Incorporating Scenarios And Heuristics To Improve Flexibility In Real-Time Embedded Systems

Iain Bate, Paul Emberson
Department of Computer Science
University of York
York, YO10 5DD
{iain.bate,paul.emberson}@cs.york.ac.uk*

Abstract

Flexibility, the ability to adapt to change, is important for real-time systems. As in any type of system, changes arise from maintenance, enhancements and upgrades. These changes are only feasible if timing requirements imposed by the real-time nature of the system can still be met. A flexible design will allow tasks to be added without impinging on other tasks, causing them to miss deadlines. The design space for these systems consists of many configurations describing how tasks and messages are allocated to hardware and scheduled on a hardware platform.

Heuristic search is a well recognised strategy for solving allocation and scheduling problems but most research is limited to finding any valid solution for a current set of requirements. The technique proposed here incorporates scenario based analysis into heuristic search strategies where the ability of a solution to meet a scenario is included as another heuristic for the changeability of a system. This allows future requirements to be taken into account when choosing a solution so that future changes can be accommodated with minimal alterations to the existing system.

1 Introduction

It is widely acknowledged that obtaining a set of consistent and stable requirements is difficult [16]. Therefore, designing to cope with change has become increasingly important as complexity increases. Changes to a poorly designed system are likely to lead to ripple effects where the cost of the change is disproportionate to its size and may even outweigh the benefits received from the change. Ideally, both the size and cost of change should be comparable to the resulting effect. Extreme examples of where a very small change to the implementation of a system can result in high costs are often found in the avionics domain. Other than making the change, costs arise from testing, verification and certification following the change. These activities may require the involvement of different bodies and contractors, hence increasing costs further through contractual and administrative matters. This is also true in any industry where sub-systems are developed separately by more than one company. For instance, early in the design of automotive systems, individual sub-systems such as braking, infotainment, etc. are contracted out to other companies. The contract details often define the hardware available to each supplier as well as details of access to shared communications networks such as message priorities. If changes to these details are needed then a contract re-negotiation is necessary which may have knock-on effects to other suppliers. These matters increase not only the expense but also the time required to make the change.

One approach to handling changes is to build in ample spare capacity. However, this is prohibitively expensive due to the cost of the extra hardware that might be needed. In a production run of 2 million cars, adding one extra processing resource at a unit cost of \$10 would cost \$20 million before any additional costs for extra cabling, housings, cooling, etc. are accounted for. Spare capacity in the system does not necessarily mean it is usable. For example. it may be physically separated from where it is eventually needed.

1.1 Designing For Change

A major difficulty in designing for change is being able to predict the changes which will be required. However, there are situations where this is feasible. It is common for large systems to use a phased development program. In this instance, future phases should be carefully planned with documented future requirements. In reality, low level changes for an anticipated future requirement, such as increases to task execution times, cannot be fully known without actually implementing the change. In this sense changes are rarely 100% predictable but rather estimated to within

*This work has been funded by BAE Systems via the Dependable Computing Systems Centre (DCSC).

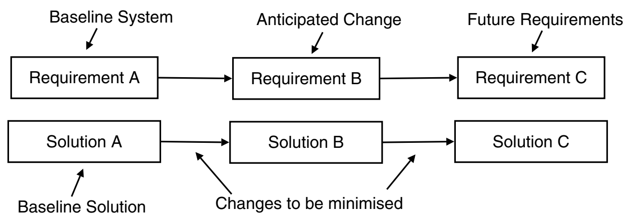

Figure 1. Upgrade paths.

some degree of correctness. Clearly, the more accurately changes can be predicted, the less impact they will have.

The method outlined in this paper uses anticipated change scenarios to select a design for a system under development that will allow the eventual upgrade described in the scenario to be realised with fewer changes to the design than if no scenarios were used. In addition, there is an investigation of how using scenarios for anticipated changes increases system flexibility for unanticipated changes. The chosen design for the current system requirements is referred to as the baseline design, shown as the baseline solution in figure 1. In addition to creating a flexible baseline design, metrics are used which measure the change between two solutions which should correlate with the cost of an upgrade to meet new requirements. This enables us to select an upgrade that requires the fewest and cheapest changes. A longer term aim is to consider how minimising the cost of an upgrade trades off with the flexibility of the new baseline being created.

The structure of the paper is as follows. The next section describes the task allocation and scheduling problem, the solution to which forms the basis of this work, and the related work that has previously addressed the problem. Section 3 discusses how scenarios have previously been used to promote flexibility within designs. As part of this section the contributions of this paper, the first use of scenarios as part of an automated search environment and improving the flexibility of task allocations and schedules, are justified. Then, the design of the framework is explained and justified in section 4. Section 5 presents the evaluation that demonstrates the approach and shows that the use of scenarios as part of an automated search environment can in fact improve the flexibility of systems both when the changes are close to those anticipated but also when they are unanticipated. Finally the conclusions are given.

2 Architecture Selection Problem

The problem of selecting an architecture that will withstand the requirements placed upon the system through its lifetime challenges all system designers. Methods exist to analyse the quality of an architecture from a model [8, 2]. Flexibility is often considered in terms of other metrics such as maintainability or module dependencies [10]. Whilst it is possible to efficiently assess the quality of a given architecture, producing an architecture to meet particular quality requirements is difficult.

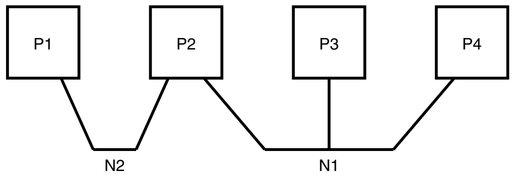

Figure 2. An example hardware platform of processors and networks.

This work will limit architecture selection for real-time systems to the problem of how to distribute tasks between hardware resources and how to schedule the tasks once allocated. Both allocation and scheduling are known to be NP-hard [17]. The architecture model used is as follows. Software applications are assumed to consist of chains of communicating tasks, called *transactions*. Both the tasks and messages sent between tasks must be allocated to a hardware platform. Tasks are assigned to processors and messages are assigned to networks.

The example platform in figure 2 shows it is not necessary for all processors to be directly connected to each other so the availability of networks for message allocation is dependent upon where the sending and receiving are allocated. It is assumed that each processor also has a means of passing messages between a pair of communicating tasks both allocated to that processor. Once allocated, all messages and tasks must be scheduled according to some scheme. Most scheduling algorithms assign some value to each object which influences the order in which tasks are run. All the investigation and evaluation performed here is based upon distributed fixed priority pre-emptive scheduling [5, 12]. In this model, each task and message is assigned a unique priority and at any time the highest priority task that is ready to or already running is the one that it is executing. A priority and allocation pair for an object is the *configuration* of that object. The configuration of all objects taken together is the configuration of the system.

Each task or message also has a set of timing properties and timing requirements. For a task, these are:

- Worst case execution time (WCET) - the maximum time a task will take to execute.
- Best case execution time (BCET) - the minimum time a task will take to execute.
- Period - the time between each arrival of a task. (Aperiodic and sporadic tasks are not considered).
- Deadline - the time by which a task must complete since its arrival.

User Input				Generated Input	
Task	WCET	Period	Deadline	Alloc.	Priority
τ_1	C_1	T_1	D_1	A_1	P_1
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
τ_n	C_n	T_n	D_n	A_n	P_n

Table 1. Input data required to perform distributed scheduling analysis.

For a message, the timing properties and requirements are:

- Maximum size - the maximum size of a message. The worst case communication time (WCCT) of a message is derived from this depending on the speed of the network the message is allocated to.
- Minimum size - the minimum size of a message. This is used to calculate the best case communication time (BCCT) of a message.
- Deadline - the time by which the message must reach its destination.

The period of a message is inherited from the sending task. In addition to timing properties, the sending and receiving task must also be specified for each message.

This information for a task or message can be used along with its configuration to perform scheduling analysis which will ascertain whether all deadlines will be met. The scheduling analysis used is the dynamic offset approach described in [12]. To handle transactions efficiently an event triggered model is used so that a message is sent as soon as possible after the sending task completes and a task executes as soon as possible after a message arrives. When a task or message must wait for an incoming event, the delay is modeled through the use of jitter. The results from the scheduling analysis give the values for release jitter along with worst case response times for each task and message. If all response times are less than the deadline for the corresponding object, then the timing requirements for the system have been met.

The basis of the task allocation and scheduling problem is therefore to generate a configuration for each task (and message) so that all timing requirements are met. This is summarised in table 1. The columns to the left of the table are the timing requirements. The right hand two columns make up the configuration data which forms the solution to the requirements.

While a solution to this problem has been looked at by many authors [13, 17, 3, 1, 19, 18], little work has been done on considering the flexibility of a system based upon the results of scheduling analysis. For example, if a processor has particularly low utilisation, it suggests that there is scope to add more tasks to the system on that processor. The *scalability* of a task refers to its ability to increase its execution time (or size for a message) without breaking any timing requirements. This concept has been considered previously by Regehr [15]. Attempting to make certain tasks scalable which are thought to be susceptible to change will give one solution preference over another. This work is complementary to [15] in that it considers how to select which tasks to make scalable and how to make use of more flexible execution environments by considering more precise potential changes.

3 Scenarios

A scenario is a textual description of a possible requirement for a system. They may be qualitative as in "how secure is the system against a particular type of attack" or quantitative as in "how much will response times increase if the number of expected users increases by a factor of 10". For this to be most useful it is necessary to quantify the non-functional attributes from an architectural model. This is easier for some qualities, such as timing and memory usage, than for more subjective ones such as security.

Correctly predicting scenarios allows design to have flexibility built into the appropriate parts of the system. A problem arises when a subsystem that was predicted to be relatively stable requires a large change. Without infinite resource, scalability must be traded off between subsystems hence a subsystem incorrectly determined to be stable may be difficult to change. This leads to the general problem of how to generate realistic scenarios for a system and how they should be prioritised [8].

There are well established techniques such as SAAM [7] and ATAM [8, 6] which are manual processes for creating and using scenarios. However, they only allow for a very small number of iterations of the architecture due to the time taken for the process. For more details on a wide range of scenario-based techniques proposed, refer to [4]. Many of these methods consider multiple quality attributes of which flexibility is just one. It must be recognised that one quality cannot be considered in isolation. Increasing flexibility may reduce performance. A common example is adding additional layers in a layered architecture.

Based on the available literature surveyed in this section and the previous one, it is believed that our work is the first to consider

1. the use of scenarios as part of an automated design framework
2. the use of scenarios to build flexibility into the schedules of systems

4 Architecture Selection Framework

Heuristic search methods are used to find good approximations to optimisation problems with exponentially large

```
$\Omega = \{\omega_0, \ldots, \omega_N\}$              /* Solution space */
$f : \Omega \rightarrow [0, 1]$                    /* Cost function */
$\eta : \Omega \rightarrow \Gamma$ where $\Gamma \subset \Omega$     /* Neighbourhood */
$\psi \in \Omega$                            /* Initial configuration */
$\omega^* = \psi$                           /* Best configuration */
$\omega = \psi$                             /* Current configuration */
$t = t_0$                              /* Set initial temperature */
$\alpha = 0.99$                           /* Cooling factor */
do
    $i = 0$
    do
        select new $\omega'$ s.t. $\omega' \in \eta(\omega)$
        $\delta = f(\omega') - f(\omega)$
        $R$ = random value $\in [0, 1]$
        if ($R < e^{-\frac{\delta}{t}}$) then $\omega = \omega'$ endif
        if ($f(\omega) \leq f(\omega^*)$) then $\omega^* = \omega$ endif
        $i = i + 1$
    until ($i = M$ or (stopping condition))
    $t = \alpha t$
until (stopping condition)
```

Figure 3. Simulated annealing

```
$r$ = random value $\in \{0, 1\}$
if ($r = 0$) then
    $object\_type = task$
else
    $object\_type = message$
endif
$obj$ = random object of type $object\_type$
$r$ = random value $\in \{0, 1, 2\}$
if ($r = 0$) then          /* do priority swap */
    $swap\_obj$ = random object with
                        same allocation as $obj$
    $tmp = obj.priority$
    $obj.priority = swap\_obj.priority$
    $swap\_obj.priority = tmp$
else                       /* do allocation change */
    if ($object\_type = task$)
        $scheduler$ = random processor
                        other than allocation of $obj$
    else
        $scheduler$ = random network
                        other than allocation of $obj$
    endif
    change allocation of $obj$ to $scheduler$
endif
```

Figure 4. Neighbourhood function

search spaces. They take steps with the aim of optimising an objective function. In this instance, the objective function is a measure of the quality of architecture candidates. All such search methods must make a trade-off between converging quickly to a local optima and allowing local optima to be escaped in order to move towards a global optima. Whilst there are a number of such strategies, the tool referred to in this paper and used in the evaluation uses simulated annealing [9]. This has been shown to effective at solving allocation style problems [17, 14, 11].

Simulated annealing starts at some point in the design space and uses a cost function to direct itself towards valid solutions. Solutions with lower costs are said to be better than those with a higher cost. The starting point may be chosen at random or in the case of searching for a configuration to perform an upgrade may start at the baseline configuration. The algorithm for simulated annealing in relation to finding a configuration is given in figure 3.

The algorithm tries to minimise a cost function which assesses the quality of solutions. The temperature parameter governs the likelihood of accepting a higher cost solution and is reduced as the search progresses. The temperature is decreased by multiplying it by a cooling factor every M moves. The initial temperature value will be related to the nature of the problem. More difficult problems will typically require a higher starting temperature. For the task allocation and scheduling problem, we have found an initial temperature of 0.01 to be suitable for creating baseline configurations when starting with a random initial solution. Typically the cooling factor would be set to 0.99 and reduced every 3000 moves.

An important part of the simulated annealing algorithm is the neighbourhood function, η. The neighbourhood of a solution is the set of moves that it is possible to move to in a single step. The algorithm for making a move is given in algorithm 4. The algorithm randomly selects between changing the allocation or changing the priority of an object. Allocation changes are preferred over priority changes in the ratio 2:1. This ratio does not change the neighbourhood of solutions but makes some solutions more likely to be chosen. Geometrically, this can be pictured as moving in a random direction to reach a neighbour where some neighbours cover a larger area than others.

4.1 Cost Function

The cost function f in figure 3 is calculated from the results of the scheduling analysis. It is calculated from the scalar product of a vector of cost function components and a weightings vector.

$$\mathbf{g} = (g_1, \ldots, g_n)^T \tag{1}$$

$$\mathbf{w} = (w_1, \ldots, w_n)^T \text{where } w_i \in \mathbb{R}\ \forall\, i \tag{2}$$

$$f = \frac{\mathbf{g} \cdot \mathbf{w}}{\sum_1^n w_i} \tag{3}$$

The range of all cost components, g_i, is designed to be $[0, 1]$ so that when weightings are set, two components with equal weightings have an equal contribution to the overall cost. There are three classes of component functions. There are some which ensure a solution meets all timing requirements, some which try to give the solution additional desirable properties and finally, components which help in guiding the search towards a solution. Many components are multi-purpose, falling into more than category.

The first components assess the number of unschedulable tasks and messages.

$$g_1 = \frac{\#\{\text{unschedulable tasks}\}}{\#\text{TASKS}} \tag{4}$$

$$g_2 = \frac{\#\{\text{unschedulable messages}\}}{\#\text{MESSAGES}} \tag{5}$$

where $\#X$ is the number of elements in the set X, TASKS is the set of all tasks and MESSAGES is the set of all messages.

When tasks have dependencies and not all processors are interconnected, some allocations are likely to be invalid. For example, a task needs to receive a message but the message has been allocated to a network not connected to the processor on which the task resides. These components are typically associated with a high weighting.

$$g_3 = \frac{\#\{\text{invalid input allocations}\}}{\#\text{TASKS} + \#\text{MESSAGES}} \tag{6}$$

$$g_4 = \frac{\#\{\text{invalid output allocations}\}}{\#\text{TASKS} + \#\text{MESSAGES}} \tag{7}$$

The concept of invalid allocations is divided into two components. The first counts how many allocations are such that a task cannot receive a message or a message cannot be sent by a task. The second is the converse. It counts how many tasks cannot send a message and how many messages cannot reach their destination.

A task or message can be unschedulable for more reasons other than breaking its deadline. If a preceding task is not schedulable and does not have an accurate worst case response time, the release jitter for the current task cannot be calculated. Similarly if any higher priority tasks cannot be scheduled for this reason, lower priority tasks which are affected by jitter of higher priority tasks cannot be scheduled. This situation often occurs when tasks or messages are arranged in such a way that there is a circular dependency.

$$g_5 = \frac{\#\{\text{invalid arrangements}\}}{\#\text{TASKS} + \#\text{MESSAGES}} \tag{8}$$

This metric is useful in penalising tasks that cannot have a worst case response time calculated more heavily than tasks which have a response time greater than the deadline.

When a system is completely schedulable, all essential requirements have been met so it is desirable to create a local optima around this point. Some have used separate bonus functions for this purpose [11] but here it is preferred to include such a function as an additional component as it can be included without complicating the framework further.

$$g_6 = \#\{\text{unschedulable systems}\} \tag{9}$$

Obviously, this functions results in a value of 0 or 1.

Sensitivity analysis is one measure of flexibility of a set of tasks or messages. The calculation is based upon by how much worst case execution times can increase before the system becomes unschedulable. With complex dependencies, increasing the execution time of a single task can have far reaching effects to tasks on other processors. In the context of searching, sensitivity analysis can be viewed slightly differently to provide guidance to the search. Instead of asking by how much execution times can increase, it is preferable to ask what are the maximum possible execution times that can be used for the system to be schedulable. When the system is schedulable under the current configuration, these questions amount to the same thing. However, when the system is unschedulable, the latter question quantifies by how much the system is unschedulable.

Sensitivity values are calculated by multiplying all tasks or messages in the set of objects to be studied by a scaling factor to find the highest possible scaling factor with which the system is schedulable. This is implemented with a binary search. For each evaluation, a complete rerun of the scheduling analysis is required making this metric computationally intensive. For this reason, it is currently only used on the set of objects which includes every task and message in the system. Applying it to individual tasks or small sets of tasks may produce a higher quality solution but this is currently infeasible for moderately sized systems.

Sensitivity is calculated as follows for a set of objects S which may include tasks, messages or a combination.

$$g_7 = \frac{\sum_{i \in S} \text{SENS}_i}{\#S} \tag{10}$$

where SENS_i is the sensitivity of each object i calculated using

$$\text{SENS}_i = e^{-\lambda \text{SCAL}_i} \tag{11}$$

where SCAL_i the largest value of a factor $s \in \mathbb{R}$ such that the system is schedulable when the worst case execution time of task i, C_i, is set to $\lfloor sC_i \rfloor$. The parameter λ can be used to vary the range of scalability values over which the value of the cost component varies most. If the value of s reaches 0 then the system may still be unschedulable if not all objects are included in the set S.

Another function used to help a search move from an unschedulable solution to a schedulable solution is the load balancing function. This is based upon the variance of the utilisations of processors. It could also be applied to network utilisation but is not currently used for this purpose. The basis of this component is as follows.

$$b = \frac{\sum_{i \in \text{PROCS}} (U_i - \mu)^2}{\#\text{PROCS}\mu^2} \tag{12}$$

where PROCS is the set of all processors, U_i is the utilisation of processor i, and μ is the mean utilisation. The value b is in the range $[0, 1]$ and could be used for the load balance component. However, we do not rate a perfectly balanced system to be that much better than a slightly unbalanced one. This is particularly the case, as will be seen later, when the system design requires more flexibility in some parts of the system than others. The spare capacity should be targeted in the right places, not necessarily spread evenly through the system. Therefore the metric used is

$$g_8 = e^{k(b-1)} \tag{13}$$

where k is a positive constant. If the system is extremely unbalanced, it is likely that some processors are more than 100% utilised and the metric will have the effect of moving tasks to less utilised processors. This has proved extremely effective in moving towards a schedulable solution and is much more efficient to compute than sensitivity. As the system gets more balanced, this component has much less influence on the overall cost. k is set to 10 in our framework.

The final four components of the cost function are all concerned with minimising the number of changes that need to be made when moving from a baseline configuration to a new configuration for new requirements with changes from which the baseline configuration was intended. Since there are currently two elements to a configuration, namely allocation and priority, a component is needed to measure changes to each of these. These two components are split into separate components for tasks and messages. This allows separate weightings to be used if minimising task changes is preferred over minimising message changes or vice-versa.

Since only changes to the baseline configuration are of interest, new objects in the new requirements version are ignored. Similarly, any objects that have been removed in the new configuration are also ignored. If it is required an object is removed, it will be true of all new configurations and is not a change that can be minimised.

The components to measure the changes in allocations for tasks and message are as follows.

$$g_9 = \frac{\#\{i \in \text{CT} : A_i \neq A_i'\}}{\#\text{CT}} \tag{14}$$

$$g_{10} = \frac{\#\{i \in \text{CM} : A_i \neq A_i'\}}{\#\text{CM}} \tag{15}$$

where A_i is the current allocation of the object and A_i' is the allocation in the baseline configuration. CT is the set of tasks which are common to both the old and new requirements. CM is the equivalent set for messages.

The components for priorities are more complex than that for allocations. It would be possible to simply count priority differences. However, this is overly pessimistic as it is the priority ordering relative to other tasks and messages that is important rather than the specific identifier. In reality, flexible design should allow for new tasks to be inserted between existing tasks without modifying any priorities. This could be achieved by initially assigning priority values in multiples of 10 allowing new tasks to be inserted in between. We also want to consider two sets of tasks with the same priority order but with different numbering schemes to be considered equivalent.

For these reasons, the priority difference metric is based on Spearman's rank correlation coefficient. This requires the two sets of objects for the old and new configurations to be ranked in priority order. $\text{RANK}(i)$ is the rank of object i in the current configuration and $\text{RANK}(i)'$ is the rank of the same object in the baseline configuration.

$$n = \#\text{CT} \tag{16}$$

$$g_{11} = \frac{3}{n(n^2-1)} \sum_{i \in \text{CT}} (\text{RANK}(i) - \text{RANK}(i)')^2 \tag{17}$$

$$m = \#\text{CM} \tag{18}$$

$$g_{12} = \frac{3}{m(m^2-1)} \sum_{i \in \text{CM}} (\text{RANK}(i) - \text{RANK}(i)')^2 \tag{19}$$

These last four components all pull the solution back towards the original baseline configuration. This may be in conflict with making the system schedulable. In practice, this pull was in fact found to be too strong when major changes needed to be made to make a system schedulable. For this reason, components g_9, g_{10}, g_{11} and g_{12} are all currently set to 0 when the current configuration does not achieve a schedulable system. They are only used for the search to decide between schedulable solutions. This has achieved acceptable results. However, allowing the influence of these components to vary, depending on how close to being schedulable a current solution is, would allow the search to always be guided to some extent towards solutions with fewer changes. This may allow better solutions to be found more quickly.

4.2 Use Of Scenarios In Evaluating Configuration Quality

A problem with sensitivity analysis alone as a changeability heuristic is that it does not focus on any specific parts of the system. In general, it is impossible to make all of a system changeable. An alternative suggestion is to apply a number of scenarios and calculate the proportion of scenarios where requirements are met to those that are not met for a particular configuration. However, some issues have been identified with this method. Firstly, if a scenario incorporates new tasks, there will be no configuration information for those tasks so a question arises of how to allocate and schedule the new tasks alongside existing tasks. To truly evaluate whether a scenario could be met would require performing a search within the search over the space of configurations for the new tasks. This soon becomes computationally infeasible and certainly would not scale to looking at more than one level of future requirements.

Another issue is that purely counting whether a scenario is met or not does not give an accurate view of the system. Future scenarios are inexact in nature and hence it is desirable to evaluate how close a configuration is to meeting a future scenario. It may be impossible to meet a future scenario given the available hardware but some configurations will come closer to meeting requirements than others. Similarly, many configurations may be able to accommodate a future scenario but some may do it in a more flexible way than others allowing for a second tier of upgrades. Flexibility measures need to be applied to scenarios as well as the current system. This leads to the conclusion that the cost function presented in equation (3) should be applied to scenarios also.

The method chosen to incorporate scenarios into the search environment is to search for a global configuration that will simultaneously meet requirements of several systems, one of which is the current baseline system and the others are future scenarios. If different systems contain different tasks and messages, only relevant parts of the global configuration are applied to that system. A new cost value incorporating scenarios can then be calculated using

$$F = \frac{W_0 f(\omega) + W_1 f(\sigma_1) + \cdots + W_N f(\sigma_N)}{\sum_{i=0}^{N} W_i} \tag{20}$$

where f is the cost function defined in equation (3) and N is the number of scenarios. ω represents the current system requirements with the global configuration applied. $\sigma_1, \ldots, \sigma_N$ represent each scenario with the same configuration applied. The weightings $W_0, \ldots, W_N$ are associated with the importance of each scenario. W_0 should always have the highest weight. The weight for different scenarios may depend on the probability of particular system requirements becoming a reality.

To clarify the method, consider the following example. The baseline requirements for a system contains 30 tasks. Two future scenarios are to be considered when selecting a baseline configuration for these requirements. The first scenario only increases worst case execution times of some of the thirty tasks. The second scenario requires an additional two tasks as well as modifying some execution times. The global configuration will contain information for all 32 tasks. To evaluate the cost of using a particular configuration for the baseline requirements and first scenario, information for the appropriate subset of 30 tasks will be selected from the configuration. To evaluate the cost of the second scenario, all of the configuration will be used.

Say, at a particular point in the search, the cost of using a configuration on the baseline is 0.05, the first scenario has a cost of 0.08 and the second scenario has a cost of 0.14. The new cost function F is calculated as

$$\begin{aligned} F &= \frac{W_0 f(\omega) + W_1 f(\sigma_1) + W_2 f(\sigma_2)}{W_0 + W_1 + W_2} \\ &= \frac{10 * 0.05 + 4 * 0.08 + 3 * 0.14}{17} = 0.073 \end{aligned}$$

This method elegantly allows tasks and messages not present in the current system to be configured using the existing heuristic search without requiring a nested search. The number of cost function evaluations which includes scheduling and scalability calculations increases linearly with the number of systems.

The method may also be used in association with the cost components in equations (14), (15), (17) and (19) which measure the degree of change between current and previous configurations. This would be the case when it was desirable to both stay near the current solution but allow for flexibility in upgrading to future scenarios when selecting a configuration. This balancing act has not yet been attempted. Currently, either the number of changes from a previous configuration is minimised or the ability to meet future scenarios is maximised. In the latter situation, the components mentioned above are unused since there is no baseline configuration to measure the change from.

5 Evaluation

The evaluation performed in this section sets out to test the hypothesis given in the introduction. That is, whether the method described in the previous section successfully reduces the number of alterations when performing an anticipated upgrade. Also, it is hoped that using scenarios will improve flexibility for some changes which haven't been anticipated.

The problem used for the evaluation is taken from that used in [19] which describes a mapping problem from the automotive domain. The set of tasks, shown in table 2, are based on the cruise controller application described in [19]. In this table, C, T and D represent the worst case execution time, period and deadline of each tasks respectively. Figure 5 shows the dependencies between tasks. Where a message is passed between tasks its worst case communication time (WCCT) depends on how it is allocated. If it is allocated to a bus on the processor rather than a network, the time is assumed to be negligible and has a WCCT of 0. However, its resulting worst case response time may be non-zero due to jitter from preceding items in the transaction. If a message travels between processors, the connection rate and size of the message results in a WCCT of 2ms. The size of messages is fixed so this is true of all messages.

Task	C	T	D	Task	C	T	D
τ_1	0	150	150	τ_{18}	6	150	150
τ_2	12	150	150	τ_{19}	13	150	150
τ_3	7	150	150	τ_{20}	5	150	150
τ_4	10	150	150	τ_{21}	20	150	150
τ_5	5	150	150	τ_{22}	17	150	150
τ_6	18	150	150	τ_{23}	9	150	150
τ_7	14	150	150	τ_{24}	5	150	150
τ_8	8	150	150	τ_{25}	6	150	150
τ_9	5	150	150	τ_{26}	5	150	150
τ_{10}	10	150	150	τ_{27}	5	150	150
τ_{11}	6	150	150	τ_{28}	12	150	150
τ_{12}	7	150	150	τ_{29}	10	150	150
τ_{13}	11	150	150	τ_{30}	7	150	150
τ_{14}	5	150	150	τ_{31}	6	150	150
τ_{15}	8	150	150	τ_{32}	0	150	150
τ_{16}	11	150	150	τ_{33}	0	150	150
τ_{17}	15	150	150				

Table 2. System timing requirements

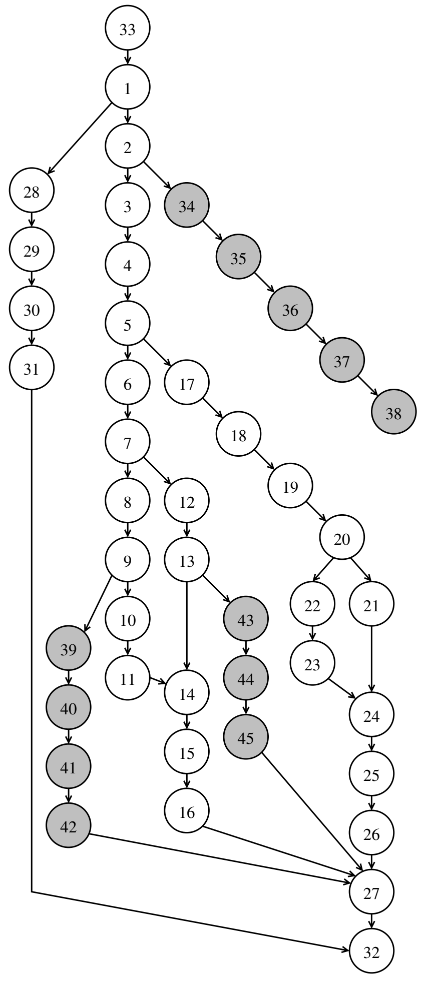

Figure 5. Task graph. Shaded tasks are used in scenario S_4.

Task	C
τ_2	14
τ_3	9
τ_4	12
τ_5	7

Table 3. Scenario S_1

Task	C
τ_{21}	22
τ_{22}	22
τ_{23}	12
τ_{24}	7
τ_{25}	9

Table 4. Scenario S_2

Task	C
τ_{21}	22
τ_{22}	22
τ_{23}	12
τ_{24}	7
τ_{25}	9
τ_{28}	15
τ_{29}	12
τ_{30}	10

Table 5. Scenario S_3

Task	C	T	D
τ_{34}	30	150	150
τ_{35}	32	150	150
τ_{36}	24	150	150
τ_{37}	18	150	150
τ_{38}	15	150	150
τ_{39}	10	150	150
τ_{40}	8	150	150
τ_{41}	12	150	150
τ_{42}	6	150	150
τ_{43}	8	150	150
τ_{44}	10	150	150
τ_{45}	10	150	150

Table 6. Scenario S_4

The hardware platform for the system consisted of 4 processors and 3 networks. The first network allows all processors to communicate. However, like a processor, the ability to schedule messages on a single network is limited so the other two networks are needed but only connect pairs of processors. One connects processor 1 to processor 2 and the other connects processor 3 to processor 4. Therefore, there is less capacity for communicating between each pair. This is based on the platform in [19].

Four change scenarios were created. The scenarios were purposely created to introduce changes of different sizes and for some scenarios to be similar to others. This is so that the ability to handle situations where all changes are anticipated, most changes are anticipated and the case where changes are incorrectly predicted are evaluated. Scenarios S_1 and S_2 both increase worst case execution times for a selected group of tasks. The tasks that are modified and the new execution times are shown in tables 3 and 4.

Scenario 3, whose details are in table 5, is an extension of scenario 2. It makes the same modifications as that of scenario 2 as well as increasing the execution times of some additional tasks.

Scenario 4 makes far larger changes. The existing tasks were not changed but several new tasks, shown as the shaded tasks in figure 5 and listed in table 6, were added. It is not possible to schedule all of these tasks on the current platform as at least one processor would have a utillisation above 100%. Therefore scenario S_4 allows for the addition of an extra processor. The processor is connected

Upgrade scenario	Scenarios used for baseline	g_9	g_{10}	g_{11}	g_{12}
S_1	NONE	0.2727	0.3333	0.2167	0.0968
S_1	S_1	0.0000	0.0000	0.0000	0.0000
S_1	S_1, S_2	0.0000	0.0000	0.0000	0.0000
S_1	S_2	0.1515	0.2778	0.1068	0.1318
S_1	S_3	0.0000	0.0000	0.0000	0.0000
S_1	S_4	0.1515	0.1944	0.0699	0.0875
S_2	NONE	0.3030	0.5278	0.1681	0.1683
S_2	S_1	0.0909	0.1111	0.0563	0.1000
S_2	S_1, S_2	0.0000	0.0000	0.0000	0.0000
S_2	S_2	0.0000	0.0000	0.0000	0.0000
S_2	S_3	0.0000	0.0000	0.0000	0.0000
S_2	S_4	0.2121	0.4444	0.0904	0.1559
S_3	NONE	0.1212	0.1944	0.1678	0.0752
S_3	S_1	0.2424	0.3889	0.1158	0.1931
S_3	S_1, S_2	0.0000	0.0000	0.0000	0.0063
S_3	S_2	0.0000	0.0000	0.0000	0.0000
S_3	S_3	0.0000	0.0000	0.0000	0.0000
S_3	S_4	0.3636	0.3611	0.0847	0.1607
S_4	NONE	0.3636	0.3611	0.1629	0.1040
S_4	S_1	0.3333	0.3333	0.1103	0.1611
S_4	S_1, S_2	0.2727	0.2222	0.1730	0.0937
S_4	S_2	0.3030	0.3056	0.1367	0.1095
S_4	S_3	0.2424	0.2778	0.2081	0.1420
S_4	S_4	0.0000	0.0000	0.0000	0.0000

Table 7. Results

to the broadcast network connecting all processors and the network also connecting processors 3 and 4.

A number of baseline configurations were created to meet the requirements in table 2. The baselines differed in the scenarios they were created to anticipate. A baseline was created with no changes anticipated. Rather than just selecting the first solution that met all requirements the search environment was allowed to improve the solution based on the load balance and sensitivity components. The other baselines were created for certain scenarios with solutions being assessed using the function in equation (20). In each case the weighting W_0 as described in equation (20) was set to 10 with the weighting for any scenarios set to 4. Once the baselines were created, an upgrade to the new requirements as set out in each scenario was attempted from each of the baselines. Situations where the scenarios the baseline was prepared for match the actual upgrade scenario is equivalent to all changes being correctly anticipated. When upgrading from baselines prepared with scenario 2 to meet the requirements in scenario 3 some but not all changes were anticipated. In other cases, either no changes were anticipated or a differing set of changes was anticipated.

The results of these experiments are presented in table 7. The success or failure of the baseline flexibility was judged on the number of changes required to meet the upgrade scenario. This was calculated from the change related cost components formulated in equations (14) to (19). Recounting these equations, with reference to table 7, g_9 and g_{10} measure the task allocation changes and message allocation changes respectively, while g_{11} and g_{12} measure task priority changes and message priority changes.

Weightings			
Component	Function	Weight	
Unschedulable Tasks	g_1	1200	1200
Unschedulable Messages	g_2	1200	1200
Unconnected Inputs	g_3	4500	4500
Unconnected Outputs	g_4	4500	4500
Invalid Arrangements	g_5	1500	1500
Unschedulable System	g_6	500	500
Sensitivity Analysis	g_7	50	50
Load Balancing	g_8	150	150
Task Allocation Change	g_9	0	200
Message Allocation Change	g_{10}	0	200
Task Priority Change	g_{11}	0	175
Message Priority Change	g_{12}	0	175
Search Parameters			
Parameter		Value	
Initial temperature		0.01	0.004
Inner loop iterations (M in figure 3)		3000	4000

Table 8. Search parameters. The left hand column values are for creating baselines. The right hand values are for upgrades.

In all cases where the changes were correctly anticipated, no, or in the case of scenario 4, very few, changes were required to meet the requirements of the upgrade scenario. In all but two cases, baselines created with scenarios performed better than those without. Both of these cases occurred when attempting to upgrade to scenario 3 but different changes had been anticipated. It is interesting to note that preparation with scenario 1 alone performed badly, while using both scenarios 1 and 2 to create the baseline required no changes to meet scenario 3 and also no changes to meet scenario 1. In the converse case, preparation with scenario 3 and then upgrading to scenario 1 also required no additional changes.

The two counter cases, support the unsurprising fact that the method performed better when the anticipated changes matched or were similar to the actual changes. However, in general, baselines which anticipated some form of change were more flexible than those which didn't, even if the wrong changes were anticipated. This implies that doing some form of change assessment and using this information to create scenarios for establishing a baseline solution is likely to be beneficial.

The parameters used for the evaluation are shown in table 8. Of particular note is that the initial temperature used when performing an upgrade is lower than that used when creating baselines. Also, the number of iterations at each temperature is increased hence slowing the rate of cooling for the upgrade situation.

When performing an upgrade, since the existing solution is used as a starting point, the initial solution is closer to meeting requirements than when creating a baseline in which case the initial solution is chosen at random. A slower rate of cooling and lower temperature result in a more intense search around the starting configuration. This is desirable for upgrading to new requirements. If a higher temperature is used the search has more energy and is more likely to find solutions further from the initial starting point. This produces better results for an initial baseline since more of the search space will be covered.

6 Conclusion

This work has set out a case for why the flexibility of an architecture should be considered early in the design process and argued that if change is predicted to some extent, then future upgrades and modifications will have less impact and can be achieved at a lower cost.

Using an architecture model restricted to task allocation and scheduling, a framework has been described which can make use of such change predictions. It is acknowledged that changes cannot be predicted precisely and therefore the framework should also improve how unanticipated changes are handled, particularly when they do not differ significantly from the actual changes.

A method has been outlined which builds on previous work solving allocation and scheduling problems with heuristics. The additional contribution is that change scenarios are introduced into the framework so that solutions not only meet all essential timing requirements but also are flexible to changes suggested by the scenario.

An evaluation was performed using an example based on a cruise controller application taken from the automotive domain. It was demonstrated that using scenarios to create solutions for a baseline set of requirements improved the flexibility of that solution when it was subjected to an upgrade using the same or similar changes described by the scenario. In the majority of cases, the flexibility with respect to unanticipated changes also improved.

References

[1] J. Axelsson. Three search strategies for architecture synthesis and partitioning of real-time systems. Technical Report R-96-32, Department of Computer and Information Science, Linkoping University Sweden, 1996.

[2] J. Daniels, P. W. Werner, and A. T. Bahill. Quantitative methods for tradeoff analyses. *Systems Engineering*, 4(3):190–212, 2001.

[3] R. P. Dick and N. K. Jha. MOCSYN: multiobjective core-based single-chip system synthesis. In *DATE '99: Proceedings of the conference on Design, automation and test in Europe*, page 55, 1999.

[4] L. Dobrica and E. Niemelä. A survey on software architecture analysis methods. *IEEE Transactions on Software Engineering*, 28(7):638–653, 2002.

[5] 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*, pages 136–143, June 1997.

[6] R. Kazman, M. Barbacci, M. Klein, S. J. Carrière, and S. G. Woods. Experience with performing architecture tradeoff analysis. In *ICSE '99: Proceedings of the 21st international conference on Software engineering*, pages 54–63, 1999.

[7] R. Kazman, L. J. Bass, M. Webb, and G. D. Abowd. SAAM: A method for analyzing the properties of software architectures. In *International Conference on Software Engineering*, pages 81–90, 1994.

[8] R. Kazman, M. Klein, and P. Clements. Evaluating software architectures for real-time systems. *Annals of Software Engineering*, 7(1-4):71–93, 1999.

[9] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[10] B. S. Mitchell and S. Mancoridis. Modeling the search landscape of metaheuristic software clustering algorithms. In *GECCO*, pages 2499–2510, 2003.

[11] M. Nicholson. *Selecting a Topology for Safety-Critical Real-Time Control Systems*. PhD thesis, Department of Computer Science, University of York, 1998. DPhil Thesis.

[12] J. C. Palencia and M. G. Harbour. Schedulability analysis for tasks with static and dynamic offsets. In *RTSS '98: Proceedings of the IEEE Real-Time Systems Symposium*, pages 26–37, 1998.

[13] C. C. Price. Task allocation in distributed systems: A survey of practical strategies. In *ACM 82: Proceedings of the ACM '82 conference*, pages 176–181, 1982.

[14] C. Reeves, editor. *Modern Heuristic Techniques for Combinatorial Problems*. Blackwell Scientific Publishing, Oxford, England, 1993.

[15] J. Regehr. Scheduling tasks with mixed preemption relations for robustness to timing faults. In *RTSS '02: Proceedings of the 23rd IEEE Real-Time Systems Symposium (RTSS'02)*, pages 315–326, 2002.

[16] I. Sommerville. *Software Engineering*. Pearson Addison Wesley, seventh edition, 2004.

[17] K. Tindell, A. Burns, and A. Wellings. Allocating hard real-time tasks: An NP-hard problem made easy. *Real-Time Systems*, 4(2):145–165, June 1992.

[18] S. Wang, J. R. Merrick, and K. G. Shin. Component allocation with multiple resource constraints for embedded real-time software design. In *Proceedings of the 10th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS 2004)*, pages 219–228, 2004.

[19] A. Yadav. Semi automatic mapping of automotive electronic functionality. Master's thesis, Volvo Technology Corporation, 2004.