12

Designing with Non-Functional Requirements

Colin J Theaker and Jenny Whitworth

School of Computing, Staffordshire University,
Beaconside, Stafford ST18 0AD, England

Abstract

A major class of software and systems engineering projects are critically dependent upon the attributes of the final system as much upon its functionality. These attributes reflect the quality of the design and implementation of the product, yet in most circumstances, they are treated in a very loose and haphazard fashion during the system development. From very poor requirements specifications, the successful implementation of a system all too often depends upon the intuitive skills of experienced designers. There are many documented examples of systems which have failed as a result.

This paper examines a framework for systems design and implementation that addresses both the problems of correct functionality and also the specification and tracking of non-functional requirements, through to an auditable solution. The objective is to provide an environment in which design engineers (and project managers) can consider design trade-offs and make informed judgements accordingly. The research is ongoing and so the paper outlines some of the longer term aspirations of the project.

Keyword Codes: D.2.1; D.2.8; D.2.6; D.2.10
Key words: Non-functional requirements, system attributes, process modelling, object orientation, CASE and workbench tools, real-time systems.

1. INTRODUCTION

The class of computer-based systems which include reactive, real-time and embedded systems presents us with some of our greatest challenges when it comes to designing and implementing the software. The constraints that are imposed on the final product development may significantly impact upon whether the product itself is a success or failure, with factors such as cost, performance and time-to-market playing a significant role. The nature of reactive systems is also that they are involved in control applications, where issues of safety and reliability may play a dominant role. The 'quality' aspects of system development are therefore often more important than simply providing the expected functionality.

This paper discusses an approach for systems engineering which addresses the wider issues of software development and the quality attributes which are associated with the system. In broad terms we shall refer to these as the **non-functional attributes** of a system, to distinguish them from the logical behaviour (or functionality) that is embodied in the algorithmic implementation of the software. The range of these is very open-ended, but might include performance, reliability, cost, size, etc.

Once a system has been implemented, it is generally feasible to measure the attributes of the system and thereby validate that the requirements have been met. Some may be relatively easy to quantify, for example size and cost may be measured directly. Other attributes may take on a stochastic distribution, whilst others may only be estimates of future behaviour, such as the prediction of future software reliability based on past performance. The situation we would like to achieve is one in which a designer

M. Lee et al. (eds.), *Software Quality and Productivity*

can recognise that these non-functional requirements are present, and that during the design process, they are refined in a manner similar to the function attributes. The refinement process naturally relies upon engineering expertise for both the functional and non-functional partitioning. In many cases the different types of attribute are heavily interrelated. A significant part of the design process, which currently relies largely on intuition or 'gut feeling', is to apply design trade-offs in a quantifiable way so that desired attributes are optimised and that the design is 'safe' [1] with adequate tolerances.

2. CURRENT PRACTICES

Although recognised as an important factor in the design and management of systems [2], in most cases the non-functional characteristics are neither (well) specified as a requirement nor measured, and the achievement of a satisfactory solution to this nebulous problem too often rests in the skill and expertise of experienced practitioners. Even when a well-engineered solution is manifest, with no control or measure of the attributes during system development, the final quality may be unrecognised and subsequently lost for future developments. This is clearly poor engineering practice and equally poor economics.

Within specific design methods, temporal aspects are probably the one area in which non-functional attributes have been most widely addressed. In considering temporal behaviour, some methods have incorporated stochastic techniques. For example, Stochastic Petri nets [5] support the modelling of a system using Petri nets and at the same time use random variables to specify the time behaviour of the system. These nets are obtained by associating an exponentially distributed random variable with each transition of the net. The variable defines the delay from the enabling to the firing of the transition. An alternative approach that has been adopted is to attach an interval of time $[t_{min},t_{max}]$ to each transition. This interval represents the minimum and maximum time that may elapse between the enabling and the firing of a transition. PRM-Nets [4] have been used to develop performance models of parallel programs at the implementation stage. These nets support the modelling of both the software and hardware structures as well as the mapping of software process to physical processors. Other models have adopted a deterministic approach. In CSP [3], the process a-->P models a system which first engages in the event a and then behaves as P. In timed CSP [1], the process $a\text{-}^{t}\text{->}P$ will behave as P precisely t time units after the event a has occurred. These various methods highlight the different approaches to incorporating time within the various development methods.

Apart from these, where any mechanisms are used, they tend to be of a very ad-hoc nature and are ancillary to the main development process rather than being an integral part of it. They mainly take the form of simulation systems, such as [7], which can be used very effectively in determining overall system architectures and configurations, particularly for hardware/software systems in which the hardware structure may be one of the biggest issues. However unless they become part of the main development process, the use of such techniques will only be viable on those systems (and parts of systems) for which the parallel effort in developing a simulation model will be worthwhile.

3. MOOSE

The work described in this paper is part of a much larger collaborative project concerned with the tools and techniques to support systems engineering, entitled MOOSE, or **M**odel-based **O**bject **O**riented **S**ystem **E**ngineering [6]. The focus of the work is on the tools and techniques for the development of

---

[1] 'Safe' in this context applies as much to the development process and the likelihood of achieving a successful product as to any issues of safety criticality.

executable models of systems, through which their functionality can be explored along with the design issues which result by considering the non-functional requirements.

The models we are concerned with here are generated at the very early stages of the life-cycle. The models are executable and this distinguishes the work from other forms of design notation, such as DFDs and ERMs. By using these models from the start, the look and feel of a potential product can be demonstrated, the functionality refined, and an evaluation of its technical feasibility explored.

An object oriented approach has been adopted in which a system is represented as a network of collaborating objects. A system may be refined hierarchically to allow the elaboration of systems of objects into further subsystems. Initially the objects are considered to be *uncommitted* as no assumption is made as to how they will be implemented. The system allows systematic consideration of the design options available for each object, and through the interactions between objects, allows for a dynamic view of the system behaviour. In considering performance requirements (processors, stores etc.) to support an implementation, we could use the system as a vehicle for hardware/software co-design to obtain optimum design trade-offs. In this respect, consideration of the non-functional requirements becomes even more important.

Support for the modelling process is based upon a standard CASE tool, namely Select Yourdon. Diagrams are captured and edited using the standard system, but to overcome limitations in the graphical notation, specific naming conventions have been adopted for the textual annotations to allow the distinctive nature of each object to be identified. The captured database can be interrogated by other software packages to provide additional functionality over and above the standard tool. This mechanism also allows us to enhance the model with a variety of different textual notations to support different aspects of the design process. For example, a language has been defined which allows the model to be 'exercised' to examine the dynamic behaviour of a system. Its functionality can be reviewed and in particular, the interactions between objects closely scrutinised. Similarly the different non-functional requirements of each object can be represented and refined.

The consideration of non-functional requirements starts with the initial specification of the product. Although current practice in generating such specifications is very poor, good guidance on how this may be addressed may be found, for example, in [2]. We have chosen to restrict our activity principally to performance (and other temporal aspects) and reliability. These have interesting differences, not least in the state of the art for their specification and analysis, but they also exhibit many similarities.

In reality, most non-functional attributes would exhibit a stochastic distribution in their behaviour. Such distributions are difficult to apportion during the design refinement and add complexity to the analysis process. Current engineering practice (if done at all) might be to consider single values such as typical or worst cases. The notations we are currently considering therefore lie between these extremes, representing ranges of values and probabilities of falling outside the ranges. Notations of this form are capable of providing very powerful analysis information. For example, with temporal behaviour we could obviously identify logical inconsistencies such as race conditions or incorrect sequencing of events. We could also quantify the performance behaviour, such as typical (mean), best case and worst case responses and also the likelihood of the behaviour falling outside acceptable criteria. This would allow us to identify appropriate levels of hardware performance (in a cost effective way) or to design appropriate recovery mechanisms into the system.

Starting at an initial context diagram with its overall requirements specification, the design refinement progresses. At each refinement, the requirements, both functional and non-functional, are further apportioned to the constituent objects. This is naturally where the skill of the designer is used to derive an appropriate partitioning. The interaction between the objects is important, for example, objects interacting sequentially or in parallel would obviously produce different timing behaviour. With 'rules' in place, it is possible to validate at each stage of the refinement if the partitioning is acceptable in meeting the higher level requirements and to assess the tolerances which ensue.

A verification against the requirements will take place once an actual implementation has been derived (or once we are sufficiently sure of its implementation). In the case of functionality, this may be a simple check list to identify whether the function is present or not. With non-functional requirements, we need to measure the values of the actual attributes of the system. It is worth noting that the overall system may still meet its requirements even though individual sub-systems fall outside the tolerances apportioned to them. However, an awareness of the 'hot spots' is still useful for the management of subsequent developments.

4. FUTURE DEVELOPMENTS

This project is still in a comparatively early stage, particularly for the inclusion of non-functional requirements in the design process. Some tool support has been developed, a number of models have been captured of actual and potential systems, and routes to synthesis are being explored. Issues relating to non-functional requirements are still a subject for debate, particularly with respect to the notational aspects.

The refinement of requirements is one of the more challenging areas, particularly if one considers the support that could be given to the designer at that stage. Whilst certain techniques, such as complexity analysis or function point analysis could help in certain areas, we must still recognise that some of the attributes are unmeasurable even when the system has been implemented, and we are forced to predict future behaviour based on the past. Building up an historical perspective through the collection of metrics is obviously a desirable form of support.

There are issues relating to the refinement which will be addressed as part of this project. For example, as an executable model of the system exists, there is naturally data available relating to the dynamic behaviour. It is proposed to make use of this in connection with the performance analyses. The identification of how objects interact and how the inverse can be utilised in the design process is the subject of further work.

ACKNOWLEDGEMENTS

The work described in this paper has been partially funded by the SERC (project number GR/J09840). The authors would like to acknowledge the help provided by the collaborators on this project, including Professor D Morris at UMIST, Professor R Phillips at the University of Hull, and their respective colleagues. We would also like to thank the companies who have provided support in terms of tools and applications to model, in particular to Select Software Tools Ltd and to ICL.

REFERENCES

1. Davies J., Jackson D.M., Reed G.M., "Timed CSP -Theory and Practice", LNCS Vol. 600, pp640-675.
2. Gilb T., "Principles of Software Engineering Management", Addison-Wesley.
3. Hoare C.A.R., "Communicating Processes", Prentice Hall.
4. Hull M.E.C., O'Donoghue P.G., "Timed Petri net Approach to Performance Modelling with the MOON Method", *Software Engineering Journal*, Vol 9, No 3, pp 95-106.
5. Marsan A., Balbo G., Conte G., "Performnce Models of Multi Processor Systems", MIT Press.
6. Morris D., Evans D.G., Schofield S., "Model-based Object Oriented System Engineering (MOOSE) - A Design Method and Notation", To be published.
7. SES Workbench, Software Engineering Software Inc., Austin Texax.