Available online at www.sciencedirect.com
ScienceDirect
The Journal of Systems and Software 81 (2008) 1253–1271
The Journal of Systems and Software
www.elsevier.com/locate/jss

Systematic approaches to understanding and evaluating design trade-offs

I. Bate

Department of Computer Science, University of York, York, United Kingdom

Received 27 June 2006; received in revised form 1 October 2007; accepted 2 October 2007
Available online 9 January 2008

Abstract

The use of trade-off analysis as part of optimising designs has been an emerging technique for a number of years. However, only recently has much work been done with respect to systematically deriving the understanding of the system problem to be optimised and using this information as part of the design process. As systems have become larger and more complex then a need has arisen for suitable approaches. The system problem consists of design choices, measures for individual values related to quality attributes and weights to balance the relative importance of each individual quality attribute. In this paper, a method is presented for establishing an understanding of a system problem using the goal structuring notation (GSN). The motivation for this work is borne out of experience working on embedded systems in the context of critical systems where the cost of change can be large and the impact of design errors potentially catastrophic. A particular focus is deriving an understanding of the problem so that different solutions can be assessed quantitatively, which allows more definitive choices to be made. A secondary benefit is it also enables design using heuristic search approaches which is another area of our research. The overall approach is demonstrated through a case study which is a task allocation problem.

1. Introduction

For a number of years research has been performed research into two important areas in isolation from one another. These are: understanding design choices, trade-offs and evaluation criteria during the design of systems; and the application of search-based techniques to optimise designs. These two areas are by their very nature linked with the former effectively being an input to the latter, i.e., the first stage derives an understanding of the problems and the second stage finds the solution. However, the majority of work in either one of these areas has been done quite independently of the other. This has lead to a loss of traceability and rationale between the derivation of the problem and its solution.

The aim of this work is to present a technique for understanding the trade-offs. This information can then be used as part of either a manual or automated design process. The motivation for the work is borne out of experience working on embedded systems in the context of critical systems where the cost of design changes can be large and the impact of design errors potentially catastrophic. During our previous work (Bate and Burns, 2003), adapting existing scheduling and timing analysis for use in 'real' critical systems, a number of important design decisions (e.g., the type of timing watchdog used to identify timing failures) were faced whose impact had far reaching consequences across the system's design. However, a lack of suitable techniques was found for considering the trades-off involved. The key deficiencies with these techniques are a lack of a systematic method and poor support for capturing rationale and assumptions. Both of these meant a lack of support for maintenance and the results obtained were of questionable integrity. Therefore, the research into suitable methods presented in this paper was instigated.

The method used for deriving the trade-off analysis problem is based on the goal structuring notation (GSN)

E-mail address: iain.bate@cs.york.ac.uk

0164-1212/$ - see front matter © 2008 Elsevier Inc. All rights reserved.
doi:10.1016/j.jss.2007.10.032

(Kelly, 1999). GSN was originally developed for constructing safety arguments for systems and has since achieved widespread use. In the context of this work, it is considered to have some advantages including stronger traceability, better support for capturing rationale and is easily mapped onto traditional optimisation algorithms. However, the key contribution in this paper, with respect to GSN, is how GSN is used as part of designing systems rather than for constructing safety cases. It is noted that other notations offering similar ways of decomposing objectives and capturing assumptions etc could be used within the technique proposed.

The process of establishing the trade-off analysis problem begins with using GSN to decompose the top-level objectives (often referred to as quality attributes) of the system in a hierarchical tree-like fashion. The decomposition is continued until the objectives reach a suitably low-level that they measure how well the specific individual objectives are met. An example of this is a higher-level objective of meeting the requirements could be decomposed through an objective of meeting the timing requirements to a lower-level objective that tasks' response times must always be less than or equal to their deadline. This last objective can be assessed, and appropriate evidence gathered, using timing analysis (Audsley et al., 1995). Using the hierarchy information given by the tree, individual results can later be combined to give results for higher-level objectives. For instance, a weighted sum of results for lower-level objectives may be used to provide a single higher-level result. Combining the results of individual objective functions into a single overall objective. This can then be often used in any form of cost benefit analysis including as part of a fitness functions within a search algorithms. As such the approach represents a logical approach to systematically building knowledge of how to optimise designs featuring arbitrarily complex trade-offs.

At the same time as performing this decomposition, design choices (e.g., choice of computational model between static scheduling and fixed priority scheduling) and assumptions (e.g., the tasks have predictable and bounded execution times) are captured.

This is a distinctly different approach from techniques such as the Architecture trade-off analysis method (ATAM) (Bass et al., 2003) as their method relies heavily on the information being provided by experts or reusing information derived during previous applications of the technique (e.g., from an associated handbook). There are also a wide variety of methods of performing trade-offs including automated search techniques such as simulated annealing and genetic algorithms (Rayward-Smith et al., 1996). Of these, to the best of our knowledge none of the work has addressed how the problems should be derived by systematic means. This is a key benefit of the work presented here.

The combination of deriving an understanding of the design problem with the mechanism for making the design decisions in a more traceable manner at the same time as capturing the rationale has a number of significant advantages for practitioners in different domains. For all domains, the rationale will provide better support for the change process as the reasons behind the original design and links between parts of the design will allow the risk and impact of change to be considered more thoroughly (Bass et al., 2003). For safety-related domains, the traceability of design through to objectives (including the decomposition of high-level objectives to low-level objectives), mapping the objectives to assessment criteria and mechanisms and then to evidence the objectives are met is the basis for most standards (Herrmann, 2000; United Kingdom Ministry of Defence, 1996; RTCA Inc, 1992; CENELEC, 2001; United Kingdom Ministry of Defence, 2004). In particular, some standards are now moving away from traditional process-oriented approaches (United Kingdom Ministry of Defence, 1996) to product-based (in other words evidence-based) standards (United Kingdom Ministry of Defence, 2004). The reason is the process-oriented standards tend to lead to a tick box mentality rather than fundamentally questioning the needs of the project regarding safety (McDermid, 2001).

Therefore, the key contributions of this paper are: the understanding of the contextual information upon which the design trade-offs are made in order to support reuse and maintenance of the system's design; the well-defined and systematically derived knowledge of the design problem and solution aids certification which could also be reused on similar systems; and then the mapping of these onto quantitative measures, where possible, raising the possibility of using these measures as part of an automated search strategy. A significant part of the contribution is the means of turning difficult to assess criteria into quantitative measures using a variety of means, e.g., through the use of scenarios.

The structure of the paper is as follows. Section 2 contains a literature survey that considers what related work exists and as such helps establish the contribution in this paper. Background on the GSN and its application in the critical systems domain is given in Section 3. A detailed description of the method is given in Section 4. A case study is then used to demonstrate the method in Section 5. Finally, Section 6 provides a summary of the work, concluding remarks and suggests areas for future work.

2. Related work

The purpose of this section is to consider the existing work on design and trade-off analysis that is related to this paper. A comprehensive survey of the subject can be found at Dobrica and Niemela (2002).

ATAM, a technique for evaluating architectures for their support of architectural qualities and trade-offs in achieving those qualities, has been developed by the Software Engineering Institute (Bass et al., 2003; Kazman et al., 1999). The approach is largely based on deriving quality attributes from overall system objectives, and then

turning the quality attributes into questions that can be asked of the architecture and its designers. Our approach could be used as part of the ATAM approach in order to give a more systematic means for deriving the quality attributes and to help improve maintainability of the information. There are a number of stages within ATAM in which our work would fit including the following: identify architecture approaches, generate quality attribute utility tree, analyze architecture approaches and analyze architecture approaches, respectively.

There are numerous other techniques that follow a similar philosophy as ATAM – for example goal question metrics (GQM) (Rombach and Basili, 1988), cost benefit analysis method (CBAM) (Moore et al., 2003), goal-based requirements analysis method (GBRAM) (Anton, 1996; Anton and Potts, 1998), software architecture analysis method (SAAM) (Kazman et al., 1994), quality function deployment (QFD) (Kogure and Akao, 1993) and Mylopoulos' goal graphs (Mylopoulos et al., 1992; Tahvildari et al., 2003; Chung et al., 1999).

GQM (Rombach and Basili, 1988) addresses slightly different needs to our approach and as such the two approaches could also be integrated. GQM is a process by which objectives can be turned into specific questions for which the answers are measurable. For instance, GQM could be used to derive assessment criteria from the objectives contained within augments in GSN. Again, we can use some of the findings when converting our arguments into assessment criteria. SAAM (Kazman et al., 1994) was produced by the same people as ATAM and as such it effectively represents an early cut-down version of the approach. In a similar fashion to ATAM approach could be used within SAAM's process to derive the understanding of the quality attributes. CBAM (Moore et al., 2003) provides a means of gaining measures of software quality in financial terms, however even though it has been developed by the same people as ATAM only a small amount of work has been done to explicitly integrate the two approaches and this is largely superficial (Nord et al., 2003). CBAM again complements our approach in that it uses the knowledge of the quality for the system, and turns this knowledge into measures.

GBRAM (Anton, 1996; Anton and Potts, 1998) is another goal-oriented approach that explores the systems objectives and how they can be evaluated. However, there are two distinct differences. First, the goals are organised in a flat structure rather than supporting hierarchical decomposition. Secondly, it is assuming the objectives are mined from existing sources, e.g., documentation and uses cases, rather than determined as a by-product of the process itself. One of its main foci is to capture the rationale linking the requirements mined from the documentation to the actual design. A language to support the work on GBRAM is provided by Lee (Lee, 1991). Lee's work also extends GBRA to provide better support for capturing the rationale.

Potentially the closest work to ours is that of Mylopoulos (Mylopoulos et al., 1992; Tahvildari et al., 2003; Chung et al., 1999) in that it is largely based on a graphical notation and it originated from the consideration of non-functional properties of systems. Again, the work is more focussed on representing the requirements of the system rather than how they are derived and consequently used in the design of systems.

The differences between our strategy and other existing approaches for deriving quality attributes and measures, e.g., ATAM, include the following:

1. The techniques used in our approach are already accepted and widely used for constructing safety arguments (e.g., nuclear propulsion system and missile system safety arguments) (Anton and Potts, 1998), and as such processes exist for ensuring the correctness and consistency of the results obtained.
2. The techniques offers: (a) strong traceability and a rigorous method for deriving the attributes and assessment criteria (this is considered to be more rigorous than questions) with which designs are analysed; (b) the ability to capture design rationale and assumptions which is essential if component reuse is to be achieved (Kelly, 1999).
3. Information can be reused in a context different from their original intended use, rather than repeating the effort (Bate and Kelly, 2003).
4. The method has considered how to derive the quality attributes as well as deduce what is needed from quantitative measures to show whether they are met or not.
5. The method is equally applicable as a design technique to assist in the evaluation of the architectural design and implementation strategy as it is for evaluating a design at particular fixed stages of the process.

As stated above, a defining characteristic of our proposed method is the use of a well-established argumentation notation, GSN, to explore alternative satisfaction arguments for desirable architectural criteria.

3. Background on the GSN and its Previous Use

3.1. Overview of GSN

As stated earlier, the problem derivation is based on GSN (Kelly, 1999). GSN was originally derived for use in the production of safety cases as part of the certification of systems. During the establishment of the safety argument's claims (often referred to as goals), context, assumptions and justifications are captured which has a number of uses including managing change. The GSN (Kelly, 1999) – a graphical argumentation notation – explicitly represents the individual elements of any safety argument (requirements, claims, evidence and context) and, perhaps more significantly, the relationships that exist between these elements (i.e., how individual requirements are supported by specific claims, how claims are supported by evidence and the assumed context that is defined for the argument).

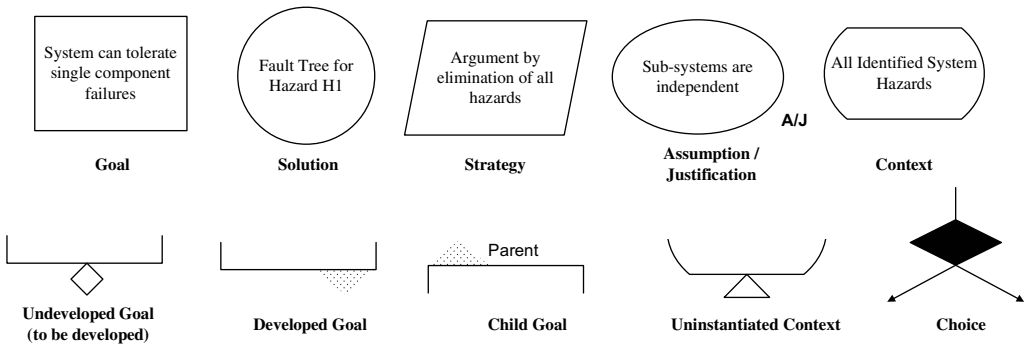

Fig. 1. Principal elements of the goal structuring notation.

The principal symbols of the notation are shown in Fig. 1 (with example instances of each concept).

The principal purpose of a goal structure is to show how goals (claims about the system) are successively broken down into sub-goals until a point is reached where claims can be supported by direct reference to available evidence (solutions). As part of this decomposition, using the GSN it is also possible to make clear the argument strategies adopted (e.g., adopting a quantitative or qualitative approach), the rationale for the approach (i.e., assumptions, justifications) and the context in which goals are stated (e.g., the system scope or the assumed operational role). To prevent the need for a single arguments for the whole system, decomposition is supported by allowing arguments to be split into smaller parts. The link between arguments is provided by the parent/child goal concept which is a linking goal between arguments. That is, the parent (or developed) goal sits at the bottom of a higher-level argument and the child goal at the top of the supporting argument. For further details on GSN see (Kelly, 1999). It should be noted that GSN has been widely adopted by safety-critical industries for the presentation of safety arguments within safety cases and hence appropriate processes exist to support their production.

3.2. Example safety case argument in GSN

Fig. 2 presents an example of the use of GSN as part of a safety case. The argument presents a portion of an argument towards supporting a claim that a system is acceptably safe. In particular, the argument decomposes the goal "*Show that a system meets its timing requirements*" to lower-level claims (evidence requirements) until solutions are reached, e.g., evidence based on the "*Results from sensitivity analysis*". Where an argument has not yet been developed to a point at which a solution has been reached, then the lowest-level claim is considered undeveloped, e.g., claim *G*6, and this is indicated by having a small diamond below the claim. The argument also shows the use of context information (i.e., defining the context *C*1 as the system has hard real-time requirements), the use of justifications (i.e., justification *J*1 that explains why the evidence from sensitivity analysis supports the claim *G*4), and the use of assumptions (e.g., assumption *A*1 that the timing requirements have been validated).

A key aspect of GSN is the clarity with which decisions about how claims are argued is presented. For instance claim *G*3, that the system is shown to meet its timing requirements in the presence of anticipated failures, can be supported in a number of ways. *G*3 can be supported in one or more ways which means either a single tactic can be used or a defence in depth approach featuring multiple tactics. A defence in depth strategy helps strengthen the argument of a system as it adds diversity and possibly independence to the evidence (United Kingdom Ministry of Defence, 1996). It is feasible for more than one of the options to be chosen, however to be beneficial clear independence needs to be maintained. For example it is usual to have a timing watchdog to protect against timing failures implemented in both hardware and software. However, the best solution, from a dependability perspective, tends to be one where there is an enforced separation between the protection mechanism for hardware and software.

4. Method for deriving an understanding of system trade-offs

This section is concerned with presenting the method that has been derived along with the reasons behind it.

4.1. Process

In Bate and Kelly (2002); Bate and Kelly, 2003, our method for architectural trade-off analysis for use within a systems engineering process was originally introduced. Fig. 3 provides an overview of the method. The individual stages are explained in the following sections.

4.2. Stage 1 – Presenting the current design

Stage (1) of the trade-off analysis method is producing a model of the system to be assessed. This model should be decomposed to a uniform level of abstraction. In this paper, simple block diagram-based approaches are used for this purpose, however it could be applied to any modelling approach that clearly identifies components and the interfaces between the components. For instance other

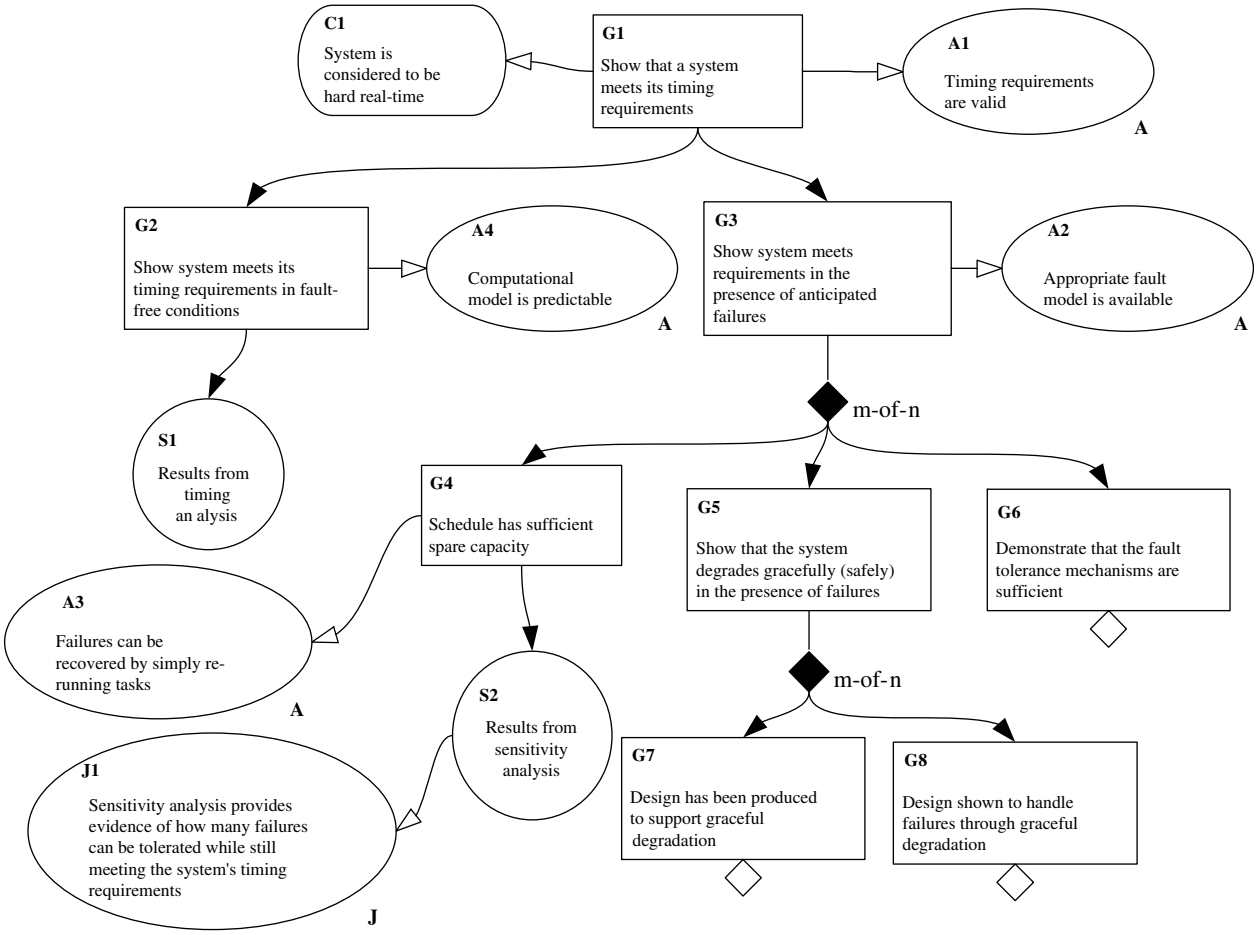

Fig. 2. Safety argument example.

examples have used UML class diagrams. There are two key parts to this stage. Part 1(a) is presenting the initial design and part 1(b) performing any modifications that may be needed. If changes are made to the design then it may be necessary to repeat stages 1–3 of the process.

4.3. Stage 2 – Producing an argument for the key objectives

In stage (2), the key objectives and properties of the system are decomposed into detailed design requirements that need to be satisfied. Rationale for these detailed requirements is encapsulated by structured arguments, along with the appropriate context, identifying where design choices are available. The arguments are structured using GSN (Kelly, 1999). Key properties of interest include: lifecycle cost, dependability, and maintainability. Clearly these properties can be broken down further, e.g., lifecycle cost into development, future upgrades and maintenance. Objectives of interest include; managed change, ease of integration and ease of verification.

Fig. 4 presents the GSN symbols but this time annotated with design-related comments.

The type of argument presented in Fig. 2 is also representative of how the design of systems is typically approached. To illustrate this a complementary, to Fig. 2, design argument is given in Figs. 5 and 6. The design arguments show some key similarities and differences between the safety and design arguments.

One of the key similarities between the arguments for safety and design is that there are often similar choices available independent of whether the argument is concerned with design or safety. For example, both arguments contain a higher-level claim that the timing requirements are met in the presence of failures and similar choices of ensuring there is spare capacity, degradation is graceful and fault tolerance schemes are sufficient. Another similarity, is higher-level objectives for both safety and design are decomposed to lower-level ones by stepwise refinement. At the lowest level, both design requirements and safety claims need to be supported by evidence showing, with sufficient confidence, that they are met.

The key differences are that:

1. Rather than there being an assumption (*A2*) in the safety case argument concerning the availability of a failure model, for the design argument there is context that the failure model has actually been obtained via simulation. The reason for the difference is that for the safety

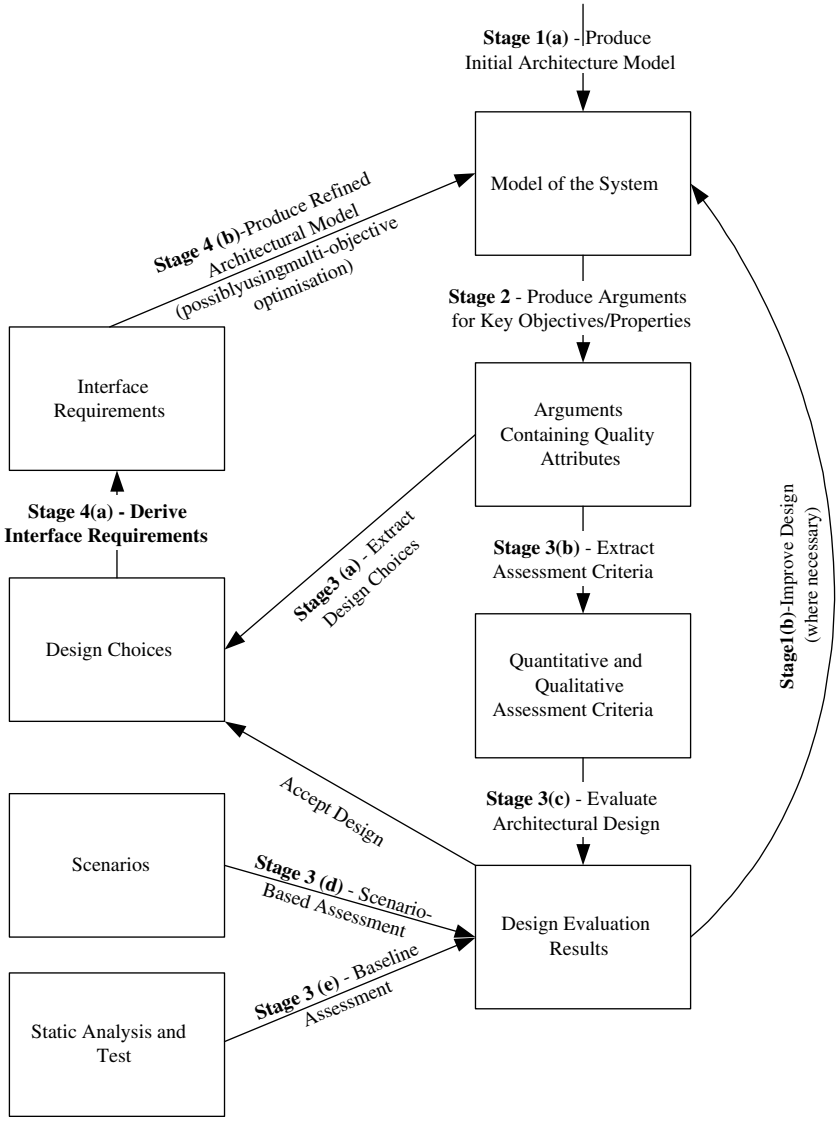

Fig. 3. Overview of the method stage.

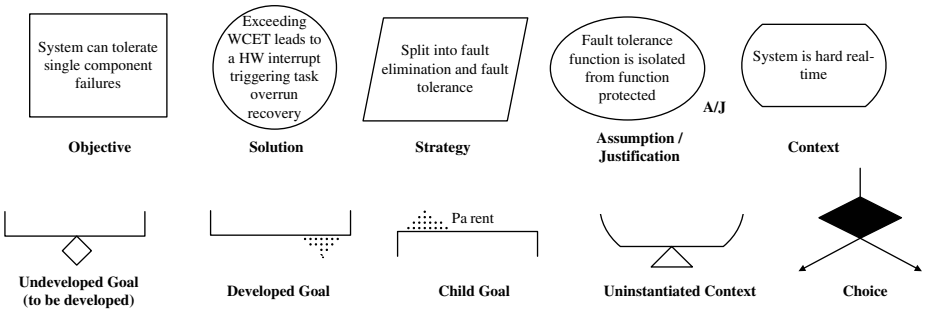

Fig. 4. Principal elements of the goal structuring notation for design.

case argument it is sufficient to make the assumption the failure model is available and the actual details are not needed. For the design argument, the actual details are needed in order to do a quantitative analysis.

2. The argument is augmented with an importance factor. In the case of Goal $D10$, it is annotated with the symbol VHO which indicates its relative importance to other objectives. The importance factor can later be used to identify the range within the set of weightings that should be used.
3. In the case of safety arguments the goals are either met or they are not, i.e., it is a binary result. In contrast a design goal can be partially met, for example a goal may be that percentage of the task deadlines are met.

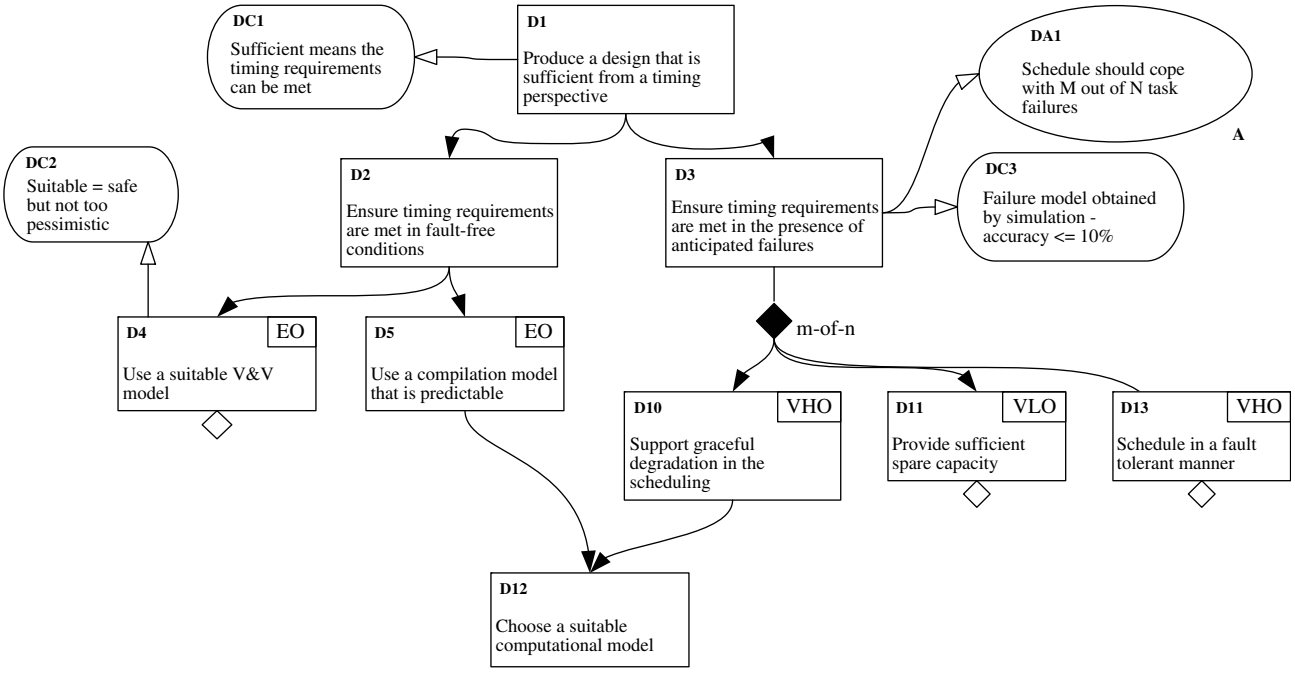

Fig. 5. Design argument example.

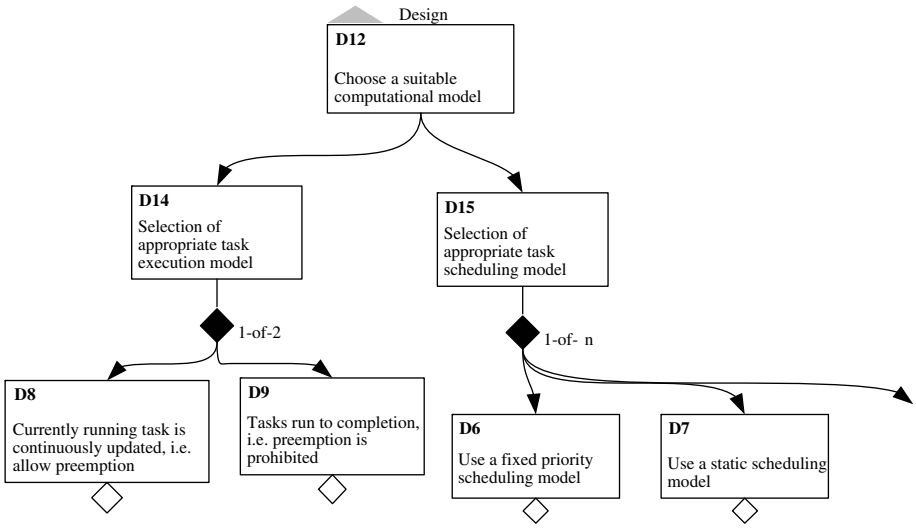

Fig. 6. Continuation of the design argument example.

For our work, three levels of importance have been used as application of this method has shown this to be sufficient. Less levels lack expressiveness but more levels are difficult to distinguish between. *EO* are those objectives considered essential to the correct operation of the system. For example in a hard real-time system an objective to meet the timing requirements is considered essential. The other two levels are *value added* objectives that improve the design but are not considered essential. *VHO* refers to *value added* objectives that are considered more important than those classed as *VLO*. An example of a *VHO* objective could be the ability to meet timing requirements in a soft real-time system. A *VLO* objective could be the ability to increase task execution times during software maintenance. Once the importance levels have been assigned then weightings can be chosen to support the evaluation of systems.

There are two principal drivers for choosing the appropriate weightings. First, and most importantly, the weightings help balance the individual results from cost functions to give an overall cost function for the full range of objectives. Second, in the case of automated searching

the weightings should be chosen in order to reduce the likelihood of the system getting stuck in a local minima (Rayward-Smith et al., 1996; Coello Coello, 1999).

The precise value of weightings has been the subject of a great deal of academic work. The current approach taken in this work is an initial assignment based on a subjective assessment of the relevant objective category. Then, some degree of hand tailoring is performed based on the results of the optimisation. Future work will explore more principled methods of using the information from the GSN-based representative of the system's objectives into weightings.

4.4. Stage 3 – Extracting information from the argument

Stage (3) uses the structured argument to further derive design and verification options, and to determine assessment criteria which can be used to judge how well a particular design solution meets the system objectives. Initially in the early stages of design, the evaluation may have to be qualitative in nature but as the design is refined then quantitative assessment may be used where appropriate. Part of this activity may use representative scenarios to evaluate the solutions. In the case of timing, representative scenarios will include situations where the software/system is changed which leads to modified task execution times and added/removed tasks. The use of scenarios in the context of real-time control systems has been extensively investigated in Bate et al. (2003), Bate et al. (2003) and Bate and Emberson (2006).

4.4.1. Stage 3(a) – Extracting design choices

When decomposing objectives, sometimes there is an option over how the objective may be met. For example, an objective that the system is dependable could lead to a choice of fault elimination, or fault identification and recovery. Where a choice is reached, from a safety perspective both choices are normally decomposed until a solution is reached. However, from a design perspective an individual choice may not be pursued further. The reason for this could be one of many including certain design restrictions have been made before the derivation of the problem commenced (signified as an *assumption*) or the designer(s) applying the method decides not to pursue the option further. A choice may not be pursued further based on any number of reasons including experience or simply a desire to manage the size of the potential design space. Such a decision could be supported by a *justification*.

In Fig. 6 a number of objectives are shown for which there are design options, e.g., over the choice of execution and scheduling approaches as part of the computational model. Objectives can be satisfied by one or many of the options proposed. The figure also shows dangling references, signified by a filled diamond under the goal symbol, which indicates that in this particular example there are other design options that are not extrapolated.

Fig. 6 also shows how at one level there can be a choice over how an objective is met and then at another (lower level) the individual choices can lead to further choices in how the system is designed. This reflects the natural way in which decisions can be made in a hierarchical fashion and hence provides an appropriate mapping onto real world problems. This will be demonstrated later in this paper.

4.4.2. Stage 3(b) – Extracting evaluation criteria

Once a suitable design argument exists, the evaluation criteria (i.e., objectives) need to be extracted. This is a relatively simple step as they are indicated in the argument by their importance level. For example in Fig. 5 there are a number of criteria including support for graceful degradation, providing sufficient capacity and scheduling in a fault tolerant manner.

4.4.3. Stage 3(c) – Evaluating the design

Given a set of evaluation criteria to be assessed, the next stage is to determine how an individual objective from the argument can be converted into a value such that different design options can be compared. The stages involved in turning a given objective into a value function are: understanding how different solutions may meet the objective to varying degrees, deriving a means of metrication and turning the function into a normalised form. The requirements for deriving a value function are as follows:

1. *Normalised:* Independent of the scoring mechanism used, the value returned should be normalised because
   (a) It eases the problem of weighting different functions as with normalised values a function that is twice as important as another is simply given a weighting that is twice the magnitude of the other.
   (b) It prevents the function returned being dependent on the particular application being considered and hence greater reuse (without modification) can be achieved. For example, a measure of task schedulability should not be proportional to the number of task deadlines achieved as the value returned is then dependent on the number of tasks associated with the application. This could cause two problems. First, the weightings in the value function may need to be altered in order to accommodate the number of tasks. Second, it makes it difficult to compare how well the same design copes with two different applications and hence reuse is affected. Instead, the measure of task schedulability could be based on the ratio of the number of deadlines met versus the number of tasks. This results in a range of values between zero and one.
2. *Quantitative:* Wherever possible the function should be evaluated by quantitative rather than subjective means. There are a number of reasons for preferring quantitative assessment but the main ones are that greater consistency is achieved between repeated applications of the function and that subjective assessment cannot be automated. It is however recognised that some objectives cannot easily be represented quantitatively.

The following two sections describe the methods used to evaluate the criteria.

4.4.4. Stage 3(d) – Scenario-based assessment

The simplest form of objective to evaluate are those that can be evaluated using standard static analysis and test, i.e., stage 3(e) of the overall method – refer to Section 4.4.5 for details. Examples of these forms of analysis include the system meets its timing requirements and sufficient capacity is provided. Both of these can be assessed using appropriate analysis techniques. The first of these objectives can be scored according to the percentage of requirements that are met. The latter by the ratio of the utilisation needed versus the utilisation available.

However, not all objectives are as easy to assess. The main reason for this is a lack of precise information that can be used as an input to the evaluation. An example of a seemingly difficult objective to turn into a quantitative normalised function is given to illustrate how the requirement for quantitative assessment can be achieved. Consider an objective that a particular part of a design is flexible to change. This type of objective is considered difficult to change as it is a very open question especially as the property considered or the type of change is not specified. However, even this objective can be measured using quantitative methods. One method of assessing this objective is to use scenarios. Scenarios of change can be applied to a particular design configuration and the impact of change assessed (Kazman et al., 1994). The impact of change can be assessed from two perspectives; whether the specific changes requires a change to the wider design and if the wider design has to be altered then by how much.

Taking the example further. For a given set of changes to how a software procedure is implemented, this can be assessed to see whether the procedure's interface is affected and whether the resulting changes to its execution time affect the scheduling of the system. If the interface is affected, then the effect could be measured based on the resulting lines of code that need to be changed. If the scheduling is affected, then the effect could be measured based on the number of priorities that need to be changed. These forms of measure allow a wide range of properties and scenarios to be evaluated limited by the time required to assess the scenarios. Clearly some scenarios and properties will take longer to evaluate than others. If the changes are measured via the lines of code changed then this is likely to be costly and laborious. However, changes to task attributes, e.g., priority levels, can be performed quickly via automation. Due to the time, and hence cost, involved in performing the scenarios combined with the potential lack of certainty of the exact changes to be handled, then the careful selection of the number and types of scenarios needs to be made. A tiered approach could be performed where different weightings are given to different types and degrees of change. The selection and use of scenarios is outside the scope of this work. Refer to Dobrica and Niemela (2002) and Bate and Emberson (2006) for further details. The results of the scenario-based assessment are then normalised by considering the ratio of scenarios handled without change to the overall number of scenarios. Further details are provided in Section 5.4.4 of the mapping of the scenario-based assessment to a quantitative measure.

An important decision to be made during the conversion of an objective into cost functions is choosing the best way of performing the calculation when there are a number of options. For instance, static analysis of how well a system copes with certain parameters changing and under these conditions the results are accurate. For example, using sensitivity analysis (Burns et al., 1996) to find out by how much individual task's WCET (worst-case execution time) can be increased. However, scenarios can cope with many more parameters being altered, e.g., whether the schedule can cope with different combinations of tasks and different magnitudes of WCET increase. When choosing the most appropriate method the principal influences are the possible errors in value and the time required for computation.

4.4.5. Stage 3(e) – Baseline assessment

The final stage of evaluation is to consider the baseline design, i.e., the design operating without any form of scenario being applied. For most system problems there already exists a wide range of analysis and test methods that can evaluate the key properties and objectives of concern. For example, with respect to timing there exists schedulability analysis, e.g., for fixed priority scheduling (Audsley et al., 1995), energy usage means of assessing the cost of processing, e.g., WATTCH (brroks et al., 2000), and reliability means of assessing failure properties, e.g., mean time between failures (Villmeur, 1992).

4.5. Stage 4 – Decomposing the design

Once a suitable design has been produced as part of stages 1–3 of the process, the next stage is to consider how the design should be decomposed to the next level before the stages recommence at stage 1. There are two distinct parts to this stage; defining appropriate interfaces and considering the choices that are available.

The interface requirements refer to those constraints that exist between different parts of the design. These may be represented by assumptions and context within the design argument. For example in Fig. 5 the design assumption *DA1* represents a constraint between the timing requirements of the applications, the scheduler and more specifically the scheduler's fault model. The constraint is that the scheduler should allow the applications to meet their requirements in the presence of M out of N task failures. Another constraint between the applications' requirements and scheduler is given by context *DC*3. This states that the design should be tolerant to bounded errors in the simulation used for analysis.

Examples of design choices are shown in Fig. 6. The two sets of choices are related in that the first deals with how

tasks are scheduled and the second with how they are executed. The choice of scheduling model, which emerges from the decomposition of goal *D*15, is concerned with whether fixed priority scheduling or static scheduling is used. This form of design decision can be taken at any time but it is more appropriate to do so once more information is available on the applications' timing requirements, the fault tolerance characteristics needed from the system and the desired flexibility.

5. Case study

5.1. Background on the problem

The problem considered in this paper is that of task allocation. The main aim of the task allocation problem is to ensure the system's timing requirements are met where the requirements feature both independent tasks and relationships (i.e., dependencies) between tasks. The task allocation problem has two main parts. First assigning tasks to specific processors (when there are more than one). Second choosing attributes (e.g., priority, ordering, etc.) for tasks. Where there is more than one processor, then similar attributes and allocations have to be made to messages that communicate information between tasks via databus(s). For example, the message has to be allocated so that data are routed between the two tasks and so the message is sent at the same rate as the source task generates data.

The task allocation problem has been chosen because it has already been studied by a number of research projects including (Fohler and Koza, 1989; Nicholson, 1998, 1996). However, none of these have established the problem to be solved, and the eventual solution, by systematic means. Also, the majority of work on this subject has solved the problem with little regard for properties other than those judged essential. For instance, their main goal is to meet the timing requirement with some of the work having other goals such as a secondary goal of minimising the amount of hardware needed. That is, most have entirely concentrated on ensuring timing requirements with a few projects also considering the issue of optimising the resources used.

5.2. Stage 1 – Presenting the current design

In this paper, the starting position for the design is shown in Fig. 7. It is noted that at this stage the structure corresponds to that of most evolutionary and heuristic search algorithms.

The diagram shows five principal components that are used when turning the requirements (including the objectives) into a satisfactory solution. These are:

1. *Initial solution* – an initial solution is provided by an appropriate means, e.g., random selection.
2. *Modify task allocation* – the search algorithm modifies the task allocation of the current solution.
3. *Modify task attributes* – the search algorithm modifies the task attributes of the current solution.
4. *Range of design choices* – represents the means of changing the design space over which the search algorithm operates.
5. *Check objectives are met* – evaluates the current solution against the objectives. Depending on the results, the solution will either be judged as satisfactory or fed back to an earlier stage so that it can be modified.

5.3. Stage 2 – Producing an argument for the key objectives

The design in Fig. 7 corresponds to the argument shown in Fig. 8. The argument shows three principal sub-goals, to the overall objective of *Best possible system is produced*, each related to one of the basic components represented in Fig. 7. The first two goals are aimed at having a suitable mechanism for taking the current solution and adapting it. The concern here is the adaption mechanism results in a suitable design within a reasonable amount of time which is influenced by the number of times we have to go round the loop. The final goal is concerned with how we assess the quality of the solution. It is this goal we concentrate on here. Figs. 9–13 show how the goal is developed.

Fig. 9 presents the decomposition of the goal, *G*1, which is considered to be the system meets its requirements in a cost effective manner. The purpose of this leg of the argument is to consider the range of design choices and means of evaluating task allocations. In contrast the goals *Allocation* and *Assignment* are concerned with how fast and well the chosen search algorithm operates. In the context of this work it is assumed, *A*1, that the work is to concentrate on the timing properties of the system. Context *C*1 also establishes that in terms of cost the argument is concerned with

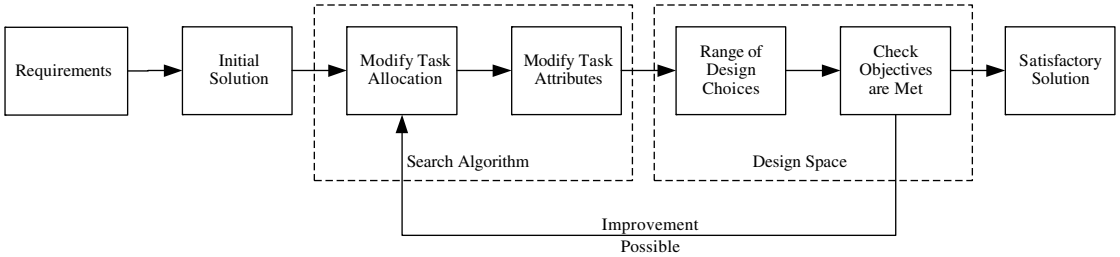

Fig. 7. Top level design.

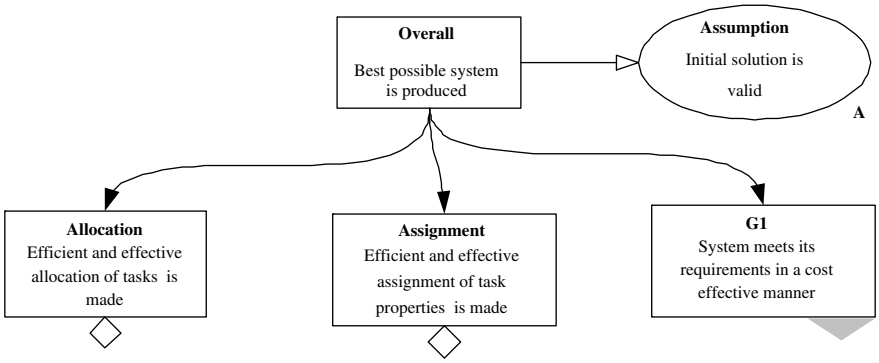

Fig. 8. Top-level argument for overall system objective.

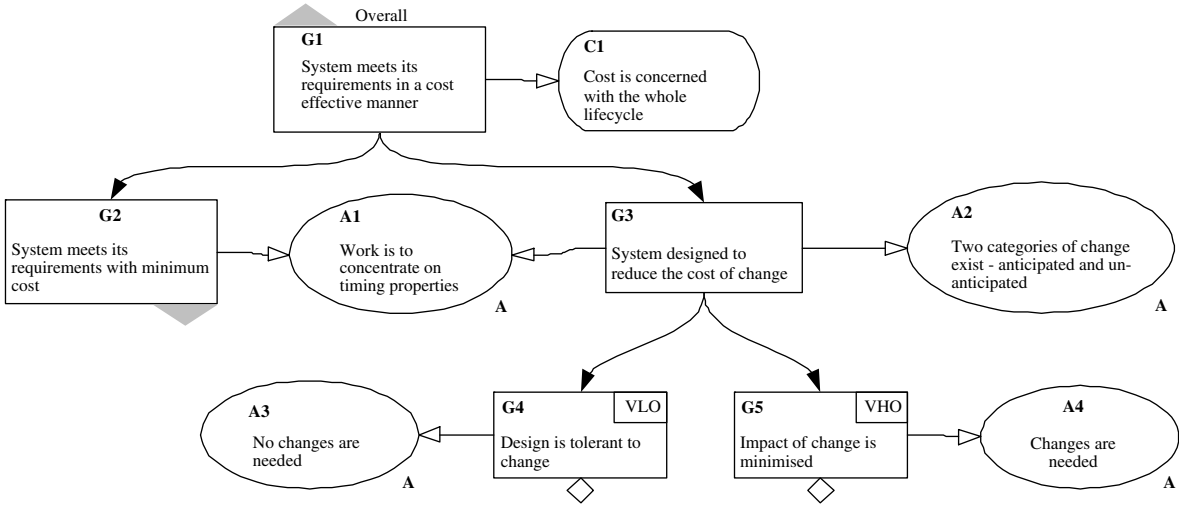

Fig. 9. Top-level argument for objectives met.

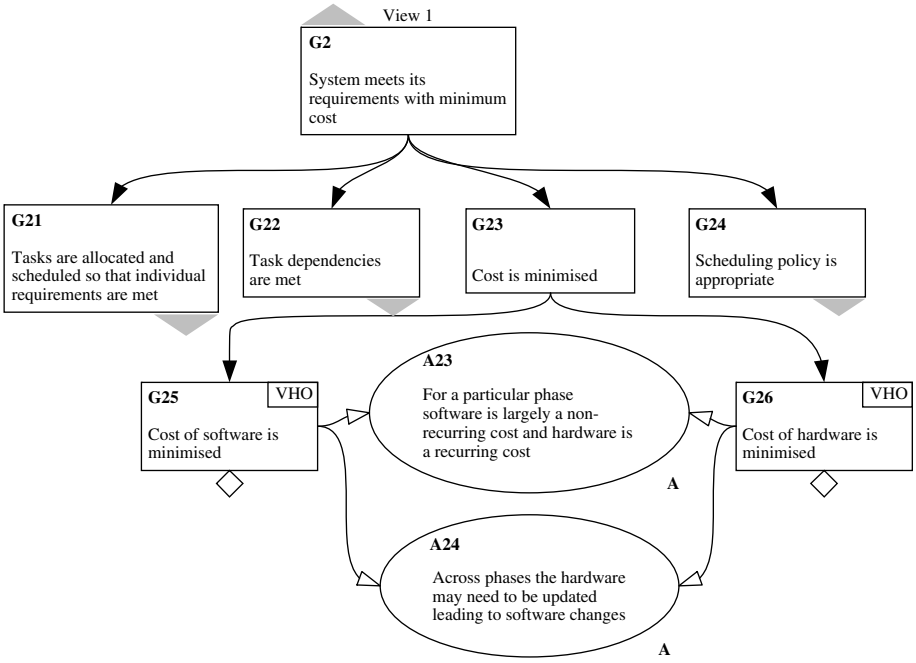

Fig. 10. Trade-offs related to minimum cost.

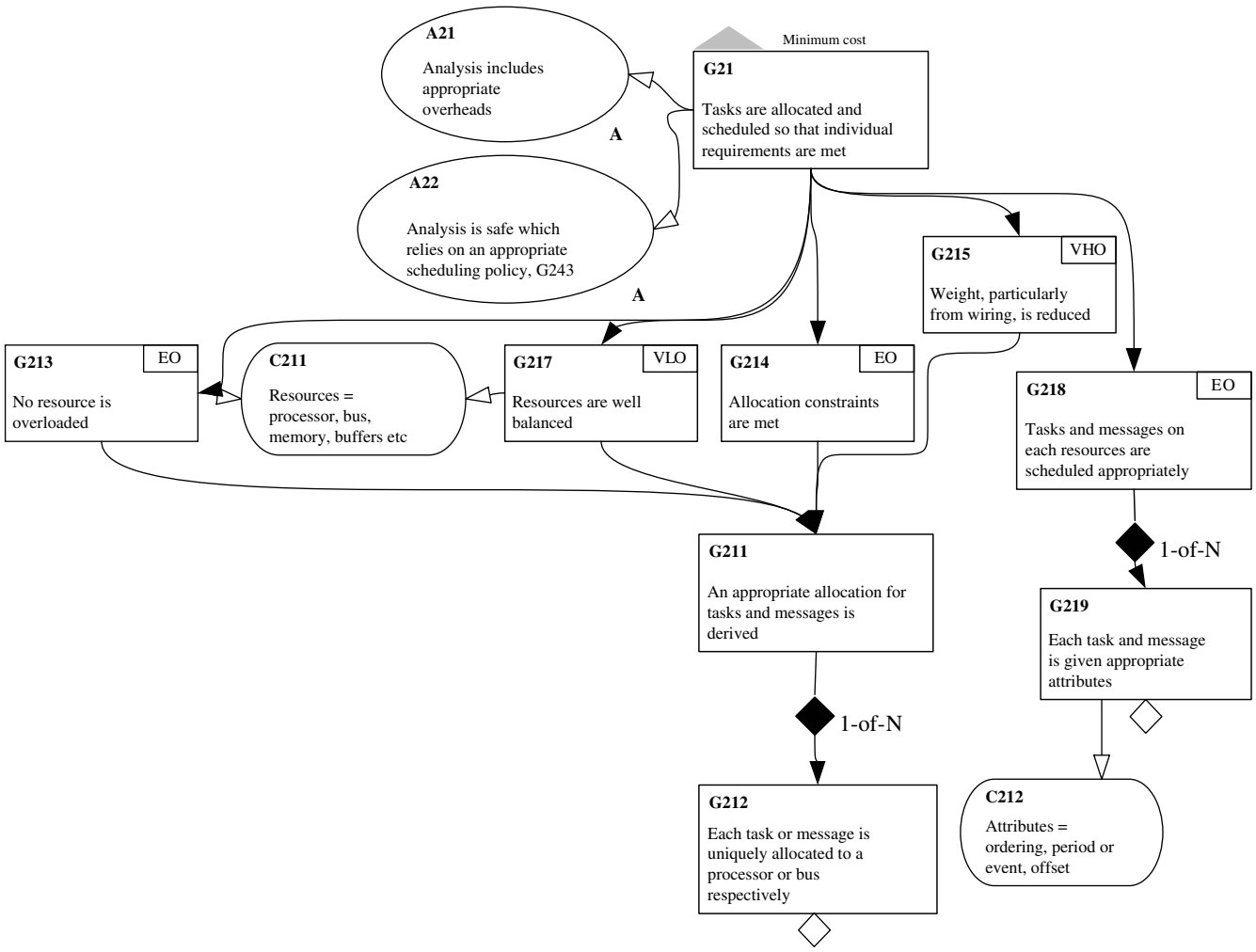

Fig. 11. Trade-offs related to individual requirements.

that of the whole lifecycle, i.e., from project inception all the way to de-commissioning and disposal.

The top-level goal is then split into two distinct arguments. First, the system meets its requirements with minimal cost, *G*2, and second the cost of change is minimised, *G*3. The assumption *A*2 clarifies that changes are considered to fall into two categories; anticipated and un-anticipated. This assumption recognises the important fact that in the development of embedded systems many of the changes to be performed are not to fix errors but instead planned as part of a phased development program. Whilst it is recognised that changes cannot be predicted with 100% accuracy, other work (Bate and Emberson, 2005; Bate and Audsley, 2004) has shown that using scenarios to introduce flexibility helps reduce the impact of change even if the scenarios differ from the actual changes introduced.

The goal *G*3 is further separated into two parts. First, goal *G*4 introduces the objective that the design should be tolerant to change. Then, goal *G*4 deals with the situation where a change is necessary but its cost should be minimised. These goals are not expanded further as they are measurable quantities.

The only goal from Fig. 9 that needs further expansion is goal *G*2 which is concerned with the system meeting its requirements with minimum cost. A key assumption *A*1 is made that in the context of this work we are to concentrate on the timing properties of systems. This goal is expanded further in Fig. 10. Fig. 10 shows how goal *G*2 can be decomposed into four sub-goals – *G*21, *G*22, *G*23 and *G*24. The four sub-goals deal with individual requirements, task dependencies, cost and scheduling policy separately. In Fig. 10, goal *G*23 is decomposed so that minimal cost is separated into hardware and software, respectively. These goals are now considered to be measurable. The goal *G*26 represents an interesting objective from a project lifetime perspective as on the surface its assessment is simple, i.e., the sum of the unit costs for the hardware including the manufacturing and assembly costs. However, allowing for the bigger picture the hardware costs have to allow for the potential problems of scalability and obsolescence. That is, the costs should allow for the risk that changes will be needed to the hardware to accommodate future expansion of the software and that the hardware goes obsolete necessitating a re-design of the system including its software.

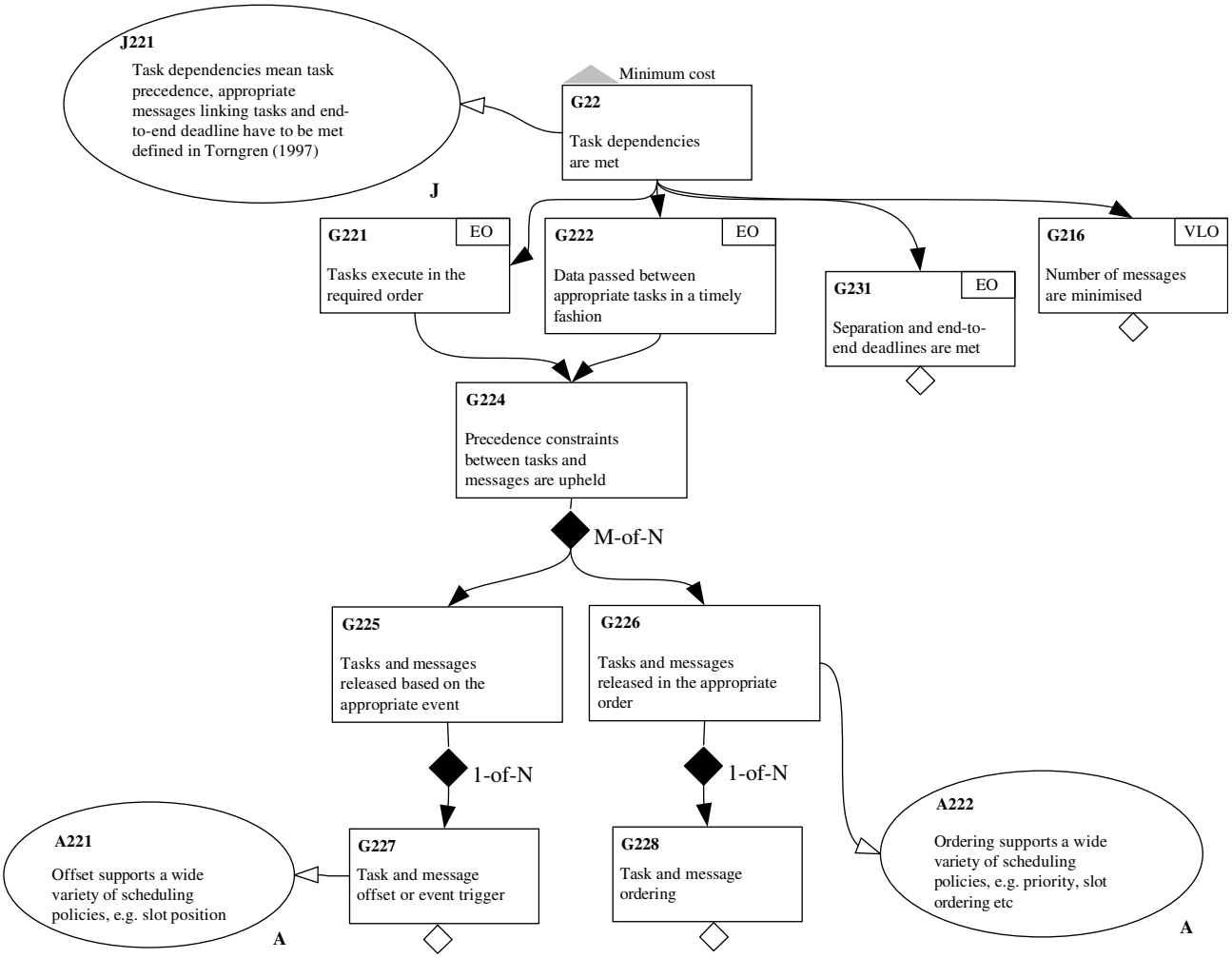

Fig. 12. Trade-offs related to task dependencies.

As such there is a strong link between goals *G*25 and *G*26 which is reflected in the assumptions *A*23 and *A*24. Capturing key points such as these is a key benefit of using a systematic structured approach to argumentation.

Of the un-expanded goals in Fig. 10, goal *G*21 is expanded further in Fig. 11. Here, goal *G*21 is split into two sub-goals that deal with the problems of allocation and attribute assignment separately. Goal *G*214 is then further split into a number of sub-goals *G*213–*G*216 which deal with a variety of objectives. For example, goal *G*213 deals with resources should not be overloaded. The sub-goals are then satisfied by the design choice of exactly which processor or databus that each task or message is allocated to. Similarly, goal *G*216 is handled by a design choice over the exact value of the task and messages attributes assigned. As no scheduling policy has been defined yet, general purpose attributes are used as detailed in context *C*212.

Goal *G*22 from Fig. 10 is expanded in Fig. 12. The goal is decomposed into three sub-goals which are the tasks that form the transaction are executed in the required order (goal *G*221), there is a means of passing data between communicating tasks (goal *G*222) and timing aspects of the requirements are met (goal *G*231). A justification, *J*221, is added here that this definition of task dependencies is appropriate based on existing papers (Torngren, 1998). Goal *G*231 does not need further decomposition as it is already measurable. Goals *G*221 and *G*222 are measurable but are decomposed further to represent the set of design choices (i.e., release and scheduling attributes) that are relevant.

In Fig. 13 goal *G*24 is expanded. In this figure, the goal is decomposed in such a way that considers whether the scheduling policy is efficient (goal *G*241), effective (goal *G*242) and safe (goal *G*243). Goals *G*242 and *G*243 are decomposed further with goal *G*245 being decomposed to represent the range of scheduling policies that may be used (note the set of policies is not considered complete) and goal *G*243 is split into whether the scheduling policy is predictable (*G*249) and whether shared resources are protected (Goal *G*250). Goal *G*250 is further decomposed to consider the design choice of using off-line or on-line mechanisms. Assumption *A*242 shows how objectives in different arguments may be linked and hence trade-offs between

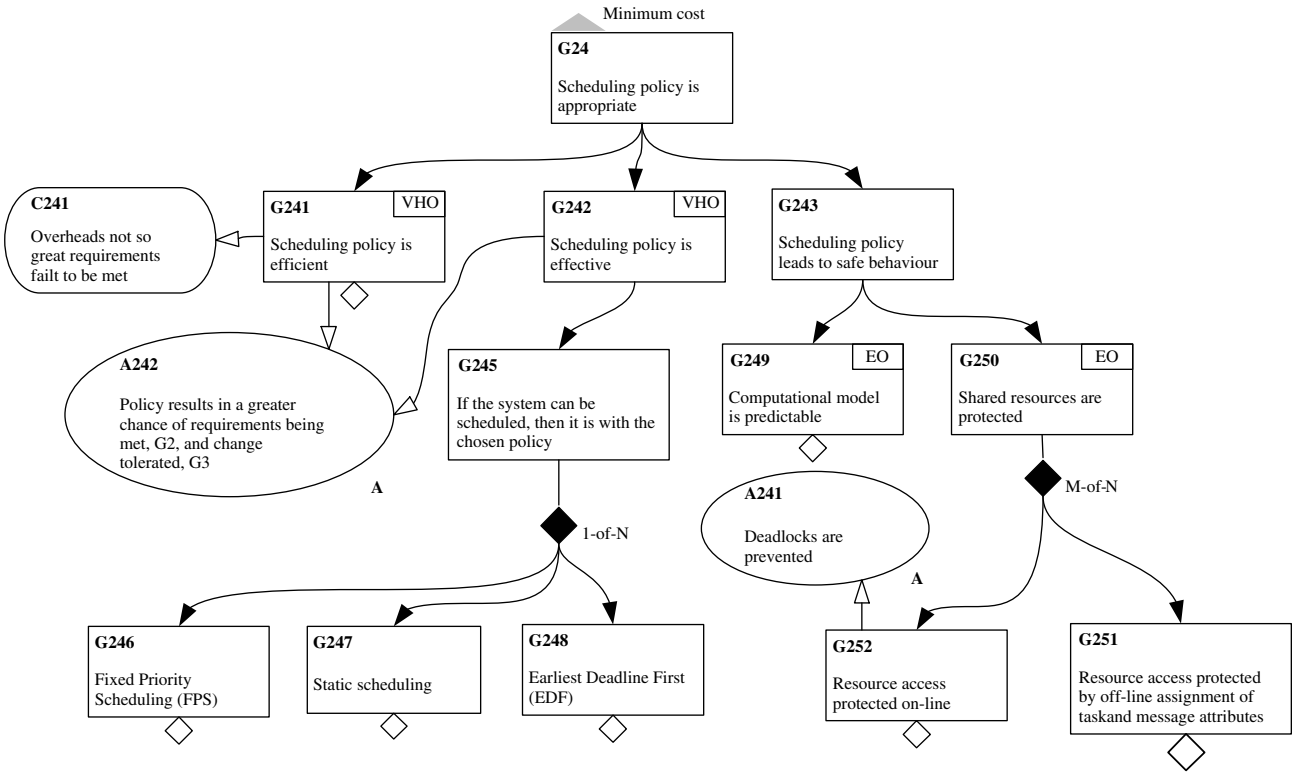

Fig. 13. Trade-offs related to scheduling policies.

properties formally established. Other trade-offs will still exist, e.g., having more processors may improve scalability but adds weight/cost, but these may not be formally stated in the arguments.

The problem is now considered decomposed in sufficient depth as all goals have reached the stage that they can be measured and the majority have a number of design choices available.

5.4. Stage 3 – Extracting information from the argument

In Section 5.3 a great deal of discussion has already taken place concerning the information contained in the argument, e.g., design choices. Rather than repeat this information, the purpose of this section is to summarise the relevant information contained within the arguments.

5.4.1. Stage 3(a) – Extracting design choices

The following are the choices that emerge relating to the design:

1. From goal *G*211 emerges the choice of where each task or message is allocated.
2. A choice of task and message attributes results from the decomposition of goal *G*216.
3. In Fig. 12 there are choices related to how tasks and messages are released. It is noted this is an example of a hierarchical design choice where a higher level decision then influences the lower level choices that are available. The first choice comes from goal *G*224 which is related to how task precedence is maintained. There are two options; through the use of higher-level scheduling mechanisms such as priorities or slot position, or at the individual task level. Depending on this higher level choice, there are lower level choices. That is, a choice of priorities or slot position for the scheduling level approach, or a choice of release mechanisms (and corresponding attributes) if the objective is handled at the individual task level. It is noted these decisions are related to the earlier goal *Assignment* from Fig. 8.
4. The final choices are taken from Fig. 13. These choices are related to the type of scheduling policy used (from goal *G*245) and how resources are protected from failures (from goal *G*250).

5.4.2. Stage 3(b) – Extracting evaluation criteria

The arguments and Section 5.3 highlighted a number of objectives to be evaluated. These are all signified by the goals having either *EO*, *VHO* or *VLO* in the top right hand corner. These are represented in Table 1.

5.4.3. Stage 3(c) – Evaluating the design

The design of the system is assessed in two parts: scenario-based assessment that deals with a variety of test cases encompasses specific situations (failure and non-failure) the system must handle but also covering the ability to handle future requirements (i.e., changes); and general

Table 1
Objectives for the system

Id	Objective
*G*4	Design is tolerant to change
*G*5	Impact of change is minimised
*G*25	Cost of software is minimised
*G*26	Cost of hardware is minimised
*G*213	No resource is overloaded
*G*217	Resources are well balanced
*G*214	Allocation constraints are met
*G*215	Weight, particularly from wiring, is reduced
*G*218	Tasks and messages on each resource are scheduled appropriately
*G*221	Tasks execute in the required order
*G*222	Data passed between appropriate tasks in a timely manner
*G*231	Separation and end-to-end deadlines are met
*G*216	Number of messages are minimised
*G*241	Scheduling policy is efficient
*G*242	Scheduling policy is effective
*G*249	Computational model is predictable
*G*250	Shared resources are predicted

analysis of the current design's ability to meet its objectives. To illustrate the difference consider a timing analysis. A scenario may be the ability to handle a processor failing or an execution time being increased by 20% larger, whereas the general analysis would using timing analysis to show all the timing requirements are met. These two forms of assessment are considered in the following sections. As far as the overall evaluation is concerned, there is a need to combines the results from each of these stages. This is done using Eq. (1) that takes the results of the individual evaluations and combine them in a weighted normalised fashion.

$$f_o = \frac{(w_g \cdot f_g) + (w_s \cdot f_s)}{w_g + w_s}, \tag{1}$$

where f_o is the overall fitness value for the system, f_g is the result of the general analysis of the current design, f_s is the result of the scenario-based assessment, w_g is the weighting for the results of the general analysis and w_s is the weighting for the results of the scenario-based assessment.

5.4.4. Stage 3(d) – Scenario-based assessment

There are two goals, *G*4 and *G*5 in the arguments considered difficult to quantify as they deal with how the system may change through the development lifecycle. However, through the use of scenarios changes can be applied to the system (e.g., the worst-case execution time of a task could be increased) and an assessment made of whether any changes are needed to the task allocation in order to solve the new problem. In Bate and Emberson (2006), it is shown how taking this approach can be used to assess the flexibility of systems to change and use this information as part of the design process to improve flexibility.

Eq. (2) assesses the tolerance of the design to change, i.e., if any changes to the design solution is needed to cope with the change then *zero* is returned otherwise *one* is returned. In Eq. (3), *tac* is the number of task allocation changes, *mac* is the number of message allocation changes, *tpc* is the number of task ordering changes and mpc is the number of message ordering changes. A weighting factor, α_N, is applied to each change type in order to reflect the relative importance of each type of change before the values are combined. The result from Eq. (3) is a measure for the cost required for a particular change to the design solution. Clearly if the value returned by Eq. (2) is zero then the value given by Eq. (3) will also be zero. Each of the two equations can be assessed for each individual change considered. In our previous work, (Bate et al., 2003; Bate et al., 2003; Bate and Emberson, 2006), a change is considered a scenario and for each potential design a number of scenarios are applied. An individual design can be assessed with respect to a number of scenarios using Eqs. (4) and (5). These equations effectively show two levels of assessment which possibly feature more than one level of importance. That is, the first equation (Eq. (4)) is of higher level, and more important, as it signifies whether the change is handled appropriately. In contrast, the second equations give a detailed evaluation of how large a change is needed. The overall result from the scenario-based assessment is given in Eq. (6) where w_{cost} and w_{tol} are the weightings for the cost and tolerance, respectively.

$$\text{tolerance\_cost\_changes} = \begin{cases} 1 & \text{if changes} > 0, \\ 0 & \text{if changes} = 0, \end{cases} \tag{2}$$

$$\text{change\_cost} = \frac{\alpha_1 \cdot \text{tac} + \alpha_2 \cdot \text{mac} + \alpha_3 \cdot \text{tpc} + \alpha_4 \cdot \text{mpc}}{\alpha_1 + \alpha_2 + \alpha_3 + \alpha_4}, \tag{3}$$

$$\text{overall\_tolerance} = \sum_{\forall \text{scenarios}} \text{tolerance\_cost\_changes}, \tag{4}$$

$$\text{overall\_cost} = \sum_{\forall \text{scenarios}} \text{change\_cost}, \tag{5}$$

$$f_s = \frac{(w_{cost} \cdot \text{overall\_cost}) + (w_{tol} \cdot \text{overall\_tolerance})}{w_{cost} + w_{tol}}. \tag{6}$$

5.4.5. Stage 3(e) – Baseline assessment

There are a number of arguments with quantifiable objectives related to the baseline design. The first argument to be considered is Fig. 11. This argument essentially deals with whether the individual tasks are allocated and scheduled so that their timing requirements are met and to help support other physical properties of systems. The objectives within the argument that can be quantitatively measured are discussed below. The final value obtained is *overload*.

1. *G213:* This objective is concerned with whether any resource, e.g., processor or network, is overloaded. For most systems overload can be considered when the utilisation is greater than 100%. However, for some systems the designers are contracted to provide spare resource to support a growth margin and provide

flexibility for changes. The utilisation of a resource can be measured using Eq. (7), where i is the set of services hosted on the resource. A fitness function can then be formed over the set of resources, j, using Eqs. (8) and (9), where U_j^{bound} is the upper bound on the utilisation allowed and and $\#j$ is the number of resources. It is noted that many of the quantitative measures could be made in ways that give different degrees of information. For instance, a form of Eq. (8) could be altered to measure by how much the utilisation bound is not met rather than a simple *yes*/*no* answer.

$$U_j = \sum_{\forall i} \frac{C_i}{T_i}, \tag{7}$$

$$U_j^{\text{met}} = \begin{cases} 1 & \text{if } U_j > U_j^{\text{bound}}, \\ 0 & \text{otherwise}, \end{cases} \tag{8}$$

$$f_{\text{overload}} = \sum_{\forall j} \frac{U_j^{\text{met}}}{\#j}. \tag{9}$$

2. *G217:* This objective is concerned with whether each of the available resources has a similar utilisation. The purpose of this is to help make each resource equally difficult to schedule and equally likely to be scalable. Eq. (10), where $U_{\min}$ is the utilisation of the least lowest resource, shows how the objective could be assessed.

$$f_{\text{balance}} = \sum_{\forall j} \frac{U_j - U_{\min}}{\#j}. \tag{10}$$

3. *G214:* This objective is concerned with whether allocation constraints are met or not. An allocation constraint can be of two forms. First, two tasks or messages may be prohibited from being resident on the same processor. This may be the case when the tasks or messages are replicas used for fault tolerance in which case there is little point in both being on the same resource as a resource failure would be a common cause failure. The second case is where a specific task or message needs to be allocated to a specific resource, e.g., if a task was reading a sensor value and there was a limit on the physical length of the wiring. This can be assessed simply by checking what percentage of the constraints are met as shown in Eq. (11).

$$f_{\text{alloc\_const}} = \frac{\sum_{\forall \text{constraints}} \begin{pmatrix} 1 & \text{if constraint met} \\ 0 & \text{otherwise} \end{pmatrix}}{\#\text{constraints}}. \tag{11}$$

4. *G215:* This objective is concerning the weight of the resources used in the solution. Weight can be reduced in two ways. First, by using less resources, e.g., processors, which tends the solution to hardware efficiency. Second, by reducing the amount of cabling needed. This second aim can be achieved in two main ways; not using a fully connected network, and by physically locating the processors appropriately. The weight can simply be assessed by adding up the weight of all resources and cabling. In some applications, more advanced forms of assessment may be needed. For example, in aircraft applications centre of gravity (CoG) is also important.
5. *G218:* The final objective is whether the individual timing requirements are met. This can be assessed straightforwardly using Eq. (12). As discussed earlier, a step function may be applied to the objective so that the difference between a 100% successful solution and a partially successful solution is clear.

$$f_{\text{ind\_req}} = \frac{\sum_{\forall \text{individual\_requirements}} \begin{pmatrix} 1 & \text{if requirement met} \\ 0 & \text{otherwise} \end{pmatrix}}{\#\text{individual\_requirements}}. \tag{12}$$

The argument whether dependent tasks are adequately handled is given in Fig. 12. This argument essentially deals with whether the fundamental requirements are met at the same time the number of messages is reduced. The objectives within the argument can be assessed in the following way:

1. *G221, G222 and G231:* These objectives can be collectively described as whether the timing requirements are met. This can be assessed straightforwardly using Eq. (13). As discussed earlier, a step function may be applied to the objective so that the difference between a 100% successful solution and a partially successful solution is clear. If the different categories of requirements (i.e., precedence and end-to-end deadline requirements) had different levels of importance then Eq. (13) could be split further with each category being given a different weighting which equate to importance.

$$f_{\text{dep\_req}} = \frac{\sum_{\forall \text{dependency\_requirements}} \begin{pmatrix} 1 & \text{if requirement met} \\ 0 & \text{otherwise} \end{pmatrix}}{\#\text{dependency\_requirements}}. \tag{13}$$

2. *G216:* The final objective for this argument is that the number of messages are reduced. That is, tasks with dependency requirements tend to be co-located on the same processor. There are two benefits of taking this strategy; the utilisation of the inter-processor communications is reduced and schedulability is improved. However, it is noted that this objective may conflict with other objectives such as the need for certain tasks to be allocated on specific processors or for pairs of tasks not be co-located. This leads to a need for a careful balancing act within the trade-offs performed.

The final argument shown here, Fig. 13, is concerned with whether an appropriate choice of scheduling policy has been taken. The objectives within the argument can be assessed in the following way. It is noted that the scheduling policy is often a fixed design variable and therefore would not be considered as part of the trade-off analysis.

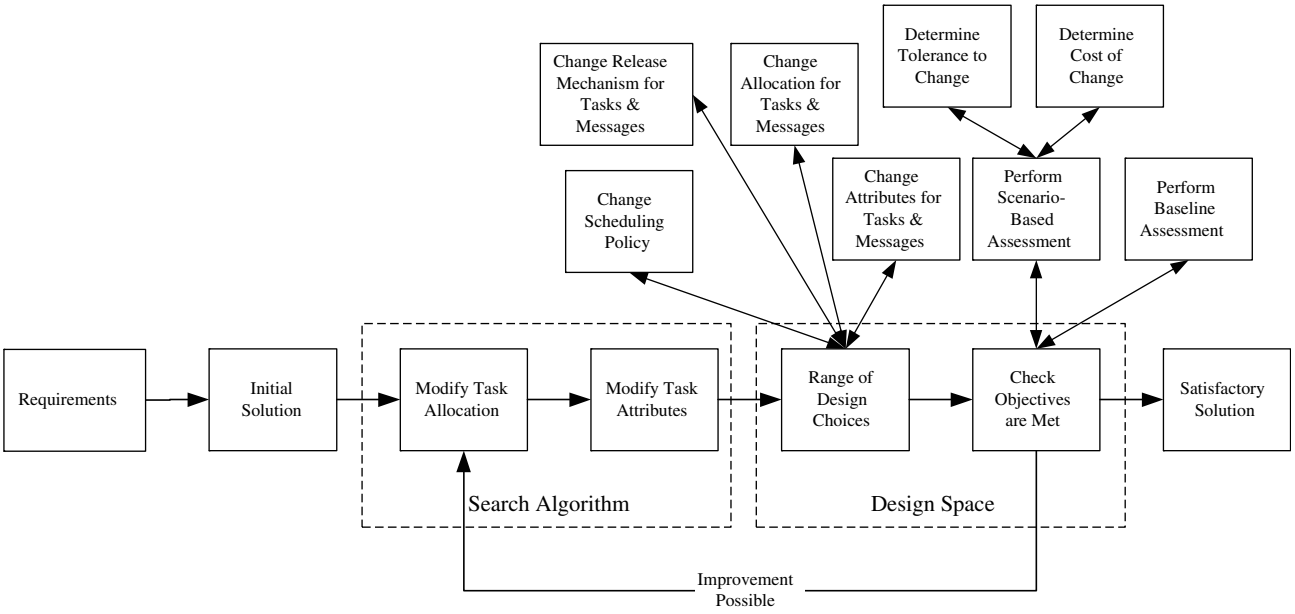

Fig. 14. Decomposed design.

1. *G241:* This objective considers whether the scheduling policy is efficient, i.e. whether it has low overheads. Having low overheads can make it more likely that the system is schedulable assuming the policy chosen is effective. The balance between the complexity/optimality of the scheduling policy and the amount of run-time overheads is an issue that should not be ignored when designing systems. Another benefit of low overheads is that more of the systems resources would be used by "useful" functionality. The objective can be assessed using Eq. (7) to determine the amount of run-time overheads.
2. *G242:* As discussed above, an important objective is that the scheduling policy chosen is effective. The main whether of measuring this objective is by judging whether all the timing requirements of the system are met. The objective can be assessed by returning a value of '1' if they are, otherwise a value of '0' is returned. Using such an objective represents an alternative way of providing a discriminating factor for whether objectives relating to requirement being met, e.g., *G218* are completely satisfied without the need for a bonus factor, e.g., step function.
3. *G249:* Many of the other objectives rely on there being suitable analysis that allows a quantitative assessment of whether the requirements are met. Therefore this objective is simply concerned with whether the scheduling policy is predictable and hence analysis is possible. The objective can be assessed by returning a value of '1' if it is possible, otherwise a value of '0' is returned.
4. *G250:* The final objective is whether the scheduling mechanism used provides adequate protection where needed, e.g., for shared resources. Again, the objective can be assessed by returning a value of '1' if it is possible, otherwise a value of '0' is returned.

An overall value, f_g, for the fitness of the baseline design can be found using Eq. (14), where w_i are the individual weightings for the baseline objectives.

$$f_g = \frac{\sum_{\forall i \in \text{baseline\_objectives}} f_i w_i}{\sum_{\forall i \in \text{baseline\_objectives}} w_i}. \tag{14}$$

5.5. Stage 4 – Decomposing the design

In Fig. 7 the top-level design of the system is presented. Based on the method presented in this paper, the next level of design is given in Fig. 14 that shows how *Design Space*, which contains *Range of Design Choices* and *Check Objectives are Met is handled*. The figure clearly shows the set of design choices and objective measures established. It is noted the *Evaluate Baseline Design* is not expanded for reasons of clarity. One key issue the designer might make at this stage is to fix certain design choices to limit the search space. An obvious example is the scheduling policy that should not be continually changing. At this stage the designer can decide whether to carry on iterating around the design using this method or proceed to implementation. As part of this decision process, it is reasonable to only apply the method to crucial, or difficult, parts of the design.

6. Conclusions and summary

In this paper, we have motivated the need for more systematic approaches to understanding the trade-offs within the design of systems. The specific contributions of the paper are to show how our approach provides the degree of rigour suitable for critical systems applications at the same time as capturing the rationale and justifications to

help support maintainability of systems in general. The approach presented is based on a well-established approach to safety argumentation that has been adapted for the purpose of building design arguments and then extracting the relevant information needed, i.e., design choices and assessment criteria. The assessment criteria are then converted to a quantifiable measure and an appropriate weighting applied. During the course of the paper a number of ways have been demonstrated for evaluating the assessment criteria and clear relationships (trade-offs) between objectives and assessment criteria demonstrated. Finally, a case study, the task allocation problem, is used to demonstrate the approach. Open research problems include deciding which parts of the design problem the method should be applied to and to what depth, and how to derive weighting in a more systematic manner.

Acknowledgements

I would like to thanks a number of colleagues who have cooperated with earlier work in this area including Neil Audsley, Paul Emberson, Tim Kelly and Peter Nightingale.

References

Anton, A., 1996. Goal-based requirements analysis. In: Proceedings of the 2nd International Conference on Requirements Engineering (ICRE'96), pp. 136–144.

Anton, A., Potts, C., 1998. The use of goals to surface requirements for evolving systems. In: Proceedings of the 20th International Conference on Software Engineering, pp. 157–166.

Audsley, N., Burns, A., Davis, R., Tindell, K., Wellings, A., 1995. Fixed priority pre-emptive scheduling: an historical perspective. Real-Time Systems 8 (2–3), 173–198.

Axelsson, J., 1996. 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.

Bass, L., Clements, P., Kazman, R., 2003. Software Architecture in Practice. Addison-Wesley, Reading, MA.

Bate, I., Audsley, N., 2004. Flexible design of complex high-integrity systems using trade offs. In: Proceedings of the 8th IEEE International Symposium on High Assurance Systems Engineering, pp. 22–31.

Bate, I., Burns, A., 2003. An integrated approach to scheduling in safety-critical embedded control systems. Real-Time Systems Journal 25 (1), 5–37.

Bate, I., Emberson, P., 2005. Design for flexible and scalable avionics systems. In: Proceedings of the IEEE Aerospace Conference, No. 9.0101.

Bate, I., Emberson, P., 2006. Incorporating scenarios and heuristics to improve flexibility in real-time embedded systems. In: Proceedings of the 12th IEEE Real-Time and Embedded Technology and Applications Symposium, pp. 221–230.

Bate, I., Kelly, T., 2002. Architectural considerations in the certification of modular systems. In: Proceedings of the Computer Safety, Reliability and Security – 21st International Conference, SAFECOMP 2002, volume LNCS 2434, pp. 321–333.

Bate, I., Kelly, T., 2003. Architectural considerations in the certification of modular systems. Reliability Engineering and System Safety 81, 303–324.

Bate, I., McDermid, J., Nightingale, P., 2003. Establishing timing requirements for control loops in real-time systems. Microprocessors and Microsystems 27 (4), 159–169.

Bate, I., Cervin, A., Nightingale, P., 2003. Establishing timing requirements and control attributes for control loops in real-time systems. In: Proceedings of the 15th Euromicro Conference on Real-Time Systems, pp. 121–128.

Brooks, D., Tiwari, V., Martonosi, M., 2000. Wattch: a framework for architectural-level power analysis and optimizations. In: Proceedings of the 27th International Symposium on Computer Architecture (ISCA), pp. 83–94.

Burns, A., Davis, R., Punnekat, S., 1996. Feasibility analysis of fault-tolerant real-time task sets. Euromicro Real-Time Systems Workshop (June), 29–33.

CENELEC. IEC 61508 Functional Safety of electrical/electronic/programmable electronic safety-related systems, 2001.

Chung, L., Nixon, B., Yu, E., Mylopoulos, J., 1999. Non-functional Requirements in Software Engineering. Kluwer Academic Publishers, Dordrecht.

Coello Coello, C., 1999. A comprehensive survey of evolutionary-based multiobjective optimization techniques. Knowledge and Information Systems 1 (3), 129–156.

Dobrica, L., Niemela, E., 2002. A survey on software architecture analysis methods. IEEE Transactions on Software Engineering 28 (7), 638–653.

Fohler, G., Koza, C., 1989. Heuristic scheduling for distributed real-time systems. Technical Report No. 6, Institut fur Technische Informatik, Technische Universtate Wien, Austria.

Herrmann, D., 2000. Software Safety and Reliability: Techniques, Approaches, and Standards of Key Industrial Sectors. Wiley, New York.

Kazman, R., Bass, L., Webb, M., Abowd, G., 1994. SAAM: A method for analyzing the properties of software architectures. In: International Conference on Software Engineering, pp. 81–90.

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

Kelly, T., 1999. Arguing Safety – A Systematic Approach to Safety Case Management. Ph.D. Thesis, Department of Computer Science, University of York.

Kogure, M., Akao, Y., 1993. Quality function deployment and cwqc in japan. Quality Progress, 25–29.

Lee, J., 1991. Extending the potts and bruns model for recording design rationale. In: Proceedings of the 13th International Conference on Software Engineering, pp. 114–125.

McDermid, J., 2001. Software safety: Where's the evidence? In: Proceedings of the Sixth Australian Workshop on Industrial Experience with Safety Critical Systems and Software, vol. 3 of Conferences in Research and Practice in Information Technology Series, pp. 1–6.

Moore, M., Kazman, R., Klein, M., Asundi, J., 2003. Quantifying the value of architecture design decisions: Lessons from the field. In: Proceedings of the 25th International Conference on Software Engineering (ICSE 25).

Mylopoulos, J., Chung, L., Nixon, B., 1992. Representing and using nonfunctional requirements: a process-oriented approach. IEEE Transactions on Software Engineering 18 (6), 483–497.

Nicholson, M., 1998. Selecting a topology for safety-critical real-time control systems. DPhil Thesis, Department of Computer Science, University of York, YCST-98-08.

Nord, R., Barbacci, M., Clements, P., Kazman, R., Klein, M., O'Brien, L., Tomayko, J., 2003. Integrating the Architecture Tradeoff Analysis Method (ATAM) with the Cost Benefit Analysis Method (CBAM). Technical Report CMU/SEI-2003-TN-038, Software Engineering Institute.

Rayward-Smith, V., Osman, I., Reeves, C., Smith, G. (Eds.), 1996. Modern Heuristic Search Methods. Wiley, New York.

Rombach, H., Basili, V., 1988. The TAME project: Towards improvement-oriented software environments. IEEE Transactions on Software Engineering 14 (6), 758–773.

RTCA Inc. Software considerations in airborne systems and equipment certification. DO-178B/ED-12B, December 1992.

Tahvildari, L., Kontogiannis, K., Mylopoulos, J., 2003. Quality-driven software re-engineering. Journal of Systems and Software (66), 225–239.

Torngren, M., 1998. Fundamentals of implementing real-time control applications in distributed computer systems. Real-Time Systems 14 (3), 219–250.

United Kingdom Ministry of Defence. Defence Standard 00-55: Requirements for Safety-Related Software in Defence Equipment, July 1996.

United Kingdom Ministry of Defence. Defence Standard 00-56 (Issue 2): Safety Management Requirements for Defence Systems, December 1996.

United Kingdom Ministry of Defence. Defence Standard 00-56 (Issue 3): Safety Management Requirements for Defence Systems, December 2004.

Villmeur, A., 1992. Reliability, Availability, Maintainability, and Safety Assessment. Methods and Techniques, vol. 1. Wiley, New York.