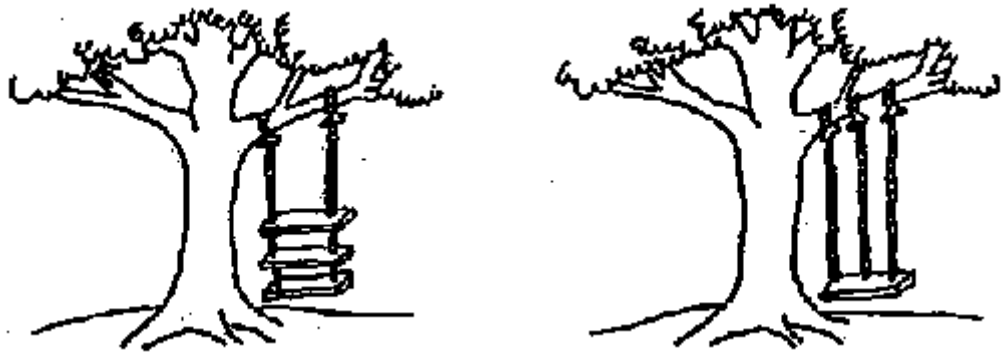

Figure 23 & 24: Design 1 & 2

9 Conclusions

In this thesis, we have discussed open and burning issues on models validation. It is our opinion that valid models, which have been proven to be correct within their domains of applicability, contain truth and hence do yield information about the real world.

We have explored different types of validation, tools and techniques employed in validation, when and why especially with regards to complex real-time systems. Comparisons have been made between validation in computational physics and that which is carried out in computer science field while exploring their areas of differences and similarities.

Most importantly, we carried out experiments to validate a model of a VxWorks system, which has led to a discovery and hence our proposal of two helpful guidelines to empirical scientists and engineers working in the field of validation.

10 Future Works

In future, we hope to further consider in greater details, well-fitted semantic definitions of information and knowledge especially as it affects computer science, engineering and other empirical sciences in order to portray the empirical scientists' view of how experimental data is understood and convey information as may be different from several other existing schools of thought on what constitutes information.

We also hope to do further research and experiments to further diversify the guidelines we have proposed in order to deduce more relations for tasks in more scenarios for instance tasks that are interleaved in nature and tasks that employ various communication paradigms different from the ones that have been considered.

References

[1] Averil M. Law, Michael G. McComas. How to build valid and credible simulation models: Proceedings of the 2001 Winter Simulation Conference.
[2] Robert G. Sarget. Verification and Validation of simulation models. A modified version of Sargent 1996b.
[3] Hans Hansson. Real-time systems Manual
[4] Anders Wall. Architectural modelling and analysis of complex real-time systems: Doctoral dissertation ISBN 91-88834-05-0 MRTC.
[5] ED. Derek P. Atheron, Pierre Borne. Concise encyclopaedia of modelling and simulations: Pergamon press.
[6] *J. F. Hetet and R. Mezencev* Mezencev. An Implementation of the Method of Characteristics Using ACSL Software
[7] Balci O, Sargent R.G 1984. A bibliography on credibility, assessment and validation of simulation and mathematical models. ACM Simuletter 15(3), 15-27.
[8] Johan Andersson, Anders Wall and Christer Nordström: Technical report (MRTC. www. mrtc.se)
[9] Anders Wall, Johan Andersson and Christer Nordström. Probabilistic simulation-based analysis of complex real-time systems
[10] Adapted from Shannon, 1975, p. 211.
[11] Ijeoma Sandra Irobi, Johan Andersson and Anders Wall. Correctness Criteria for Models' Validation. www.mrtc.mdh.se, MRTC Report, Mälardalen Real-Time Research Centre, Mälardalen University. ISSN 1404-3041 ISRN MDH-MRTC-163/2004-1-SE, May 2004.
[12] Blackwell's Guide to the Philosophy of Information. Edited by Luciano Floridi.
[13] Suppe vs. Oreskes et al. Verification, Validation, and Confirmation of Simulation Models: May 7, 1998 PHIL 250
[14] Deepak Khazanki, A Philosophical Framework for the Validation of Information Systems Concepts.
[15] Citation Paraphrased from Kaplan, 1964, p. 52-54. Kaplan, A. The conduct of inquiry, Scranton, PA: Chandler Publ. Co., 1964.
[16] Hunt, S., Marketing theory: The philosophy of marketing science, Homewood, IL: Richard D. Irwin, Inc, 1990.
[17] Dewey, J. How We Think, Boston: D. C. Heath & Co., 1910 (1933 reprint).
[18] Laurent Kaiser, Francoise Simonot-Lion. An Hybrid method for the Validation of Real-Time Systems.
[19] About Uppaal. www.uppaal.com
[20] Uppaal in a Nutshell. Kim G. Larsen, Paul Pettersson and Wang Yi. In Springer International Journal of Software Tools for Technology Transfer 1(1+2), 1997
[21] Automatic Verification of Real-Time Communicating Systems by Constraint Solving. Wang Yi, Paul Pettersson and Mats Daniels. In Proceedings of the 7*th* International Conference on Formal Description Techniques, Berne, Switzerland, 4-7 October, 1994
[22] Compositional and Symbolic Model-Checking of Real-Time Systems. Kim G. Larsen, Paul Pettersson and Wang Yi. In Proceedings of the 16th IEEE Real-Time Systems Symposium, Pisa, Italy, 5-7 December, 1995.
[23] Efficient Verification of Real-Time Systems: Compact Data Structure and State-Space Reduction. Kim G. Larsen, Fredrik Larsson, Paul Pettersson and Wang Yi. In Proceedings of the 18th IEEE Real-Time Systems Symposium, pages 14-24. San Francisco, California, USA, 3-5 December 1997.

[24] Efficient Timed Reachability Analysis Using Clock Difference Diagrams. Gerd Behrmann, Kim G. Larsen, Justin Pearson, Carsten Weise, and Wang Yi. Accepted for presentation at CAV99.
[25] Christer Norstrom, Anders Wall and Wang Yi. Timed Automata as Task Models for Event-Driven Systems. In proceedings of RTCSA'99, 1999.
[26] William L. Oberkampf, Timothy G. Trucano and Charles Hirsch. Verification, Validation and Predictive Capability in Computational Engineering and Physics, 2002.
[27] Roach P. J. Verification and validation in Computational Science and Engineering, Hermosa Publishers, Albuquerque, NM, 1998.
[28] R. Rebba and S. Mahadevan. Verification and Validation under uncertainty. *Submitted to SAMO 2004 conference, Santa Fe, NM.*
[29] AIAA. Guide for the Verification and Validation of Computational Fluid Dynamics Simulations, American Institute of Aeronautics and Astronautics, AIAA-G-077-1998, Reston, VA, 1998.
[30] Cosner, R. R. CFD Validation requirements for Technology Transition, AIAA Paper No. 95-2227, *26th AIAA Fluid Dymanics Conference, San Diego, CA, 1995.*
[31] Lin S. J., Barson, S. L., and Sindir, M.M. Development of Evaluation Criteria and a Procedure for Assessing Predictive Capability and Code Performance, *Advanced Earth-to-Orbit Propulsion Technology Conference, Marshall Space Flight Centre, Huntsville, AL, 1992.*
[32] Marvin, J. G. Perspective on Computational Fluid Dynamics Validation, AIAA Journal, Vol. 33, No. 10, 1995; 1778 – 1787.
[33] Sindir, M. M., Barson S. L., Chan, D. C., and Lin, W. H. On the Development and Demonstration of a Code Validation Process for Industrial Applications, AIAA Paper No. 96-2032, 27th AIAA Fluid Dynamics Conference, New Orleans, L.A. 1996.
{34}Henrik Thane. Monitoring, Testing and Debugging of Distributed Real-Time Systems. Doctoral thesis, Mechatronics Laboratory, Royal Institute of Technology, Stockholm Sweden.
[35] Christer Nordström, Anders Wall, Johan Andersson and Kristian Sandström. Increasing maintainability in complex industrial real-time systems by employing a non-intrusive method

Appendix A

ExpertRules Trace Comparison Report

Comparison Specification File: C:\ART-FW\toyexample\simulation\toy.ppl
Trace1: C:\ART-FW\toyexample\simulation\sim1.trc
Trace2: C:\ART-FW\toyexample\case1-1.trc

Property: max(ctrl(i).exec)
Result1: 4320
Result2: 4244
Guard: reldiff 0.05
Guard status: Ok

Property: P(ctrl(i), ctrl(i).exec < X) >= 0.98
Result1: 4310
Result2: 4213
Guard: reldiff 0.05

Guard status: Ok

Property: P(ctrl(i), ctrl(i).exec < X) >= 0.90
Result1: 4205
Result2: 4108
Guard: reldiff 0.05
Guard status: Ok

Property: P(ctrl(i), ctrl(i).exec < X) >= 0.80
Result1: 4015
Result2: 3964
Guard: reldiff 0.05
Guard status: Ok

Property: P(ctrl(i), ctrl(i).exec > X) >= 0.80
Result1: 3310
Result2: 3317
Guard: reldiff 0.05
Guard status: Ok

Property: P(ctrl(i), ctrl(i).exec > X) >= 0.90
Result1: 3205
Result2: 3209
Guard: reldiff 0.05
Guard status: Ok

Property: P(ctrl(i), ctrl(i).exec > X) >= 0.98
Result1: 3105
Result2: 3098
Guard: reldiff 0.05
Guard status: Ok

Property: max(ctrl(i).resp)
Result1: 5480
Result2: 5304
Guard: reldiff 0.05
Guard status: Ok

Property: P(ctrl(i), ctrl(i).resp < X) >= 0.98
Result1: 5370
Result2: 5243
Guard: reldiff 0.05
Guard status: Ok

Property: P(ctrl(i), ctrl(i).resp < X) >= 0.90
Result1: 5245
Result2: 5098
Guard: reldiff 0.05
Guard status: Ok

Property: P(ctrl(i), ctrl(i).resp < X) >= 0.80
Result1: 5115
Result2: 4956
Guard: reldiff 0.05
Guard status: Ok

Property: P(ctrl(i), ctrl(i).resp > X) >= 0.80
Result1: 4365
Result2: 4297
Guard: reldiff 0.05
Guard status: Ok

Property: P(ctrl(i), ctrl(i).resp > X) >= 0.90
Result1: 4245
Result2: 4183
Guard: reldiff 0.05
Guard status: Ok

Property: P(ctrl(i), ctrl(i).resp > X) >= 0.98
Result1: 4115
Result2: 4067
Guard: reldiff 0.05
Guard status: Ok

Appendix B1

Sensor task	Execution (ms)		Sensor task	Response (ms)
5986	735		5986	758
8009	257		8009	257
10010	263		10010	263
12026	234		12026	234
14010	264		14010	264
16012	253		16012	253
18011	274		18011	274
20017	287		20017	287
22010	262		22010	262
24011	269		24011	269
26010	251		26010	251
28010	243		28010	243
30015	263		30015	263
32010	256		32010	256
34010	261		34010	261
36011	260		36011	260
38012	267		38012	267
40015	281		40015	281
42011	233		42011	233
44010	259		44010	259
46011	267		46011	267
48010	255		48010	255

50014	276		50014	276
52011	259		52011	259
54010	238		54010	238
56013	250		56013	250
58011	265		58011	265
60015	265		60015	265
62010	244		62010	244
64011	249		64011	249
66011	256		66011	256
68011	253		68011	253
70015	276		70015	276
72011	247		72011	247
74011	241		74011	241
76011	241		76011	241
78012	241		78012	241
80015	259		80015	259
82011	255		82011	255

Table 1: Execution time of sensor task (only a cross section of the data represented here)

Appendix B2

ctrl	Execution time (ms)		ctrl	Response time (ms)
64	23		64	23
8294	3765		8294	4310
18327	3806		18327	4774
28306	3255		28306	4250
38322	3737		38322	4744
48326	3895		48326	4911
58293	3631		58293	4644
69610	3518		69610	4800
78297	3165		78297	4201
88326	3991		88326	4978
98297	3192		98297	4193
108316	3769		108316	6250
118310	3182		118310	4162
128318	3608		128318	4600
138328	3852		138328	4875
148553	3859		148553	4895
158306	3994		158306	4998
168327	4028		168327	5083
178299	4107		178299	6613
188296	3127		188296	4106
198303	3964		198303	4932
208301	3607		208301	4586
218313	3708		218313	4731
228308	3112		228308	5393
238318	3475		238318	4397
248306	4137		248306	5091
258319	3238		258319	4189
268327	3428		268327	6012

278307	3889		278307	6393
288321	3878		288321	6461
298310	4036		298310	5005
308323	4163		308323	6649
318323	4184		318323	6722
328322	3956		328322	4939
338310	3190		338310	4202
348325	3921		348325	4966
358323	3183		358323	4182
369564	3792		369564	5024
378566	4127		378566	5056

Table 2: Execution time of control task (only a cross section of the data represented here)

Appendix B3

0	275		0	275
2000	250		2000	250
4000	275		4000	275
6000	285		6000	285
8000	250		8000	250
10000	260		10000	260
12000	240		12000	240
14000	290		14000	290
16000	239		16000	239
18000	265		18000	265
20000	260		20000	260
22000	275		22000	275
24000	290		24000	290
26000	265		26000	265
28000	250		28000	250
30000	260		30000	260
32000	259		32000	259
34000	275		34000	275
36000	250		36000	250
38000	265		38000	265
40000	265		40000	265
42000	265		42000	265
44000	240		44000	240
46000	290		46000	290
48000	285		48000	285
50000	265		50000	265
52000	260		52000	260
54000	265		54000	265
56000	260		56000	260
58000	290		58000	290
60000	250		60000	250
62000	265		62000	265
64000	265		64000	265

66000	265		66000	265
68000	275		68000	275
70000	275		70000	275
72000	240		72000	240
74000	260		74000	260
76000	290		76000	290
78000	265		78000	265

Table 3: Simulated results of Execution and response times the of sensor task (only a cross section of the data represented here)

Appendix B4

Ctrl Task	Execution time (ms)		Ctrl Task	Execution time (ms)
8360	3810		8360	7240
18365	4010		18365	7485
28375	4220		28375	6430
38375	3210		38375	4265
48365	3310		48365	4315
58365	3715		58365	6320
68365	3110		68365	5425
78385	3715		78385	6205
88385	3710		88385	4685
98375	3805		98375	4830
108375	3310		108375	5565
118340	3615		118340	4705
128384	4011		128384	5101
138350	3115		138350	4065
149560	3415		149560	4830
158360	3405		158360	4460
168365	3310		168365	4455
178340	3615		178340	4605
188385	3710		188385	4735
198375	3220		198375	5900
208385	3410		208385	4535
218365	3105		218365	5370
228365	3805		228365	4930
238635	3505		238635	4445
248390	3515		248390	4515
259550	3115		259550	5740
269465	4315		269465	5695
278375	3405		278375	4510
288350	3910		288350	5015
299540	3510		299540	4785
308390	3620		308390	4650
318360	3710		318360	4760
328365	3805		328365	4830
338385	3105		338385	4165
348360	3105		348360	4180
358375	3510		358375	4615

368350	4105		368350	5155
378385	3915		378385	5015
388360	3110		388360	4090
398375	4115		398375	5240

Table 4: Simulated results of Execution and response times the of CTRL task (only a cross section of the data represented here).