Architecture Trade-off Analysis and the Influence on Component Design

Iain Bate and Neil Audsley
Department of Computer Science
University of York, York, YO10 5DD, UK.
{iain.bate, neil.audsley}@cs.york.ac.uk

1 Introduction

The production and assurance of systems that are safety-critical and/or real-time is recognised as being costly, time-consuming, hard to manage, and difficult to maintain. This has lead to research into new methods whose objectives include:

- Modular approaches to development, assurance and maintenance to enable:
  - Increased reuse;
  - Increased robustness to change and reduced impact of change.
- Integration strategies that allow systems to be procured and produced by multiple partners, and then efficiently integrated;
- Ways of determining the approach likely to be the "best" (the best can only be found with hindsight);
- Techniques for identifying and managing risks.

Many of the component-based engineering techniques are considered relatively mature for developing dependable components and ensuring correctness across their interfaces when combined with other components, e.g. approaches based on rely-guarantees [1]. This paper addresses the following key remaining issues:

- how the system's objectives should be decomposed and designed into components (i.e. the location and nature of interfaces); and
- what functionality the components should provide to achieve the system's objectives.

The paper develops a method for:

1. *derivation of choices* – identifies where different design solutions are available for satisfying a goal.
2. *manage sensitivities* – identifies dependencies between components such that consideration of whether and how to relax them can be made. A benefit of relaxing dependencies could be a reduced impact to change.
3. *evaluation of options* – allows questions to be derived whose answers can be used for identifying solutions that do/do not meet the system properties, judging how well the properties are met and indicating where refinements of the design might add benefit.
4. *influence on the design* – identifies constraints on how components should be designed to support the meeting of the system's overall objectives.

Our approach satisfies the objectives by building on existing approaches, i.e. Goal Structuring Notation (GSN) which is used for safety arguments [2], and UML which is used for modelling systems [3]. However, a key fact is that the methodology proposed is not dependent on the specific techniques advocated. The approach only needs a component -based model of the system's design with data flow coupling between the components and there being a way of reasoning about properties (and their inter-relationships), constraints, and assumptions associated with the coupling. (A coupling is considered as a connection between components.) The approach we are proposing would be used within the nine-step process of the Architecture Trade-Off Analysis Method (ATAM) [4]. The differences between our strategy and other existing approaches, e.g. ATAM, include the following.

1. the techniques used in our approach are already accepted and widely used (e.g. nuclear propulsion system and missile system safety arguments) [2], and as such processes exist for ensuring the correctness and consistency of the results obtained.
2. the techniques offer strong traceability and the ability to capture design rationale.
3. information generated from their original intended use can be reused, rather than repeating the effort.
4. the method is equally intended as a design technique to assist in the evaluation of the architectural design and implementation strategy as it is for evaluating a design at a particular fixed stages of the process.

The method is described further in section 2. This is followed by a demonstration of the approach through the case study presented in section 3. Section 4 considers how the architecture trade-off analysis can be used to influence the way in which components are designed.

2 Trade-Off Analysis Method

2.1 Overview of the Trade-Off Analysis Method

Figure 1 provides a diagrammatic overview of the method. 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. Currently our work uses UML for this purpose, however it could be applied to any modelling approach that clearly identifies components and their couplings. Arguments are then produced in stage (2) for each coupling to a corresponding (but lower so that impact of later choices can be made) abstraction level

than the system model. (An overview of GSN is given in section 2.2.) The arguments are derived from the top-level properties of the particular system being developed. The properties often of interest are lifecycle cost, dependability, and maintainability. Clearly these properties might be broken down further, e.g. dependability may be decomposed to reliability, safety, timing etc.. In practice, the arguments should be generic or based on patterns where possible. Stage (3) then uses the information in the argument to derive options and evaluate particular solutions. Part of this activity uses representative scenarios (e.g. what happens when change X is performed) to evaluate the solutions. The use of scenarios is not discussed in this paper.

Based on the findings of stage (3), the design is modified to fix problems that are identified – this may require stages (1)-(3) to be repeated to show the revised design is appropriate. When this is complete and all necessary design choices have been made, the process returns to stage (1) where the system is then decomposed to the next level of abstraction using guidance from the goal structure. Components reused in other context could be incorporated as part of the decomposition. Only proceeding when design choices and problem fixing are complete is preferred to allowing trade-offs across components at different stages of decomposition because the abstractions and assumptions are consistent easing the multiple-criteria optimisation problem.

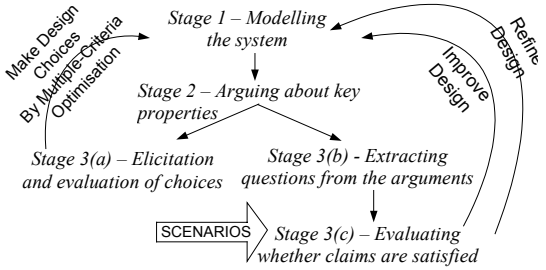

Figure 1 - Overview of the Method

2.2 Background on Goal Structuring Notation

The arguments are expressed in the GSN [2] that is widely used in the safety-critical domain for making safety arguments. In brief, any safety case can be considered as consisting of requirements, argument, evidence and definition of bounding context. GSN - a graphical notation - explicitly represents these elements and (perhaps more significantly) the relationships that exist between these elements (i.e. how individual requirements are supported by specific arguments, how argument claims are supported by evidence and the assumed context that is defined for the argument).

The principal symbols in the notation are shown in Figure 2 (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 (**assumptions**, **justifications**) and the **context** in which goals are stated (e.g. the system scope or the assumed operational role). Further details are found in [2].

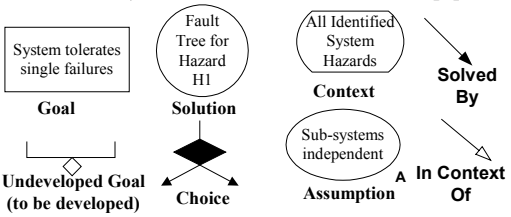

Figure 2 - Principal Elements of GSN

3 Case Study – Simple Control System

The example being considered is a continuous control loop that has health monitoring to check for whether the loop is complying with the defined correct behaviour (i.e. accuracy, responsiveness and stability) and then takes appropriate actions if it does not.

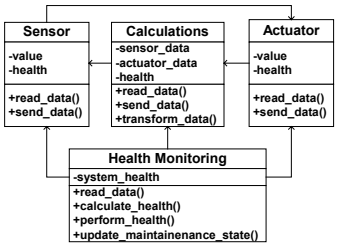

Figure 3 - Class Diagram for the Control Loop

At the highest level of abstraction the control loop (the architectural model of which is shown in Figure 3) consists of three elements; a sensor, an actuator and a calculation stage. It should be noted that at this level, the design is abstract of whether the implementation is achieved via hardware or software. The requirements (key safety properties to be maintained are signified by **(S)**, functional properties by **(F)** and non-functional properties by **(NF)**, and explanations, where needed, in italics) to be met are:

- the sensors have input limits (**S**) **(F)**;
- the actuators have input and output limits (**S**) **(F)**;
- the overall process must allow the system to meet the desired control properties, i.e. responsiveness (dependent on errors caused by latency **(NF)**), stability (dependent on errors due to jitter **(NF)** and gain at particular frequency responses **(F)**) [6] (**S**);
- where possible the system should allow components that are beginning to fail to be detected at an early stage by comparison with data from other sources (e.g. additional sensors) **(NF)**. Early recognition would allow appropriate actions to be taken including the planning of maintenance activities.

In practice as the system development progresses, the

component design in Figure 3 would be refined to show more detail. For reasons of space only the *calculation-health monitor* coupling is considered.

Stage 2 is concerned with producing arguments to support the meeting of objectives. The first one considered here is an objective obtained from decomposing an argument for dependability (the argument is not shown here due to space reasons) that the system's components are able to tolerate timing errors (goal **Timing**). From an available argument pattern, the argument in Figure 4 was produced reasoning that *"Mechanisms in place to tolerate key errors in timing behaviour"* where the context of the argument is *health monitor* component. Figure 4 shows how the argument is split into two parts. Firstly, evidence has to be obtained using appropriate verification techniques that the requirements are met in the implementation, e.g. when and in what order functionality should be performed. Secondly, the health monitor checks for unexpected behaviour. There are two ways in which unexpected behaviour can be detected (a choice is depicted by a black diamond in the arguments) – just one of the techniques could be used or a combination of the two ways. The first way is for the *health-monitor* component to rely entirely on the results of the internal health monitoring of the *calculation* component to indicate the current state of the calculations. The second way is for the *health-monitor* component to monitor the operation of the *calculation* component by observing the inputs and outputs to the *calculation* component.

In the arguments, the leaf goals (generally at the bottom) have a diamond below them that indicates the development of that part of the argument is not yet complete. An argument is complete when all leaves have been fully developed such that they are terminated by solutions. The solutions are typically requirements for the evidence to be provided. The evidence provided is normally quantitative in nature, e.g. results of timing analysis to show timing requirements are met.

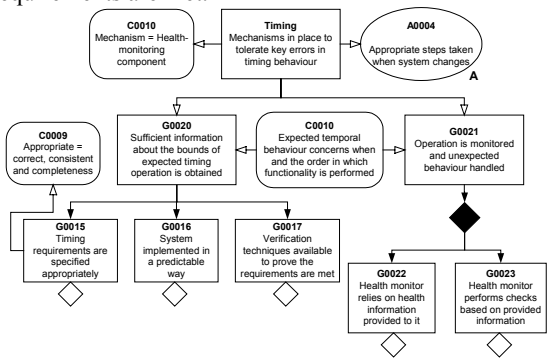

Figure 4 - Timing Argument

Next an objective obtained from decomposing an argument for maintainability (again not shown here due to space reasons) that the system's components are tolerant to changes is examined. The resultant argument in Figure 5 depicts how it is reasoned the *"Component is robust to changes"* in the context of the *health-monitor* component. There are two separate parts to this; making the integrity of the calculations less dependent on when they are performed, and making the integrity of the calculations less dependent on the values received (i.e. error-tolerant). For the first of these, we could either execute the software faster so that jitter is less of an issue, or we could use a robust algorithm that is less susceptible to the timing properties of the input data (i.e. more tolerant to jitter or the failure of values to arrive).

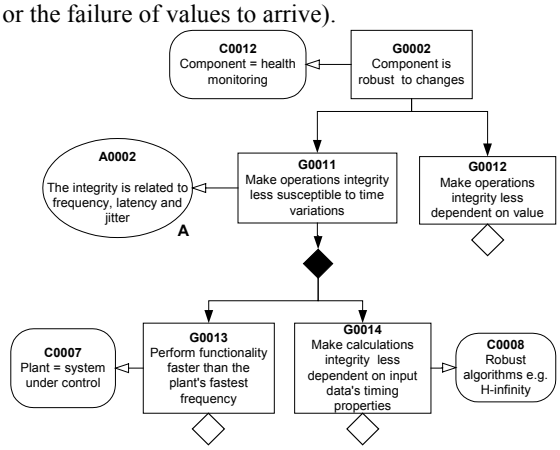

Figure 5 – Minimising Change Argument

The next stage (stage 3(a)) in the approach is the elicitation and evaluation of choices. This stage extracts the choices, and considers their relative pros and cons. The results are presented in Table 1.

Content	Choice	Pros	Cons
Goal G0021 - Operation is monitored and unexpected behaviour handled	**Goal G0022 -** Health monitor relies on health information provided to it	Simplicity since health monitor doesn't need to access and interpret another component's state.	Can a failing/failed component be trusted to interpret error-free data.
	Goal G0023- Health monitor performs checks based on provided information	Omission failures easily detected and integrity of calculations maintained assuming data provided is correct.	Health monitor is more complex and prone to change due to dependence on the component.
Goal G0011 - Make operations integrity less susceptible to time variations	**Goal G0013 –** Perform functionality faster than the plant's fastest frequency.	Simple algorithms can be used. These algorithms take less execution time.	Period and deadline constraints are tighter. Effects of failures are more significant.
	Goal G0014 - Make calculations' integrity less dependent on input data's timing properties.	Period and deadline constraints relaxed. Effects of failures may be reduced.	More complicated algorithms have to be used. Algorithms may take more execution time.

Table 1 - Choices Extracted from the Arguments

Stage 3(b) then extracts questions from the argument

that can then be used to evaluate whether particular solutions (stage 3(c)) meets the claims from the arguments generated earlier in the process. Table 2 presents some of the results of extracting questions from the arguments for claim **G0011** and its assumption **A0002** from Figure 5. The table includes an evaluation of a solution based on a PID (Proportional Integration Differentiation) loop.

Question	Importance	Response	Design Mod.	Rationale
Goal G0011 - Can the integrity of the operations be justified?	Essential	More design information needed	Dependent on *response* to questions	N/A
Assumption A0002 - Can the dependency between the operation's integrity and the timing properties be relaxed?	Value Added	Only by changing control algorithm used	Results of other trade-off analysis needed	N/A

Table 2 – Evaluation Based on Argument

Table 2 shows how questions for a particular coupling have different importance associated (e.g. *Essential* versus *Value Added*). These relate to properties that must be upheld or those whose handling in a different manner may add benefit (e.g. reduced susceptibility to change). The responses are only partially complete (design modification and rationale not at all) for the solution considered due to the lack of other design information. As the design evolves the level of detail contained in the table would increase and the table would then be populated with evidence from verification activities, e.g. timing analysis.

4 Influence on Component-Based Design

The content of the arguments presented in section 3 can be used to influence the way components in the system are designed and the way in which the architecture is decomposed. This section discusses the influences from some of the goals and in doing so demonstrates the links between the architecture trade-off analysis and component-based design.

From Table 1 it can be seen that some of the choices that need to be made about individual components are affected by choices made by other components within the system. Two cases of influence are given below:

1. *On Component's Functionality* – In Figure 5 goal **G0014** leads to a design option of having a more complicated control algorithm that is more resilient to changes and variations in the system's timing properties. However goal **G0014** is in opposition to goal **G0023** from Figure 4 since it would make the health-monitoring component more complex.
2. *On Abstractions and Interfaces* – Goal **G0021** in Figure 4 leads to a choice over where *health monitoring* functionality is situated. These are; entirely in the *health monitor* component, or partially in the *calculation* component and the rest in the *health monitor* component. The choice alters the abstractions and interfaces between the two components since all relevant data needs to be passed between the components if the *health monitor* component is entirely responsible. In contrast if it is only partially responsible, then a health level would be passed and maybe some data to allow limited validation to be performed in the *health monitor* component. The choice therefore affects the components' design as well as how achievable objectives such as reuse and maintainability are.

Other choices made may not influence the abstractions and interfaces but may affect the components' design. This can be demonstrated through the choice originating from goal **G0011**. Independent of how calculations are performed, the health monitoring is still based on whether the control loop meets the requirements given in section 3. This requires data concerning current sensor inputs and actuator outputs to be passed from the calculation components to the health monitoring. With this data it can be checked whether the inputs and outputs are within limits as well as determining the responsiveness and stability criteria are being met [6]. Hence the abstraction and interface is not affected, but the design of the calculation component and the checks performed are affected.

5 Conclusions

This paper has addressed a method to support architectural design and implementation strategy trade-off analysis, one of the key parts of component-based development. Specifically, the method presented provides guidance when decomposing systems so that the system's objectives are met, deciding what functionality the components should fulfil in-order to achieve the remaining objectives, and showing how this influences the design of components.

Further work could include performing different case studies, to show how argument and design patterns can be used to increase the efficiency of applying the technique, to understand better the relationship between system architecture and component design, and to establish a means by reusing existing work for performing the multiple-criteria optimisation.

6 References

[1] B. Meyer, *Applying Design by Contract*, IEEE Computer, 25(10), pp. 40-51, October 1992.

[2] T. Kelly, *Arguing Safety – A Systematic Approach to Safety Case Management*, DPhil Thesis, YCST-99-05, Department of Computer Science, Univ. of York, 1998.

[3] B. Douglass, *Real-Time UML*, Addison Wesley, 1998.

[4] R. Kazman, M. Klein, P. Clements, *Evaluating Software Architectures – Methods and Case Studies*, Addison Wesley, 2001.

[5] J.-C. Laprie, *Dependable Computing and Fault Tolerance: Concepts and Terminology*, in Proceedings of the 15th International Symposium on Fault Tolerant Computing (FTCS-15), pp. 2-11, 1985.

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