Design For Flexible And Scalable Avionics Systems

I. Bate and P. Emberson
Department of Computer Science
University of York
York, United Kingdom
{iain.bate, paul.emberson}@cs.york.ac.uk

Abstract— Large-scale complex embedded systems pose unique problems to developers. The development of these systems is often performed in a concurrent and iterative fashion. This has lead to a great deal of work on developing processes and product technologies to support scalability and flexibility, i.e. managing change. One example of this is the DARPA funded MoBIES project which approaches the problem by allowing the designer to concentrate on the model level. From a real-time systems perspective, one area that needs greater attention is that of task allocation and attribute assignment. The reason is that, whilst a great deal of work has been done on task allocation, it has been targeted at meeting the current set of timing requirements without giving appropriate consideration for the need to manage change.

TABLE OF CONTENTS

1. INTRODUCTION

The embedded systems required for avionics applications are large scale and complex. Avionics systems must also be validated and verified prior to deployment. Any changes made to a system must be retested as must any parts of the system which the changes may impact. If the system is not well designed, a small change could require a disproportionately large part of the system to be retested. This leads to high costs and long development lifecycles.

Although there has been significant work in making avionics embedded system design more modular such as the use of IMA in the Boeing 777 AIMS project[1], less work has been done on assessing the flexibility and scalability of a particular design. Part of the reason for this is that it is difficult to assess how a system will cope with change since the future changes are not always understood in sufficient detail. Avionics systems are hard real-time systems where deadlines must be met. If extra functionality is added to the system, there may be a need to reschedule the software on top of any functional changes. For example, following a change to a task, if the utilisation of the processor on which the task is running becomes too high, some tasks must be relocated to a different processor or a hardware upgrade is required.

Design should be produced so that they are *flexible* and the impact of changing the design is minimised and hence easier to manage. For this to be possible, it is necessary to have a method to quantitatively measure the flexibility of a design and a further method to modify the design in order to improve its flexibility. As the nature of most changes are not known in advance, especially where there may be multiple changes at once, then scenario-based assessment provides a good basis for evaluating the likelihood that changes can be handled. Scenarios may be randomly generated or pre-defined. For example it can be used to assess whether new tasks can be added to a task set. If some areas of the design are known to be more likely to change, the generated scenarios can be biased towards these areas. A value function measures the ability of the solution to meet the different scenarios. By searching the design space to maximise the value, the design can be modified to improve the way the system meets its objectives.

To measure the *scalability* of a design, factors such as individual task sensitivity and whole task set sensitivity are combined into the value function. Sensitivity is defined as the maximum changes that can be handled before the design no longer meets its requirements. The results can be used to give feedback on where the most sensitive parts of the system are. This will show system designers which parts of the system are most likely to break if changes are made.

Metrics such as the number of processors used also constitute part of the value function since the need for a flexible system must be balanced with the cost of the hardware.

In this work, given an appropriate value function, heuristic search techniques such as simulated annealing or a genetic algorithm are used to improve a design's ability to meet its objectives. The search can modify the design by assigning tasks to different processors and also by making lower level changes to task attributes such as scheduling order, e.g. by modifying priority, and release point, e.g. by altering offsets. Each design must be checked to ensure all timing require-

0-7803-8870-4/05/$20.00/
IEEEAC paper #1399

ments are still met. Instead of this being a hard constraint the schedulability of a design is a constituent of the value function to give greater freedom to the search algorithm. That is, a *null* value is not necessarily returned if the design does not meet all its requirements.

The contributions of the paper are the systematic derivation of a suitable value function for evaluating the flexibility and scalability of an embedded system design, showing the results of this function by combining it with a search environment for design and timing analysis.

Section 2 of this paper discusses the quality attributes for embedded systems which are used to guide the development of the value functions in section 3. An evaluation framework based on the value function is then presented in section 4 which is demonstrated through an example in section 5. Finally, section 6 presents the summary.

2. QUALITY ATTRIBUTES FOR SYSTEMS

The aim of this section is to consider the needs of the product and its lifecycle in order to derive typical quality attributes. The lifecycle stages considered are its development and maintenance. The approach taken is to derive the quality attributes for each of these in turn. The consideration of quality attributes should, where possible, be general across all embedded systems. This work focusses on the non-functional properties of systems rather than their functional properties.

For most embedded systems, especially consumer applications, cost is associated with the production and distribution of the final product. That is, the developers are willing to spend more time and money developing products in order to save money on the final production, e.g. optimising the software to fit on a cheaper microprocessor.

Within the scope of this work, change as part of development and maintainability can be considered to be in one of two categories. Firstly, changes that improves the way a set of requirements is met (termed flexibility) and secondly change that involves the addition of new requirements (termed scalability).

For certain classes of system (e.g. large-scale, low volume systems or individual systems in a product family), the cost of development and maintenance may become a much bigger issue. It is noted the issue of *scalability* to support maintenance has so far been raised in two separate places: firstly through dependability and secondly through cost. Therefore it is felt the issues surrounding it should be given extra attention.

Quality Attributes for the Final Product

The quality attributes of any final product are that it meets its non-functional requirements with the lowest cost and so that it is fit for purpose. The non-functional requirements of systems were summarised by [2] as timing, power, memory and dependability. Dependability is further refined by Laprie [3] to be reliability, availability, safety, confidentiality, integrity and maintainability.

A system however cheap is of little use if it is not considered fit for purpose. *Fit for purpose* covers a wide range of issues and it could be argued that it is much more dependent on the nature of the system being considered than some of the other quality attributes. It is recognised that not all non-funcional attributes are covered, some of which include:

- size
- shape
- weight
- battery life and power consumption
- usability

Other quality attributes related to perception, e.g. appearance and colour, are not considered.

Quality Attributes for Flexibility

Software and systems engineering has for a long time supported the notion of partitioning up designs with each part communicating with the others via well-defined interfaces [4]. To allow designs to evolve, the nature of interfaces should be made tolerant to the expected changes and variances that may occur on either side of it. For some systems, such as aircraft control systems, a build may be required for the engineers to learn about the dynamics of the system to be controlled before the control system can be properly implemented. This often leads to an iterative process. Other areas of engineering have demonstrated the importance and feasibility of flexible interfaces. For example, bridges are designed so that its sections can cope with thermal effects and amplifiers are produced so that defined tolerances on components (e.g. resistors) can be handled. For a design to work as expected, the magnitude of the possible differences in the design parameters must be known , including combinations of differences between parameters. Where possible the flexibility at the interfaces should accommodate anticipated and un-anticipated changes.

Quality Attributes for Scalability

Scalability shares many of the quality attributes discussed for flexibility. However, it also includes how enlargement of the system (anticipated or not) may be managed. There are two options:

1. *Expansion Capability* - Produce a system with spare capacity that can be used when needed.
2. *Expansion Facilities* - Produce a system such that extra facilities / capacity can be “bolted on”. Providing a facility for expansion is not considered a non-functional property and hence is not discussed further in this work .

3. BUILDING A VALUE FUNCTION TO MEASURE "QUALITY"

The method used here to measure system quality is based on timing properties and hardware usage. Properties such as memory usage could easily be incorporated as required. Other work has looked at how failure behaviour as well as timing can be accommodated [5]. Even with only considering a small number of system properties, the design is complex to produce and the associated design tensions are hard to trade-off against one another. For example, the need for a smaller size suggests fewer processing devices are used but having tight resource constraints is bound to make any system less flexible.

Timing Properties

The timing properties of a system can be split into two broad categories; essential and value added. The *essential* category contains the timing requirements that have to be met, often referred to as "hard" real-time requirements. The *value added* category contains all other timing requirements and includes "soft" requirements such as scalability.

Timing requirements exist for both individual tasks or messages and also for dependencies between tasks or messages. The requirements are independent of the computational model used. In [6], [7], [8] the set of possible timing requirements were defined as follows.

1. *Independent tasks and messages*
 (a) *period* - the rate at which a periodic task or message is to be executed at.
 (b) *deadline* - the time from release until when a task or message has to complete.
 (c) *completion jitter* - the allowed variation in task or message completion.
2. *Dependencies*
 (a) *precedence* - the order in which tasks and messages should be executed. A precedence sequence is sometimes referred to as a transaction or process chain.
 (b) *end-to-end deadline* - the allowed time from the release of the first task to the completion of the last task in a particular precedence sequence.
 (c) *separation* - the minimum time between a task or message commencing its execution from the completion of an earlier task or message in the precedence sequence.
 (d) *completion jitter* - the allowed variation in a precedence sequence completing.

For all of the commonly used computational models in real-time systems there exists timing analysis. E.g. refer to [9] for fixed priority scheduling and [10] for static scheduling. The methods used are based on the limits of execution times for tasks being available [11]. The execution time limits needed are the worst case and best case, although often the best case is taken as zero. Independent of the timing analysis used, the analysis is equally applicable to tasks or messages [12] and the following results are normally available.

- Worst-Case Response Time (WCRT)
- Best-Case Response Time (BCRT)

Based on the WCRT and BCRT, it can be determined whether all of the timing requirements are met except for the period. Period is an implementation detail and is therefore not considered part of this optimisation problem. For all deadlines it is necessary to show the WCRT is less than or equal to the deadline.

These results can be used to build value functions for both hard and soft timing requirements.

Value Function for Timing Requirements

To perform the trade-offs it is necessary to quantify how well a system meets certain parameters. This can be done by assigning a value to each of the design requirements.

For value functions to be generally effective, they should be normalised. That is, the value output should be based entirely on the quality of the solution rather than being scaled by other factors. The purpose of normalisation is to allow two different designs, potentially for different problems, to be compared without being prejudiced by either the number of elements (tasks or messages) that make up the system or the values of the properties for each element. For example, having a value function where the value equals the number of tasks that meet their deadlines is clearly influenced by the number of tasks as well as the quality of the solution. By changing the value function so that it is a ratio of the number of deadlines met to the number of deadline requirements removes the dependence on the number of tasks.

Value Function for "Essential" Timing Properties

For individual task deadlines, the value $value_{ind\_deadline}$ is calculated using equation (1).

$$value_{ind\_deadline} = \frac{D_{met}}{D_{num}} \tag{1}$$

where D_{met} is the number of tasks where the WCRT for the task is less than or equal to the deadline and D_{num} is the number of deadline requirements.

Completion jitter is calculated using equation (2). The value associated with the jitter requirements can then be calculated using equation (3),

$$jitter = WCRT - BCRT \tag{2}$$

$$value_{jitter} = \frac{J_{met}}{J_{num}} \tag{3}$$

where J_{met} is the number of tasks where the completion jitter is less than the maximum allowed completion jitter and
J_{num} is the number of jitter requirements.

A value for end to end deadlines are calculated in the same way as shown in equation (4). Methods for calculating end to end worst case response times based on release jitter have been presented in [12] and further improved with the use of offsets in [13].

$$value_{ee\_deadline} = \frac{EED_{met}}{EED_{num}} \tag{4}$$

where EED_{met} is the number of transactions with WCRT less than equal to end to end deadlines and EED_{num} is the number of end to end deadline requirements.

If separation requirements are included in the design then the value function may be formed using the same pattern as with the transaction requirements.

$$value_{sep} = \frac{sep_{met}}{sep_{num}} \tag{5}$$

where sep_{met} is the number of separation requirements met and
sep_{num} is the number of separation requirements.

An important aspect in building a value function is to sufficiently distinguish between a solution that is almost successful and one that is completely successful. (Successful is defined as when all essential requirements, e.g. timing, are met.) Therefore during earlier work an investigation was performed into the best way of doing this. A variety of methods were tried. Below are two examples.

1. *square law* - Nicholson [14] proposed a method whereby the result of the value functions are squared.
2. *step* - A bonus factor is given when a completely successful solution is found.

For our work, we are using the the step method but further investigation is needed. The reason is that the step method gives the greatest differential between all requirements being met and all but one requirement being met. To remain consistent in our approach the step is implemented with another value function component as shown in equation (6).

$$value_{step} = \begin{cases} 1 & \text{if all requirements are met,} \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

For example, in the case of all tasks meeting their deadlines, the step function becomes

$$value_{ind\_deadline\_step} = \begin{cases} 1 & \text{if } D_{met} = D_{num}, \\ 0 & \text{otherwise} \end{cases} \tag{7}$$

The weightings which are applied to value function components when they are combined (as shown later in equation (18)) can be used to change the size of the step.

Value Function for the Hardware Used

The value function for the hardware used has a number of influences on the design of systems including weight, physical locality and power (consumption and dissipation). The hardware associated with the design includes not only the microprocessors needed but also a range of other items including:

- Programmable Logic Devices (PLD)
- Memory
- Communication Controllers
- Disk Drives
- Cabling

As the work presented here is restricted to the timing domain, the main parameter considered is the microprocessor. The reason parameters such as memory are not considered within this problem is that their impact on the timing characteristics mainly causes effects at a lower abstraction level. That is, memory impacts on the tasks' WCET which are assumed inputs to this particular problem.

With respect to the microprocessors used, the key quality parameter is whether the most efficient use of hardware has been made, i.e. that there are enough processors to host the applications but not too many which would push up costs or take up too much space in the device. The ideal number of processors, P_{ideal}, can be approximated using equation (9).

$$U_{max} = \sum_{\forall i \in tasks} \frac{C_i}{T_i} \tag{8}$$

$$P_{ideal} = \lceil U_{max} \rceil \tag{9}$$

where C_i is the worst case execution time of task i and T_i is the period of task i.

A normalised value can be produced using equation (11).

$$P_{ideal\_dist} = \left| \frac{P_{num} - P_{ideal}}{P_{ideal}} \right| \tag{10}$$

$$value_{proc} = e^{-P_{ideal\_dist}} \tag{11}$$

where P_{num} is the number of processors used

The reason it is considered ideal is that the equation considers a processor loaded to 100% as being schedulable. Whilst a task set may always be schedulable with a 100% load under ideal assumptions and an optimal computational model [15], this is not usually the case, especially where the idealistic assumptions do not hold [16]. However, for the purposes of this work, the approximation is considered sufficient. The fact that there are no optimum computational models or timing analysis for realistic assumptions is further reason for supporting the use of heuristic search strategies.

Value Function for Flexibility

Tolerance to change is difficult to assess as the number of possible combinations of change makes the problem intractable.

It is, however, possible to assess by how much a limited number of parameters can be changed before the design no longer meets its requirements. An emerging area that can help with evaluating change is Scenario-Based Assessment (SBA) [17]. SBA has been used to evaluate the quality of architectural solutions that would be otherwise difficult to assess, i.e. attributes with subjective criteria or ones that are intractable to evaluate using static analysis. For a survey of work performed refer to [17]. The work to date on scenario-based assessment has concentrated on evaluating the quality of the functional aspects of architectures. In addition, to the best of our knowledge the application of the technique to evaluating change with respect to non-functional aspects of systems has not been addressed except briefly in our own work [5]. The work in [5] also shows how SBA can be used to assess the impact of failures.

To assess the ability of a design solution to cope with change, the types of change are split into two categories - anticipated and unanticipated. An example of an anticipated scenario is that a task's execution time is expected to increase by 20%. Unanticipated changes are those where the exact nature is not known, however the scale of the changes may be bounded. For example, it may be known that no task will have its WCET increased by more than 100%.

The means of assessing whether particular scenarios can be handled is to make a change and determine whether all the timing requirements are met or not using the value functions described previously. It should be noted SBA can be used to assess how close a solution is to meeting all the timing requirements if it does not already do so. In the context of timing this would entail creating scenarios which reduce the WCET and determine the number of timing requirements that are met. However this is outside the scope of this work.

The value function for the SBA is given in equation (12). The equation sums the value calculated with respect to how well the timing requirements are met for each scenario generated.

$$value_{scen} = \frac{\sum_{s \in scen} value_s}{N_{scen}} \tag{12}$$

where $scen$ is the set of scenarios being assessed,
N_{scen} is the number of scenarios, and
$value_s$ is derived from equations (1) to (6),
e.g. by summing the value components.

Value Function for Scalability

As well as using SBA to sample the ability to cope with change, it is also useful to be able to assess the extremes of change. For this reason sensitivity analysis [18] is used to determine individual task sensitivity and whole task set sensitivity.

Individual Task Sensitivity—For each task determine by how much its WCET can increase before the timing requirements are no longer met. The value function can be calculated using equation (16). The value function must be normalised against the size of the WCET and secondly against the number of tasks.

For each task, a value C_INC_i is calculated which is the amount the WCET of the task can increase before the system becomes unschedulable. If the system is already unschedulable, C_INC_i will be a negative value representing the amount the WCET must decrease before the system is schedulable. If decreasing the WCET of a task does not make the system schedulable (e.g. reducing the WCET of a low priority task will not make a higher priority task schedulable), then C_INC_i is undefined.

Assuming C_INC_i is defined, let

$$s_i = -\left(\frac{C\_INC_i}{C_i} + 1\right) \tag{13}$$

where C_i is the WCET of task i. As $C\_INC_i \geq -C_i$, $s_i \leq 0$. This leads to the sensitivity value for a single task being defined as

$$sens_i = 1 - e^{s_i} \tag{14}$$

If C_INC_i is undefined, it means modifying the WCET for task i will not make the system schedulable. Having several tasks which cannot be modified to make an unschedulable system schedulable is not desirable and hence the full definition for $sens_i$ is

$$sens_i = \begin{cases} 1 - e^{s_i} & \text{if } C\_INC_i \text{ is defined,} \\ 0 & \text{otherwise.} \end{cases} \tag{15}$$

The over all individual task sensitivity value is then

$$value_{sens} = \frac{\sum_i sens_i}{N_{tasks}} \tag{16}$$

where N_{tasks} is the number of tasks.

As well as having the average sensitivity for the individual tasks, it is also important to know the minimum sensitivity. The reason is as well as having a high average scalability it is also useful for a design to not be particularly sensitive in certain areas. That is, no individual task should be particularly sensitive to change. For this reason the value function, expressed in equation (17), also has a value for the minimum individual sensitivity.

$$value_{sens\_min} = \min_i sens_i \tag{17}$$

Whole Task Set Sensitivity— For a whole task set, it is determined by how much all the tasks' WCET can uniformly increase before the timing requirements are no longer met, represented by the value r_i for task set i. For example if all WCETs in a task set may increase by 10%, $r_i = 1.1$.

If all WCETs need to decrease by 10% before the system is schedulable, $r_i = 0.9$. By redefining s_i in equation (13) as $s_i = -r_i$, the same method of calculating a whole task set sensitivity value as the individual task sensitivity value, replacing the number of tasks with the number of task sets as appropriate.

Using the scalability analysis on unschedulable task sets has been found to be particularly useful to distinguish between two designs that don't meet their requirements. It can be used to guide the search towards a schedulable solution.

The means of searching for the amount by which task(s)' WCETs can be increased on decreased is a classical search problem. Currently a binary search is used. For each value examined, the timing analysis is performed to determine whether the timing requirements are met.

Combining the Value Functions

Given the individual results of the value functions, these have to be combined into a single result. This is achieved using equation (18).

$$overall\_value = \frac{\sum_{\forall i \in value\_fns} value_i \cdot weight_i}{\sum_i weight_i} \quad (18)$$

where $value\_fns$ is the set of value functions,
$value_i$ is the result of value function i, and
$weight_i$ is the weight applied to value function i.

The equation features a summation over all value functions with a weighting applied to each value function. The purpose of the weighting is to ensure appropriate balances are achieved for the design trade-offs and to help improve search efficiency. As the resulting values from this function are used to influence the simulated annealing search described later, it is necessary to normalise the value against the weightings used. This means that only the values of the weightings relative to each other affect the search rather than the weightings themselves.

One aspect shown to be effective in finding the best solution (i.e. reducing the search time and attaining the best resulting design) is to use different weightings when all the timing requirements are met. For instance, by making the number of processors used with respect to the ideal less important when the timing requirements are not all met means the searching can use more processors and make it more likely the requirements are met. Then, once the requirements are met the design can be fine tuned.

4. TURNING THE VALUE FUNCTION INTO AN ASSESSMENT FRAMEWORK

Tool Architecture

Figure 1 presents the basic architecture of the optimisation framework. The key issue is the only part of the framework that is technology dependent is the computational model which in this case is how tasks are scheduled and executed on the chosen platform. The architecture chosen maintains a clear separation between the schedulability analysis performed, the calculation of the value function (including static analysis, sensitivity analysis and scenario-based assessment) and the search algorithm used as part of optimisation. Within the scenario-based assessment, facilities are provided for the use of pre-defined scenarios and scenarion which have been randomly generated within defined bounds.

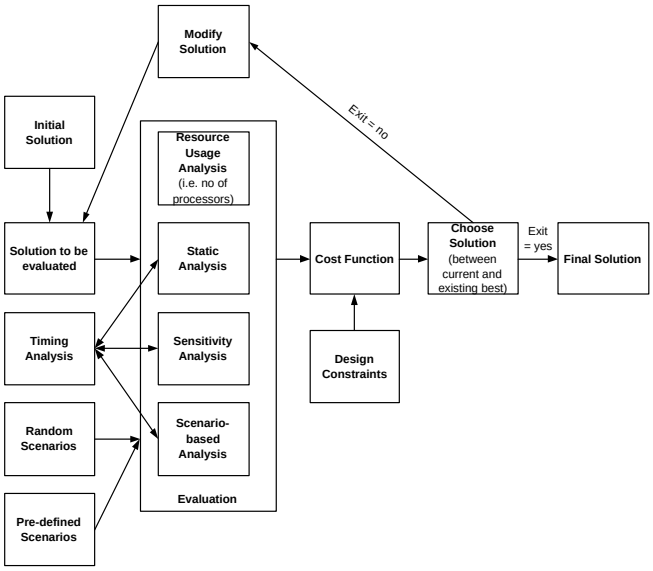

Figure 1. Architecture of the Assessment Tool

Search Algorithm

For the purposes of this paper a heuristic search technique, simulated annealing, is used in order to explore the design space and trade-off the different design tensions that are present in the problem space. The reason for choosing simulated annealing [19] is based on it having previously been demonstrated as effective for this class of problem [14]. Other work we have performed has explored a range of heuristic algorithms such as genetic algorithm, memetic algorithms and nested annealing [19]. Simulated annealing has been found to provide good performance while being relatively simple to implement. Future work will examine whether the other algorithms can be tuned so they are more effective. The simulated annealing algorithm used in our work can be described by the pseudo-code in figure 2

Design Choices

In our previous work [5], [20], the design choices for this problem were established. The choices available were split into three categories. Those applicable to individual tasks, those applicable to messages and those applicable to the system architecture. For some systems part or the whole of the system architecture may be a design invariant and in these

```
set t to initial temperature
select new model as random starting model
loop
  num_moves = 0
  loop
    move to new model
    increment num_moves
    calculate overall_value using value functions
    if value is highest value found so far
      best_model = current_model
      moves_since_improvement = 0
    else
      increment moves_since_improvement
    endif
    if value is higher than previous value
      adopt new model
    else
      produce random probability
      d = previous value - current value
      if probability < exp(-d/t)
        adopt new model
      endif
    endif
    select new model
  while num_moves < M and moves_since_improvement < N
  reduce t
while moves_since_improvement < N
```

Figure 2. Simulated annealing algorithm used to search design space

cases the changes that can be made to its design would have to be restricted. The choices are:

1. Individual Tasks
   (a) *Deadline*
   (b) *Ordering* - this can mean priority in priority scheduling schemes or slot position in static scheduling
   (c) *Offset* - the time of task's first release relative to a fixed point in time
   (d) *Release jitter* - the maximum variation in when the task is released
   (e) *Allocation* - the processor on which a task executes
2. Messages - messages exist where tasks are linked by an end-to-end deadline. They be sent on a network between processors or a bus within a processor.
   (a) *Deadline*
   (b) *Ordering*
   (c) *Offset*
   (d) *Allocation* - allocation to a network if more than one possible communications method exists.
3. System Architecture
   (a) *Addition or removal of additional processors*

Derivation of Weightings

The purposes of the weightings are two fold. Firstly to obtain the best possible design by biasing the different design tensions. For example ensuring meeting all timing requirements is more important than supporting change. Secondly, to improve the ability and efficiency of searching. For example, by having too great a weighting for meeting all timing requirements then the search algorithm may not be able to move between two areas of the design space through a region where the timing requirements are not all met. This could prevent a design with better properties, e.g. ability to cope with more changes, being found. The problem of assigning weights is widely recognised as a significant problem [14], [19]. To date our approach has derived the values by trial and error but it is recognised further research should be performed into finding a principled method.

5. EVALUATION

The evaluation presented in this section is intended to show how the framework uses a set of requirements and an estimated WCET to generate what it considers the best solution. Note that not all value function components such as scenario based analysis and completion jitter requirements are included in this evaluation. To demonstrate the way the framework operates in the available space, a small example with a few tasks is chosen. However, the approach is equally applicable to large-scale systems. In addition, other work, including [5], has shown heuristic search algorithms can handle the scalability to allocating tasks for large systems.

For the purposes of the example, the fixed priority scheduling approach is used though the framework has been applied to other approaches. For the example presented, it is assumed there are no kernel overheads or blocking. Timing analysis for fixed priority scheduling is presented in section 7. In this approach all tasks and messages are given a static priority off-line and then at run-time the priority is used to decide which task (i.e. the highest priority one) should be executed at any particular time. The priorities are assigned randomly. These priorities are then adapted during the execution of the heuristic search algorithm.

Requirements of the Example

The example consists of 18 tasks forming 3 software applications each of which may be decomposed into a number of transactions. Table 1 gives the individual task requirements and WCETs. Tasks 1A to 1E belong to application 1, tasks 2A to 2F belong to application 2 and tasks 3A to 3G make up application 3.

Messages which are sent between tasks are shown in figure 3. Messages must be allocated to a communications medium which connects the processors which the sending and receiving tasks are allocated to. Messages are assumed to have a fixed size and the execution times for messages are dependent upon the speed of the network. For this example there is a single network connecting all processors for sending messages between processors. Sending a message on this network takes 93ms. If both tasks are located on the same processor, the message can be sent on a communications bus which is assumed to take 1ms. Messages have the same period as that of the sending task. Messages are scheduled using the same fixed priority scheduling analysis as used on the tasks as pro-

Id	T	D	WCET
1A	500	500	180
1B	1000	750	150
1C	1000	1000	75
1D	1000	1000	140
1E	1000	1000	75
2A	2000	2000	200
2B	2000	2000	200
2C	2000	2000	200
2D	1750	1500	275
2E	2000	2000	225
2F	2000	1500	200
3A	2000	2000	200
3B	2000	2000	150
3C	2000	1500	200
3D	2000	2000	200
3E	2000	2000	225
3F	2000	2000	200
3G	2000	2000	200

Table 1. Individual task timing requirements

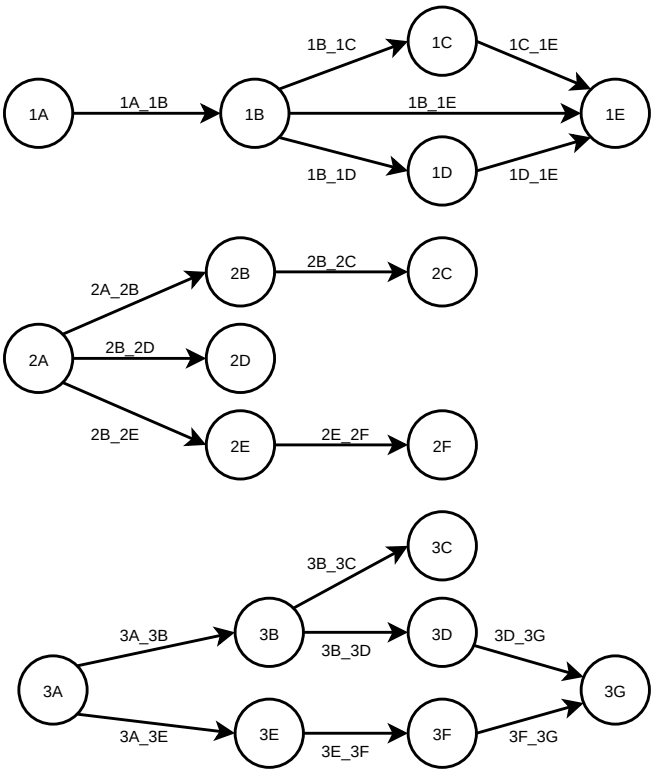

Figure 3. Messages sent between tasks

posed in [21]. Like tasks, messages are assumed to be able to preempt each other which is not realistic. However, the framework has been designed to allow different scheduling models to be used for scheduling tasks and messages. It also allows different scheduling models to be applied to different processors or networks within the system. Future evaluations will exploit this.

All transactions within application 1 must complete within 1000ms. Transactions within applications 2 and 3 may take up to 2000ms. Transaction response times are calculated based on the holistic scheduling release jitter method in [12]. This introduces a delay in the form of release jitter to take account of preceding tasks and messages. In simple cases, the release jitter is equal to the maximum WCRT of preceding tasks / messages. Some additions have been made to the method to reduce pessimism when two tasks are allocated to the same processor. If a higher priority task is sending a message to a lower priority task then the lower priority task will already be delayed by interference from the higher priority task. The lower priority task only needs to have an additional delay (release jitter) to take account of time required to send the message. If the message is being sent from a task on a different processor the release jitter must take account of the execution of the sending task as well as the time required to send the message. In the longer term, a more general approach to holistic scheduling [13] will be taken to remove restrictions such as deadlines must be less than or equal to period.

Table 2 shows the weightings used to produce the results. These weightings are intended to balance scalability with hardware usage.

Value Function	Weighting
Schedulable tasks	2000
Schedulable messages	2000
Schedulable processors	200
Schedulable communications	200
Schedulable system	500
Processors used	100
Task sensitivity	20
Processor sensitivity	20
Communications sensitivity	20

Table 2. Weightings used in framework

Static Timing Analysis Results

The results of the task allocation and attribute assignment are shown in table 3. RJ is the release jitter due to task communication and R is the worst case response time. P is the priority assigned to each task. A shows where the task is allocated to. C_MAX represents the largest possible value for the worst case execution time of a task before the schedulability of the system is broken, assuming all other execution times remain the same. Figures in the $SCAL$ column show the allowed scaling of task execution time as a percentage derived from C and C_MAX.

Table 4 shows the results of the message allocation. In the allocation column, *Net* indicates the message was allocated to the network connecting the processors and *Bus X* indicates the message was allocated to the communications bus for processor X. The communications bus for a processor is used to send messages between tasks allocated to the same processor. Scalability information is also available for messages but these results are not considered in this evaluation. Table 5 shows the utilisation and scalability of each processor using

Id	T	C	RJ	R	P	A	C_MAX	SCAL
2B	2000	200	665	865	6	1	530	165
2C	2000	200	758	1158	11	1	530	165
3D	2000	200	854	1454	12	1	530	165
3G	2000	200	870	1670	13	1	530	165
3A	2000	200	0	200	2	2	530	165
3E	2000	225	1	426	4	2	555	147
3B	2000	150	186	761	15	2	480	220
3F	2000	200	2	777	16	2	530	165
2A	2000	200	0	200	1	3	642	221
2E	2000	225	2	427	3	3	822	265
2D	1750	275	1	701	7	3	872	217
2F	2000	200	3	903	14	3	797	298
1A	500	180	0	180	8	4	275	53
1B	1000	150	3	333	9	4	341	127
1C	1000	75	5	410	10	4	266	255
1D	1000	140	5	730	17	4	331	136
1E	1000	75	9	809	18	4	266	255

Table 3. Results of task allocation and attribute assignment

Id	T	C	RJ	R	P	A
3F_3G	2000	93	777	870	4	Net
3A_3B	2000	93	200	386	7	Net
3B_3D	2000	93	575	854	9	Net
3B_3C	2000	93	575	947	12	Net
2A_2B	2000	93	200	665	15	Net
2B_2C	2000	93	400	958	16	Net
3D_3G	2000	1	1454	1455	14	Bus 1
3A_3E	2000	1	200	201	3	Bus 2
3E_3F	2000	1	425	427	11	Bus 2
2A_2D	2000	1	200	201	8	Bus 3
2A_2E	2000	1	200	202	10	Bus 3
2E_2F	2000	1	425	428	17	Bus 3
1C_1E	1000	1	410	411	1	Bus 5
1B_1D	1000	1	333	335	2	Bus 5
1A_1B	500	1	180	183	5	Bus 5
1B_1E	1000	1	330	334	6	Bus 5
1B_1C	1000	1	330	335	13	Bus 5
1D_1E	1000	1	728	734	18	Bus 5

Table 4. Results of message allocation

the weightings in table 2.

A second evaluation was performed with the weights adjusted to minimise the number of processors used without considering scalability. These weightings are shown in table 6 and the usage of each processor is shown in table 7. Whether the solution with fewer processors is more desirable than the more scalable solution is dependent upon the requirements of the system.

A third set of weightings, shown in table 8 was used to maximise scalability. This solution again used 5 processors as in the first evaluation but managed to schedule the system to achieve higher mean processor scalability as shown in table 9 although the minimum scalability was not much improved.

Processor	Utilisation	Scalability
1	40.00%	41.75%
2	38.75%	42.76%
3	46.96%	66.56%
4	10.00%	176.78%
5	80.00%	23.96%

Table 5. Processor utilisation and scalability

Value Function	Weighting
Schedulable tasks	2000
Schedulable messages	2000
Schedulable processors	200
Schedulable communications	200
Schedulable system	500
Processors used	300
Task sensitivity	10
Processor sensitivity	10
Communications sensitivity	10

Table 6. Weightings used to minimise hardware usage (evaluation 2)

Processor	Utilisation	Scalability
1	66.96%	35.24%
2	80.00%	24.65%
3	68.75%	45.07%

Table 7. Processor utilisation and scalability results for evaluation 2

Value Function	Weighting
Schedulable tasks	2000
Schedulable messages	2000
Schedulable processors	200
Schedulable communications	200
Schedulable system	500
Processors used	20
Task sensitivity	50
Processor sensitivity	50
Communications sensitivity	50

Table 8. Weightings used to maximise scalability (evaluation 3)

Processor	Utilisation	Scalability
1	21.25%	162.17%
2	80.00%	24.65%
3	30.00%	121.73%
4	38.75%	93.60%
5	45.71%	99.62%

Table 9. Processor utilisation and scalability results for evaluation 3

Sensitivity Analysis

For the first evaluation which attempted to balance hardware usage with scalability, individual task scalability ranged from 53% for task 1A to 298% for task 2F.

In evaluation 2, the task with least scalability was again task 1A with 54% but task 3B achieved a scalability of 412%. In fact, many tasks were individually more scalable than the first evaluation although the overall processor scalability values presented in table 7 were lower. An explanation for this is that by forcing the search to fit the tasks onto fewer processors also forced it to find a more efficient scheduling solution.

The scalability of tasks in evaluation 3 ranged from 54% to 481% with a mean value of 368%. This is a large improvement over the first evaluation which also used 5 processors. It is noted that the scalability of processor 2 in table 9 matches that of processor 2 in table 7. In both instances the tool chose to place the whole of application 1 on a single processor. Splitting up the tasks of this application would increase the communications overheads and so this proved to be a good solution in both cases.

These results indicate that the tool produces a better solution with a definite aim of optimising one particular quality attribute as opposed to balancing more than one. This could be due to the fact that, in the first evaluation, no great importance was given to either quality attribute over schedulability so the search stopped soon after finding a schedulable solution.

The scalability of tasks varies greatly throughout the design. It would be useful to include additional information such as the probability that a particular task will change within the design. This would allow the sensitivity analysis to be targeted more appropriately.

6. CONCLUSIONS

During the course of this paper we have presented a number of non-functional quality attributes which may be considered for real time embedded systems design. Using timing requirements as an example, we have shown how a value function may be developed to quantify a quality attribute and how it may be normalised to allow different designs to be compared using the value function.

We have developed a framework which uses the value function within a heuristic search environment to optimise the design for different requirements. An example was presented which evaluated maximising the scalability and minimising the number of processors of a design. When attempting to maximise scalability, the scalability of individual tasks varied greatly and hence further work needs to be done on how sensitivity analysis can be targeted to smaller sections of the design. The framework was successful at trading off processor utilisation against scalability and reduced the number of processors used from 5 to 3 when the weightings were biased towards reducing the number of processors.

It has been shown how scenario based assessment may be incorporated into the value function but an evaluation is yet to be done. Further work is required in assessing how to create scenarios and where in the design to apply them. Some prototype implementations are being tested and will be the subject of future research.

7. APPENDIX

Timing Analysis for the Fixed Priority Scheduling Model

The standard timing analysis for each individual processor is solved using equation (A.1) which is taken from Harter [9]. The analysis is valid for task sets with an unique critical instant. Harter's analysis assumes there are a fixed number of tasks, all of which have a fixed unique priority, zero offset, and the deadlines are not greater than the period.

$$R_i = C_i + B_i + I_i \tag{A.1}$$

where i is a task in the set of tasks for a given node
R_i is the WCRT of task i
C_i is the worst-case execution time of task i
B_i is the worst-case blocking time suffered by task i
I_i is the worst-case interference suffered by task i

The blocking time, B_i, is the longest time that a lower priority task can prevent task i when it is runnable. The blocking time is dependent on the computational model that is being used. In an idealised preemptive model, the blocking time should be zero. However cases exist, particularly with shared resources, where some blocking may need to be accounted for.

The interference a task suffers is the maximum utilisation from the critical instant for its higher priority tasks before it executes for the first time. The interference is calculated using equation (A.2) which represents the sum of the utilisations over the duration of interest for all the higher priority tasks than task i. The utilisation is the product of the number of times the task can execute and its worst-case execution time. The number of times a higher priority task can execute is found by rounding up the result of the time during which interference may occur (i.e. the response time of the task being analysed) divided by the period of the higher priority task.

$$I_i = \sum_{j \in hp(i)} \left\lceil \frac{R_i + J_j}{T_j} \right\rceil C_j \tag{A.2}$$

where $hp(i)$ is the set of higher priority tasks than task i
J_i is release jitter introduced waiting for preceding tasks or messages

Equation (A.1) is solved by forming a recurrence equation as shown in equation (A.3).

$$r_i^{n+1} = C_i + B_i + \sum_{j \in hp(i)} \left\lceil \frac{r_i^n + J_j}{T_j} \right\rceil C_j \qquad \text{(A.3)}$$

with $r_i^0 = C_i$
which terminates when $r_i^{n+1} = r_i^n$, or $r_i^{n+1} + J_i > D_i$.
where D_i is the deadline of task i, and
T_i is the period of task i.
J_i is the release jitter based on R_j for $j \in pre(i)$ and
$pre(i)$ is the set of tasks preceding task i.

To obtain the final value for the WCRT, r_i must be combined with the possible delay from previous tasks at the point at which the analysis converges or when the worst-case response time exceeds the task's deadline.

$$R_i = r_i + J_i \qquad \text{(A.4)}$$

REFERENCES

[1] Y. Yeh, "Dependability of the 777 primary flight control system," *5th IFIP Working Conference on Dependable Computing for Critical Applications*, 1995.

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

[3] J. Laprie, J. Arlat, C. Beounes, K. Kanoun, and C. Hourtelle, "Hardware and software fault-tolerance: Definition and analysis of architectural solutions," *7th Annual International Symposium Fault-Tolerant Computing*, pp. 116–121, 1987.

[4] C. Jones, "Specification and design of (parallel) programs," in *Proceedings of IFIP Information Processing*, pp. 321–332, 1983.

[5] I. Bate and N. Audsley, "Flexible design of complex high-integrity systems using trade offs," in *8th IEEE International Symposium on High Assurance Systems Engineering*, pp. 22–31, 2004.

[6] I. Bate and A. Burns, "An integrated approach to scheduling in safety-critical embedded control systems," *Real-Time Systems Journal*, vol. 25, pp. 5–37, Jul 2003.

[7] M. Torngren, "Fundamentals of implementing real-time control applications in distributed computer systems," *Real-Time Systems*, vol. 14, pp. 219–250, May 1998.

[8] K. Sandstrm and C. Norstrm, "Managing complex temporal requirements in real-time control systems," in *Proceedings of 19th IEEE Conference on Engineering of Computer-Based Systems*, pp. 103–109, 2002.

[9] J. Harter, "Response times in level-structured systems," *ACM Trans. Computer Systems*, vol. 5, pp. 232–248, Aug. 1987. ACM Transactions on Computer Systems.

[10] G. Fohler and C. Koza, "Heuristic scheduling for distributed real-time systems," Tech. Rep. Research Report No. 6, Institut fur technische Informatik, Technische Universtate Wien, Austria, 1989.

[11] P. Puschner and C. Koza, "Calculating the maximum time of real-time programs," *Real-Time Systems*, vol. 1, no. 2, pp. 159–176, 1989.

[12] J. A. Clark and K. Tindell, "Holistic schedulability analysis for distributed hard real time systems," *Microprocessing & Microprogramming*, vol. 50, pp. 117–134, April 1994.

[13] J. Gutierrez, J. Garcia, and M. Harbour, "On the schedulability analysis for distributed real-time systems," in *Proceedings of the 9th Euromicro Workshop on Real-Time Systems*, pp. 136–143, 1997.

[14] A. Burns, M. Nicholson, K. Tindell, and N. Zhang, *Allocating And Scheduling Hard Real-Time Tasks On A Parallel Processing Platform YCS 238*. Dept of Computer Science, University of York, October 1994.

[15] J. Stankovic, M. Spuri, K. Ramamritham, and G. Buttazzo, *Deadline Scheduling For Real-Time Systems: EDF and Related Algorithms*. Kluwer Academic Publishers, 1998.

[16] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard real-time environment," *J. ACM*, vol. 20, no. 1, pp. 40–61, 1973.

[17] L. Dobrica and E. Niemel, "A survey on software architecture analysis methods," *IEEE Transactions on Software Engineering*, vol. 28, pp. 638–653, July 2002.

[18] A. Burns, S. Punnekkat, L. Stringini, and D. Wright, "Probabilistic scheduling guarantees for fault-tolerant real-time systems," in *Proceedings of the 7th IEEE International Working Conference on Dependable Computing for Critical Applications*, pp. 361 – 378, 1999.

[19] V. Rayward-Smith, I. Osman, C. Reeves, and G. Smith, eds., *Modern Heuristic Search Methods*. Wiley, 1996.

[20] I. Bate and N. Audsley, "Architecture trade-off analysis and codesign for safety-related real-time embedded systems," in *Proceedings of 1st International Workshop on Embedded Systems Codesign*, pp. 8–15, 2002.

[21] K. Tindell, A. Burns, and A. J. Wellings, "Analysis of hard real-time communications," *Real-Time Systems*, vol. 9, no. 2, pp. 147–171, 1995.

Dr Iain Bate *is a lecturer in Real-Time Systems within the Department of Computer Science at the University of York. Prior to this, he has been a researcher for ten years within the department based in both the BAE SYSTEMS and Rolls-Royce research centres. His research interests include scheduling and timing analysis, systems engineering including optimisation (design trade-offs) and design assurance. He is also the editor of the journal "Microprocessors and Microsystems" and a director of Origin Consulting (York) Ltd who provide advice to the dependable real-time systems industry.*

Paul Emberson *is a Research Associate at the University of York, U.K. His research interests include scenario based analysis and real time systems. He obtained an MMath degree in Mathematics and Computer Science from the University of York in 2002. He previously worked in industry in the fields of software testing and mobile telecommunications.*