Establishing Timing Requirements and Control Attributes for Control Loops in Real-Time Systems

Iain Bate[1], Peter Nightingale[1], Anton Cervin[2]

1 Department of Computer Science,

University of York, York, YO10 5DD, UK

{ijb, pwn101}@cs.york.ac.uk

2 Department of Automatic Control

Lund Institute of Technology, Lund, Sweden.

anton@control.lth.se

Abstract

Advances in scheduling theory have given designers of control systems greater flexibility over their choice of timing requirements. This could lead to systems becoming more responsive, more flexible and more maintainable. However, experience has shown that engineers find it difficult to exploit these advantages due to the difficulty in determining the "real" timing requirements of systems and therefore the techniques have delivered less benefit than expected. Part of the reason for this is that the models used by engineers when developing systems do not allow for emergent properties such as timing. This paper presents an approach and framework for addressing the problem of identifying an appropriate and valid set of timing requirements and their corresponding control parameters based on a combination of static analysis and simulation.

1 Introduction

This paper addresses the perennial problem of how to identify an appropriate and valid set of timing requirements for a hard real-time system. Over the years, research on real-time systems has evolved techniques which provide greater flexibility in scheduling whilst still providing a means for guaranteeing that timing requirements are met [1, 6]. The increased flexibility was expected to give many benefits, including more efficient use of resources and simpler maintenance of schedules when changes to the control software are made. In addition, maintaining schedules is often a costly and error prone manual process, so these techniques have the potential to offer significant economic as well as engineering benefit.

However, experience has shown that engineers find it difficult to exploit this increased flexibility, and the techniques have delivered less benefit than expected. Based on our own experience and that of others in industry [2, 6, 10], a key reason is an absence of information about the *true* timing requirements which are needed to make best use of the approaches. In many cases current systems are developed with simple timing requirements, such as a timing margin to be achieved. (A timing margin is the amount of usable spare capacity available.) In other cases the timing requirements are largely historic, and are simply expressed in terms of iteration rates which have been proven effective in previous designs. Despite the changing contexts between systems, this strategy is normally successful because the requirements are over conservative, e.g. update rates specified are much faster than needed. Even where more modern control law design environments are used (e.g.

Matlab/Simulink [3]), the control models are often produced assuming a particular computational model. For example a 50 ms cycle/20Hz bandwidth is chosen because there is a regular clock tick in the system with a period of 25 ms (i.e. 40 Hz) and therefore it is easier to release tasks at a harmonic of this frequency.

Other techniques such as Shannon's sampling theroem [5] place an upper bound on the sampling period. When the sampling theorem is used, an actual sampling period still needs to be selected as well as other timing attributes such as the deadline of the sampling task, period and deadline of the actuator task, and the maximum separation time between data capture and sensor actuation.

A major contributor to the situation that has arisen is because both the research and practical use of control theory and scheduling theory have largely been carried out in isolation [4]. Thus for example, work on how advanced control regimes, such as H∞ [5], might ease the integration issues between control and software, have received little attention. Other pressures include the move towards model-based development that places greater onus on capturing evidence within the actual models and including low-level implementation details within the models, i.e. emergent properties such as timing.

This paper presents an approach and framework for addressing the problem of identifying an appropriate and valid set of timing requirements in order that the best use can be made of the advances in scheduling theory. The paper is an extension to previous work [15] that adds greater traceability back to the system's objectives using an argumentation technique to target the evaluation used in the framework, and for evaluation purposes using Jitterbug to perform static analysis [11] and the use of scenario-based assessment to determine the extent to which the system copes with other situations - e.g. changes to the system, errors in models and measurements, and random failures.

The approach taken is to first establish the objectives of importance (based on argumentation techniques used in the critical systems domain) and then use component-based models that allow for emergent properties of systems (in this case timing) so that the models are more representative of how an actual system would actually behave. Then, a genetic algorithm is used to explore the design space in-order to identify timing requirements and corresponding control parameters which enable objectives such as control stability to be achieved, thus deriving and validating the requirements against more realistic properties of the control system. When valid combinations of parameters are found, the framework

produces evidence that the solution is appropriate in a traceable manner via static analysis and test.

The advantage of using genetic algorithms instead of traditional model-based design approaches, such as frequency domain loop shaping [5], include it allows many properties and effects to be considered at the same time and their demands on the system to be traded-off against one another [9].

The work presented here is intended for use in a range of control problems, but is illustrated with the PID (Proportional Integral Differential) control approach [5].

The rest of the paper is structured as follows. Section 2 gives further background on the control techniques to be used in the context of this work. It also provides a technical motivation (as opposed to the "economic" motivation outlined above) for seeking a systematic approach to deriving timing requirements. Section 3 gives an overview of an argument that assesses the desirable properties of a control system scheduled on a computer and evolves an experimental method to show the properties are met. Section 4 presents the framework, and the costs of evaluating the requirements. Section 5 contains a case study which have been used to evaluate the approach, as well as presenting a discussion of how the resulting timing requirements may be used. Finally, section 6 gives a summary and suggests possible future developments for the work.

2 Background and Motivation

All scheduling approaches require a minimum set of information about timing requirements so that an appropriate scheduler can be produced. For most scheduling approaches the minimum set of information is the deadline and period of tasks [6, 7]. This section explains why these requirements are important in the context of PID loops and how they can be generated by considering basic control properties.

2.1 PID Loop

The main purpose of a PID loop is to ensure the response to inputs is sufficiently fast whilst maintaining the stability, accuracy and limits on data. Figure 1 depicts a typical PID loop used to control the operation of a plant as part of a control system. The Figure shows the key aspects and components of the controller – e.g. there is only one input and one output.

In its simplest form, a continuous ideal domain representation, the output of the PID loop is the plant input. The control system input is the difference between the input demand (denoted by I), which is the desired plant state, and the plant's actual output (denoted by O) and it is referred to as the error, (denoted by E). The continuous and discrete forms of the PID loop are given in Equation 2 and Equation 4 (current sample denoted by k) respectively.

$$E(t) = O(t) - I(t) \qquad \textbf{Equation 1}$$

$$O(t) = K_P E(t) + K_I \int E(t)dt + K_D \frac{dE(t)}{dt} \qquad \textbf{Equation 2}$$

$$E(k) = O(k) - I(k) \qquad \textbf{Equation 3}$$

$$O(k) = K_P E(k) + K_I \sum_{j=1}^{k} E(j) + K_D [E(k) - E(k-1)] \qquad \textbf{Equation 4}$$

In the computer-based approach, the *Input Demand* (e.g. pilot stick position) and the *Actual Plant Output* (e.g. aircraft's flap position) are usually analogue signals. The computer performs the rest of the processing in the digital domain. Converters are used to sample the analogue signals, e.g. to produce the *Error* input, and then converted back to analogue values at the output. Converting back to an analogue signal is often referred to as digital to analogue conversion, de-sampling or actuation. In order to give better control over jitter, the functionality that needs to be performed in software is normally split into three separate tasks – sampling, calculation and actuation [4, 6, 7].
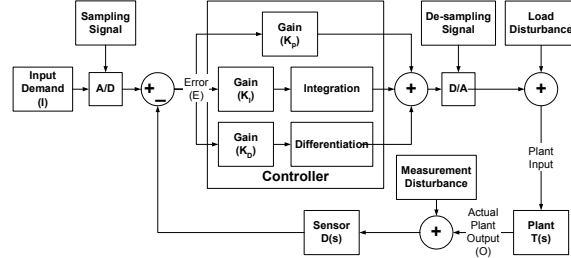

Figure 1 – Typical PID Loop

In industrial practice it is common for a controller to be developed as a continuous system based on the system's response in the frequency domain. Often modelling packages or special purpose plant simulations are used to validate the requirements. If a computer-based implementation is to be used, then once the requirements have been established in the continuous domain they are converted to the discrete domain. Typically the conversion involves calculating the PID loop gains (K_P, K_I, K_D) based on the assumption that a constant sampling period is used. This means the conversion is performed based on an idealised model of the computer system. The conversions for the PID loop gains are give in Table 1. In other words the conversion uses unrealistic assumptions, e.g. infinite processing bandwidth and zero jitter in sampling the inputs (jitter is the variation in time when an action occurs between one cycle of the controller and the next). In addition, "real" systems have errors through effects such as measurement disturbance, load disturbance and plant error. These are also represented in Figure 1.

Parameter in Continuous Domain	Discretisation Formula for a Sampling Period of T
K_P	K_P
K_I	$T.K_I$
K_D	K_D / T

Table 1 – Conversion from Continuous to Discrete

The approach presented in this paper addresses this shortcoming by taking into account the constraints of real computer systems, and thus enables valid and realistic requirements to be produced. To explain how this is done the rest of this section explains in more detail the relationship between computational properties such as jitter and control properties such as stability.

2.2 Scheduling Properties

It is, of course, essential that the sampling, core functions and de-sampling tasks are executed in that

order. Other work, e.g. [6], has shown how to specify and control the precedence of functionality for a PID loop to ensure that these requirements are met. More importantly for our discussions in this paper, sampling and de-sampling will be subject to jitter due to limits on the accuracy of clocks, and the interference of other software running on the processor. Thus the true sampling times will vary, and the simple assumptions of fixed and precise iteration rates used in validating the control model will not be representative of the computations which occur in practice. Jitter can be calculated using the results of response time analysis that gives best-case [14] and worst-case response times [6].

Figure 2 presents properties for a typical transaction of a control loop that can be controlled by the scheduler. The three tasks are sensor capture, calculation and actuation output. The Figure shows:

- how each task has jitter comprising both release and execution jitter as well as an invariant in its execution time,
- there is jitter on both sensor capture (referred to as sampling jitter) and actuation (referred to as de-sampling jitter),
- a task must be completed before the next task in the transaction starts its execution so that the next task can use fresh data,
- the response time of a transaction is equal to the time between the release of the first task and the completion of the last task (the worst-case response time for a transaction must be less than its deadline), and
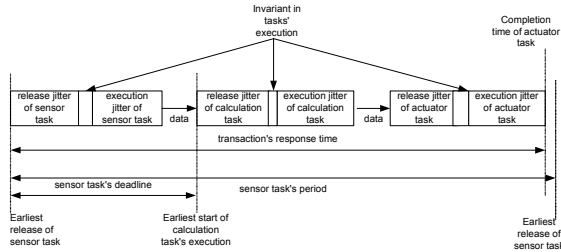- the period of a task is the time between two consecutive earliest releases.

Figure 2 - Scheduling Properties for a Transaction

2.3 Control and Scheduling Interactions

The previous section defined our scheduling model and its associated terms. The only events that are important for control are sampling and actuation. When assessing the system's control performance, it is assumed the three tasks shown in Figure 2 meet their timing requirements. For both sampling and actuation we assume the event occurs at the end of the execution of the relevant task. We define a number of parameters, which specify the timing requirements over the sampling and actuation. These are defined in Equation 5 - Equation 10, where SENS and ACT refer to the sensor task and actuation task, J refers to jitter, D to deadline, *BCRT* to best-case response time, *WCRT* to worst-case response time, *BCET* to best-case execution time and S to separation between actuation and sampling. It should be noted that for idealised cases where J_S and/or J_A are zero stability analysis is significantly simplified. For all other cases,

Jitterbug can be used where the distributions for J_S and J_A are known.

Requirements derived purely from the control model without allowing for computational effects caused by scheduling and execution are invalid. More seriously, they are also inappropriate as they can lead to instability, and other undesirable properties, e.g. lack of responsiveness.

The classic definition of stability in general terms is that a system is stable if bounded inputs return outputs that remain bounded for all time [5]. The stability of the system is related to the frequency response of the control system (and hence the calibration settings for the control loop) as well as the other properties discussed. From a scheduling perspective, only the responsiveness and size of the errors can be controlled since the gain is a functional property of the control software.

Sampling jitter (J_S)	$J_S = J_{SENS}$	**Equation 5**
Separation (S)	$S = RT_{ACT} - RT_{SENS}$	**Equation 6**
Actuation jitter (J_A)	$J_A = J_{ACT}$	**Equation 7**
Minimum Separation	$S_{MIN} = BCRT_{ACT} - WCRT_{SENS}$	**Equation 8**
Maximum Separation	$S_{MAX} = WCRT_{ACT} - BCRT_{SENS}$	**Equation 9**
Deadline	$D = S_{MIN} + BCET_A + BCET_S + J_A(MAX) + J_S(MAX)$	**Equation 10**

3 Linking Requirements to an Argument

When a control system is being developed as part of a critical application, the traditional approach is to design the system and then thoroughly evaluate it to provide evidence for its safety argument. By following this technique, it is intended that the results obtained from the framework become more focussed.

An objective of our work is to collect evidence at the model level that is valid with respect to the final system, i.e. allowing for emergent properties within the model. The remaining evidence must then be collected from the final system. This strategy means that any evidence collected about the behaviour of the system using the model is directly related to the evidence needed to justify the integrity of the final system. This is a significant departure from current practice where most of the evidence is gathered about the final system. The benefits of achieving this strategy are considered enormous, these include; more systematic reuse, lower cost of verification facilities and a later commitment to target hardware. It is recognised that some information (e.g. execution times) can only be found on the target but the impact of this should be kept to a minimum.

The argument presented here has been produced using Goal Structuring Notation (refer to section 3.1 for an overview) to establish and justify an appropriate methodology.

3.1 Overview of the Goal Structuring Notation

The Goal Structuring Notation (GSN) [13] - a graphical argumentation notation - explicitly represents the individual elements of any safety argument

(requirements, claims, evidence and context) and (perhaps more significantly) the relationships that exist between these elements (i.e. how individual requirements are supported by specific claims, how claims are supported by evidence and the assumed context that is defined for the argument). The principal symbols of the notation are shown in Figure 3 (with example instances of each concept).

The principal purpose of a goal structure is to show how goals (claims about or objectives of the system) are successively broken down into sub-goals until a point is reached where claims can be supported by direct reference to available evidence (solutions). As part of this decomposition, using the GSN it is also possible to make clear the argument strategies adopted (e.g. adopting a quantitative or qualitative approach), the rationale for the approach (assumptions, justifications) and the context in which goals are stated (e.g. the system scope or the assumed operational role). For further details on GSN see [13].
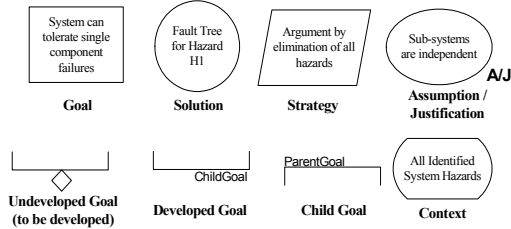

Figure 3 – Principal Elements of GSN

3.2 Top-Level Argument that the System Meets its Objectives

For reasons of space, only the top-level argument is given in this paper. The top-level argument is presented in Figure 4. The assumption **A0001** in the argument identifies the following primary objectives to be met by the system as a minimum are:

- *stability margin* – the stability margin is specified in invariant systems in terms of the minimum return difference [5]. A margin is chosen to ensure the system is not at risk of being unstable. More generally, it is the distance from the instability point on a Nyquist plot [5].
- *actuator limits* – the maximum allowable output. This limit is often chosen to prevent damage to components or give rise to system hazards.
- *settling time* – given a step response as an input, the time taken for the output to reach and stay within X% of the final value. Both X and the time taken are specified to ensure the system is suitably responsive to stimuli.
- *maximum error* – the maximum allowable error at a given time between the intended output and the actual output is specified to ensure the output is always within a bounded range. This objective covers effects due to both slew rate and overshoot.

The strategy for satisfying these goals is to split the evidence gathering into two parts; obtaining evidence that is gathered at the model level (goal **G0002**), and obtaining evidence that the model is then transformed correctly into a final system (goal **G0003**). This strategy is justified by drawing on an analogy with current

practice whereby requirements are validated and then it is verified the implementation meets the requirements – **J0001**. The proposed strategy is a significant departure from current practice where most of the evidence is gathered about the final system. The key challenges are capturing assumptions of the model and ensuring these are representative of the final system – assumption **A0004**. In particular most models do not account for the non-functional properties of the system.

The goal that the model is correctly transformed into an implementation (goal **G0003**) is left undeveloped here since other work has addressed this problem [12]. It is recognized that whilst implementing what is contained in the models some new properties may emerge (e.g. timing requirements related to interaction with specific hardware) and changes to requirements may be necessary. Any changes should be incorporated back into the model so that it can be checked whether the previously gathered evidence is invalidated. This approach is preferred so that evidence gathered while implementing the system can be used purely to reinforce the evidence gathered at the model level.

Finally, goal **G0002** is decomposed into three parts; capturing evidence a particular design meets the system's objectives, searching for an appropriate design and that a suitable framework is used. The results of the decomposition of these objectives is discussed in the following sections.
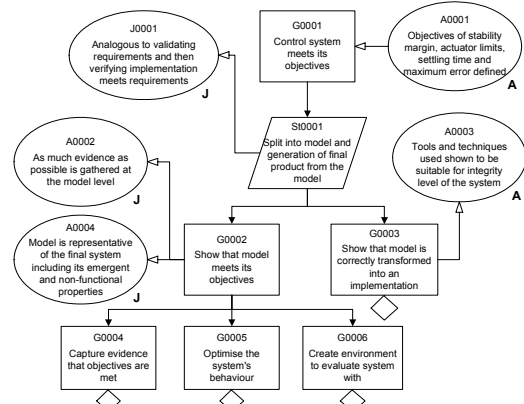

Figure 4 – Top Level Argument that the Control System Meets its Objectives

3.3 Gathering Evidence that the System's Objectives are Met

In the rest of the arguments, it is determined that the evidence is captured in two phases, simulation and static analysis. Simulation can provide detailed evaluation over a range of scenarios (e.g. changes and failures), while the main area in which static analysis provides conclusive results for control systems is in showing that the system is stable across all scenarios. The majority of evidence is to be gathered by simulation with static analysis used to provide independent confirmation of the results obtained. In many situations it is the results from the static analysis that is considered to be of paramount importance. Jitterbug is used to provide static analysis that incorporates the effects of jitter and latency [11].

Jitterbug is a MATLAB toolbox that allows the user to compute a quadratic performance index for a linear

control system with timing variations [11]. It can also be used to analyze frequency-domain properties of the closed-loop system. In this paper, Jitterbug is used to compute the stability margin of a controller with sampling jitter and actuation jitter.

The simulation evidence to be gathered is split across the objectives taken from the arguments with timing behaviour added to recognise the system is to be executed on a computer which introduces time variations. The evidence for the objectives is collected as follows.

1. *evaluate measurement disturbance rejection* – over the range of scenarios identified, add noise at the sensor and check that the system meets all other requirements.
2. *evaluate steady state response* – over the range of scenarios identified, measure the difference (i.e. error) between the actual response and the ideal response and ensure any error is less than the maximum specified.
3. *evaluate transient response* – using step responses, measure the settling time and maximum error to check whether the requirements are met.
4. *evaluate plant sensitivity* – over the range of scenarios identified, vary the plant model to the maximum that the system is intended to tolerate, and check whether the requirements are met. Plant variation is discussed further in section 4.2.
5. *evaluate temporal sensitivity* – over a range of periods, determine valid timing requirements which lead to the system meeting its objectives.

The simulation evidence to be gathered is to be collected in the following input scenarios. Based on experience, these scenarios are considered comprehensive especially when it is considered static analysis is also performed.

1. *input step response* – provides a constant amplitude across the entire bandwidth of the system.
2. *load step response* – ensures plant disturbances are rejected by the system.
3. *ramp response* – ensures that the system can cope with the specified rate of change without the maximum allowable error being exceeded.
4. *parabolic response* – allows the response of the system to be evaluated at a range of frequencies.
5. *random response* – allows the system to be evaluated with little effort and still provides useful data.
6. *representative response* – allows the system to be evaluated using scenarios taken, and maybe modified, from similar systems or from predictions of how the system may be used.
7. *noise* – for the other options, it is relevant that sensor noise of a specified level can be added to simulate real world effects.
8. *plant variations* – for the other options, it is relevant that variations in the plant model are examined to show the sensitivity of the system to component tolerances, design changes and mechanical wear.

3.4 Exploring the Design Space

Heuristic search techniques were chosen to explore the design space to find the best combination of variables to satisfy the previously stated objectives. These methods can be broadly split into two areas: Partial Assignment Search and Total Assignment Search.

Partial assignment search requires evaluation of the system when not all variables have a value, which is unsuitable in our case since we cannot simulate or perform static analysis with an incomplete parameter set. This leaves total assignment search (also called local search). There are many suitable approaches in this area, but simulated annealing and genetic algorithms have been successfully applied to engineering problems in control.

Simulated annealing requires only a fitness function, whereas a genetic algorithm requires a representation of the data in a suitable form, and genetic operators such as crossover and mutation in addition to the fitness function. However, genetic algorithms are less sensitive to changes in the evaluation function, since the individuals are ranked according to their fitness, and the set of fitness values are discarded. In our case, defining a suitable representation, and mutation and crossover is trivial. Simulated annealing sometimes determines the probability of performing moves from the magnitude of the fitness, and hence finding a suitable fitness function becomes more difficult. For this reason, we chose to use a genetic algorithm.

When performing the optimisation, it is assumed the period is constant across the tasks in a particular control loop but the maximum jitter may be different for sensor and actuator tasks.

4 Modelling Approach and Framework for Evaluating Timing Requirements

For the purpose of this work, no specific scheduling approach is assumed. Instead, it is assumed that the "scheduling problem" can be divided into two parts: devising a set of timing requirements and verifying that a chosen schedule meets those requirements. There are several solutions to the latter problem, e.g. [1, 6], so for the purposes of this paper, we consider this to be a solved problem, and focus on the issue of generating requirements. Our approach seeks to build on the capability of existing tools for developing control laws, and to exploit the power of genetic algorithms to explore the "design space" produced by the interplay of task periods, deadlines and jitter as well as the loop gains (K_P, K_I, K_D).

There are four significant parts to the framework that has been developed in MATLAB [3]:

1. Evaluating the PID Operation
2. Searching the design space
3. Determining a valid set of timing requirements by stepping through a range of periods and tuning the other timing parameters alongside those of the PID loop.
4. Generating a detailed log (with figures) by documenting the analysis of the solution at each period.

The following subsections discuss each of these parts in more depth.

4.1 Evaluating the PID Operation

The MATLAB model for simulating a discrete PID control loop is shown in Figure 1. The key elements here are that the gains of the PID loop are programmable, and that the timing of the sensor and actuation is controlled by the representative sampling signal. The purpose of the signal is to simulate the jitter on when actions (sampling and actuation) occur as shown in Figure 2. That is, the signal is used to control when signals are converted from the continuous to the discrete domain and vice versa. The signal is generated such that one sampling event and one actuation event occur within each period, and in the correct order. If time 0 is the beginning of the period, the sampling event occurs at time δ, where δ is a random value in the range $(0, J_S]$ and J_S is the maximum possible sampling jitter according to Equation 10. The falling edge occurs at time $\delta+S+\varepsilon$, where S is the minimum separation, and ε is a random value in the range $[0, J_A)$. The distribution of δ and ε is uniform. The sampling signal is fed into Jitterbug since Jitterbug needs stochastic knowledge of timing behaviour in-order to evaluate stability via static analysis. As discussed in section 3.3, simulation is also employed to evaluate the performance.

Figure 1 shows three inputs to the system. Input and load disturbance are used to simulate various scenarios (refer to Table 4 for details), and the measurement disturbance is used to test the effects of noise on the system. The level of noise is a parameter of the framework. Varying degrees of noise can be added to the simulations. This simulation model is used to explore the performance of the control system with a given set of parameters, e.g. values for K_P, K_I, K_D, J_S, J_A, S_{MIN}, etc.

4.2 Allowing for Plant Variations

The framework uses a transfer function to model the plant, and if required a second transfer function to model the sensor (otherwise the signal is passed through the simulated sensor block unchanged). The transfer function represents a linear model of the plant. A plant model is never 100% accurate because modelling introduces inaccuracies and the plant may have non-linear characteristics that can't be accurately represented in a linear model. Internal factors such as component tolerances and mechanical wear also alter the behaviour of the system. It is also common for plants to change their behaviour based on external factors such as temperature and altitude. To account for these inaccuracies and online variations, the system is evaluated with variations of the transfer function.

	Plant model	Sensor model
Nominal model	$T(s) = \dfrac{1}{1000\ s + 50}$	$D(s) = \dfrac{1}{s+50}$
Model with other variables	$T(s) = \dfrac{a}{(1000+b)s+50}$	$D(s) = \dfrac{c}{s+50}$

Table 2 – Example Models with Additional Variables

In order to vary the transfer function, the framework allows variables to be added to it. Each of these additional variables has a nominal value, which form the nominal model if substituted into the transfer function. Table 2 shows an example of plant and sensor models with and without additional variables, and Table 3 shows the nominal, minimum and maximum values for the variables a, b and c. Normally the sensor model is incorporated within the plant model in which case c would be zero.

To account for plant variation requires one additional parameter - the number of steps to be taken between the minimum and maximum of each variable. For example, a variable with minimum of 3 and a maximum of 7, where the number of steps is set to 5, would iterate through values {3,4,5,6,7}. The number of plant variations considered is N^{Steps}, where N is the number of variables, so the framework evaluates every permutation. The nominal model is also evaluated, so it need not be one of the variations. Each of the variations is subjected to the same tests, as summarised in , as the nominal model.

Variable	Nominal value	Minimum	Maximum
a	1	0.95	1.05
b	0	-5	5
c	1	0.9	1.1

Table 3 – Example Values for Additional Variables

Simulation input scenario	Parameters
Load step disturbance	Size of step
Input demand step disturbance	Size of step (equal to above)
Ramp response	Gradient of the input demand
Parabolic response	Frequency of sine input demand
Random response	Variance of the random signal which forms the input demand
User-specified	Time series data which forms the input demand of the simulation

Table 4 – List of Simulation Scenarios

4.3 Searching the design space

It is possible to use PID loop gains in the discrete domain that were originally established in the continuous domain but first they must be transformed using relationships based on the sampling period in the continuous domain and the discretisation formula. In the context of our approach, the relationships are as shown in Table 1. However in the framework produced, an option is to tune the actual gains for any given period and/or deadline. In section 4.3, a genetic algorithm was chosen for the purpose of optimising performance. The fitness function used in conjunction with the genetic algorithm allows many properties and effects to be considered at the same time and their demands on the system to be traded-off against one another.

If the search is successful in meeting the requirements, a valid set of timing parameters will be evolved alongside the controller parameters. The search is not considered to be successful if the timing requirements are invalid compared to the assumptions selected, e.g. if the deadline is greater than the period. To reduce the

number of variables in the search, J_A is determined from J_S using an expression provided by the user, for example $J_A = 2J + 1$, and the best-case execution times of the actuator and sensor tasks are assumed to be zero which is the worst-case scenario. Hence based on Equation 5 through Equation 10, the variables involved in the search are J_S, S_{MIN}, K_P, K_I and K_D.

The fitness function evaluates the system as described in section 4.1 and returns a score dependent on the success in the various tests described there.

The inputs of the fitness function are:

1. The individual to be evaluated – values of J_S, S_{MIN}, K_P, K_I and K_D.
2. The range of the actuator – i.e. the largest range of output allowed from the controller.
3. The settling time following a load step or input demand step, along with the maximum allowed error after the settling time has expired.
4. The maximum allowed error during the whole simulation, under all the scenarios in Table 4.
5. The period of the sensor capture task – i.e. the sampling period. It is assumed that other tasks in the transaction have a rate that is equal to the sensor capture task's rate.
6. The timing requirements S_{MIN}, J_S and J_A as defined in Equation 5 to Equation 10.
7. The length of the simulation – this should be appropriate for the system.
8. The level of measurement disturbance noise applied under all the scenarios in Table 4.
9. Target value for the stability margin (distance from the instability point on a Nyquist plot [5]) as computed by Jitterbug [11].
10. Plant and sensor models, and the appropriate information about their variables, described in section 4.2.
11. Those related to the input scenarios in Table 4.

Score range	Description
[1, 2)	Testing actuator limits without jitter (with set-point and load step simulations)
[2, 3)	Testing settling time without jitter (with set-point and load step simulations)
[3, 4)	Testing maximum error without jitter (with set-point and load step simulations)
[4, 5)	Computing stability margin with jitter [11]
[5, 6)	Simulating whole range of scenarios (Table 4) without jitter and checking maximum error and settling time
[6, 11)	Repeating the above tests with random jitter
[11, 12)	Repeating all the tests for score range [1, 11) for each of the system variants specified by the user
[12, ∞)	Additional score, higher for longer deadlines

Table 5 - Scoring in the fitness function

The fitness function determines a score as illustrated in Table 5. Each score indicates complete success in the lower categories, hence a score above twelve indicates that the solution meets all the requirements. All simulations are performed with measurement noise added. For a solution to be considered it must pass all the individual tests. During the evaluation, if no design solution can be found that passes all tests then the designer should evaluate whether the system's requirements can be relaxed or whether a change can be made to the search limits (e.g. range of periods).

For space reasons, full details of the genetic algorithm used are not provided – e.g. cost function, mutations.

4.4 Determining Valid Timing Parameters

Before the evaluation, the user, via a provided interface, defines the limits for the period of individual activities within the system (e.g. sampling of signals) and the size of steps taken between the limits (e.g. periods between *zero* and *ten* might be evaluated in steps of *one*, resulting in search and evaluation being performed at 0, 1, 2, ·· , 9, 10). For each of these periods the set of deadlines, and hence the maximum deadline, are determined for which the control system meets its objectives, e.g. stability.

The results of the searches can be displayed in various forms, to aid the user in picking the most appropriate. When selecting the most appropriate of the valid solutions, the user may prefer the longest period for which a solution was found, in order to reduce the utilization of the processor. Alternatively, the user may prefer the solution with the longest deadline to give a lower priority in order to improve schedulability when using a deadline monotonic priority ordering.

Once the search has found individuals that satisfy the other requirements described in section 1.1, it attempts to maximise the transaction deadline

4.5 Generating a log of safety evidence

Upon finishing the search, the system repeats the evaluation of the best solution, recording details of scenarios evaluated, responses to stimuli, static analysis results and graphs in a log. The log contains evidence such the extensive simulations of the scenarios described in Table 4 under the effects of noise, jitter, latency and periodic execution, and results of stability analysis. Both static analysis and test via simulation are performed for many variations of the plant, to account for plant variation and/or uncertainty.

5 Evaluation

In this paper, a ball and beam example [8] is used to demonstrate the technique because it normally operates in an unstable fashion with stability only being possible with external control assistance. Therefore it represents an important, albeit rather smaller, class of systems, e.g. military fast jet flight control systems.
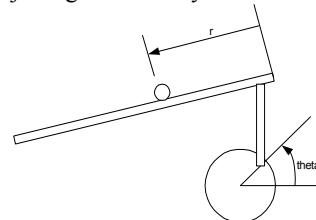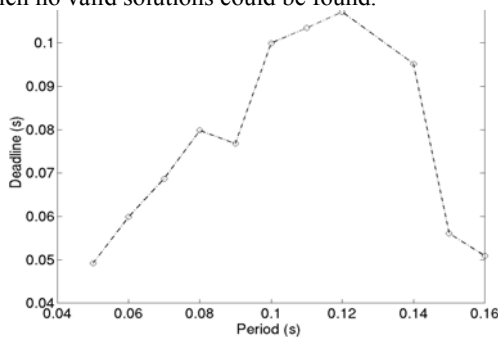

Figure 5 – Ball and Beam

The plant (i.e. the ball and beam [8]) is illustrated in Figure 5, where the variable to be controlled is the position of a free-rolling ball on a beam. It can be mathematically represented by a second-order system using Equation 11. The controller output represents the angle of the gear, $\theta(s)$, and the variable under control is the position of the ball, $r(s)$. The ball and beam is

unstable without control and as such is more difficult than examples where the control system is inherently stable.

$$\frac{r(s)}{\theta(s)} = \frac{0.21}{s^2}$$

Equation 11

The system has been simplified so that it can be represented as a linear equation. It was linearized around the point where the angle of the beam is zero with respect to the horizontal, and also a linear approximation was made in the transmission from the gear to the beam.

The evaluation was performed on a PC running Linux with an Athlon 700 MHz processor. The evaluation time for each period was approximately four hours. The results for the experiment are given in Figure 6. The results showed that the maximum deadline at a particular period increases until a period of 0.12 is reached. Beyond this period, the maximum deadline at a particular period falls until a period of 0.16 seconds after which no valid solutions could be found.

Figure 6 - Deadline vs Period for the Ball and Beam

5.1 Use of the Results

The paper has presented a means by which timing requirements can be determined. The results obtained can be used in many ways. For instance on some projects, a decision may be made not to operate the system at the limits of its ability. For example, if at period *T* the maximum deadline shown to meet the criteria is *D*, then a deadline of *0.8D* could be chosen in-order to give a "safety" margin. Another way of ensuring the system is not operating at its limits is to input objectives into the framework objectives beyond that actually needed. This would mean the valid solutions that emerge would be on the safe side.

Based on the assumption that a shorter period means we can have a longer deadline and vice versa, the results can be used to help make the system schedulable or scaleable. For instance dependent on the timing requirements associated with the rest of the system, then a larger period may be beneficial in helping reduce processor utilisation. On the other hand for systems with many tasks having short deadlines (making scheduling difficult unless the tasks are phased), it may be better to have a shorter period so that the deadline is longer.

6   Summary and Future Work

In this paper, a framework for deriving and evaluating the timing requirements for control systems has been presented. For a given control system, the framework automatically evaluates the effect of jitter, latency and period on control properties such as settling time and maximum error in order to derive the set of requirements that meet our criteria, including stability. In other words, it addresses in an integrated way the issue of evaluating the effectiveness of the control system, and the possible realisation of the controller on a computer system. The use of the approach has been illustrated for a ball and beam system that is unstable without external control.

Future work could look at applying genetic algorithms across a number of control loop(s) and other tasks at the same time to find the optimum balance of timing requirements for scheduling and scalability.

References

[1]   G. Butazzo, *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*, Kluwer Academic Publisher, 1997

[2]   S. Hutchesson and N. Hayes, *Technology Transfer and Certification Issues in Safety Critical Real-Time Systems*, Digest of the IEE Colloquium on Real-Time Systems, 98/306, 1998.

[3]   http://www.mathworks.com

[4]   K. Årzén, A. Cervin, J. Eker and L. Sha, *An Introduction to Control and Real-time Scheduling Co-design*, In Proceedings of the 39th Conference on Decision & Control, 2000.

[5]   R. Harbor and C Phillips, *Feedback Control Systems*, 4th Edition, Prentice Hall, 2000.

[6]   I. Bate, *Scheduling and Timing Analysis for Safety-Critical Systems*, Department of Computer Science, University of York, YCST-1998-04, 1998.

[7]   M. Torngren, *Fundamentals of Implementing Real-Time Control Applications in Distributed Computer Systems*, Real-Time Systems, 14(3), pp. 219-250, 1997.

[8]   http://www.engin.umich.edu/group/ctm/examples/ball/bb PID.html

[9]   I. Bate and T. Kelly, *Architectural Considerations in the Certification of Modular Systems*, Proceedings of Computer Safety, Reliability and Security - 21st International Conference, SAFECOMP 2002, LNCS 2434, pp. 321-333, 2002.

[10] C. Norström, M. Gustafsson, K. Sandström, J. Mäki-Turja and N.-E. Bånkestad, *Experiences from Introducing State-of-the-Art Real-Time Techniques in the Automotive Industry*, Proc. of the 8th IEEE International Conference on Engineering of Computer-Based Systems, 2001.

[11] B. Lincoln and A. Cervin, Jitterbug: *A Tool for Analysis of Real-Time Control Performance*, IEEE Conference on Decision and Control, 2002.

[12] M. Whalen, M. Heimdhal, *On the Requirements of High-Integrity Code Generation*, Proceedings of the 4th High Assurance in Systems Engineering Workshop, 1999.

[13] T.P. Kelly, *Arguing Safety - A Systematic Approach to Safety Case Management*, Department of Computer Science, University of York, YCST-1999-05, 1998.

[14] O. Redell and M. Sanfridson, *Exact Best-Case Response Time Analysis of Fixed Priority Scheduled Tasks*, Proceedings of the 14th Euromicro Conference on Real-Time Systems, 2002.

[15] I. Bate, J. McDermid and P. Nightingale, *Establishing Timing Requirements for Control Loops in Real-Time Systems*, To Appear in the Journal of Microprocessor and Microsystems, 2003.

[16] K. Sandström, C. Norström and G. Fohler, *Handling Interrupts with Static Scheduling in an Automotive Vehicle Control System*, Proceedings of the 5th RTCSA Conference, 1998.