Descending the Twin Peaks: Requirements and Architecture in the EGSO Project

Nathan Ching (nc@mssl.ucl.ac.uk) and Clare Gryce (c.gryce@cs.ucl.ac.uk)
University College London

Abstract

The requirements and architecture of any software system are highly interdependent. The architecture can address the requirements of the system, and also feedback to them, constraining the system under development. For any complex software system, understanding requirements and using them to make informed architectural decisions is crucial to success. The e-Science domain presents many challenges to the systems developer. Requirements are subject to change, and the development environment is populated with new technologies and paradigms. Projects need practical solutions and methods to address these concerns that are appropriate to their resources and organisation. We discuss some lightweight techniques that have been used to manage requirements in the EGSO project, and outline our approach to the definition of a system architecture. We also highlight the ways in which these have been successful in addressing development issues. The Twin Peaks model of software development focuses explicitly on requirements and architecture, allowing these two key concerns to evolve concurrently and independently. Finally, we discuss how the Twin Peaks model addresses some of the challenges characteristic of e-Science projects, and present a style of prototyping that is suited to bridging the gap between requirements and architecture in the context of the model.

1. Introduction

In this paper, we consider the evolution of *Requirements* and system *Architectures* in the e-Science domain. Within any system, these two key concerns are highly interdependent. The architecture is the first artefact in the development process that addresses the requirements of the system. Conversely, architectural choices can feedback to the requirements, constraining the system under development. In designing and building software systems of any complexity, understanding requirements and using them to make informed architectural decisions is crucial to project success.

The e-Science domain presents many challenges to the systems developer. Within the objective of enabling new and innovative ways of doing science, is an implicit acknowledgement that requirements are likely to evolve over time. The technical environment of e-Science projects is populated with new technologies, tools and paradigms, each with their own implicit architectural constraints. Additionally, the organisational characteristics of e-Science projects are often complex. Project teams are usually distributed and constraints on resources mean that individual team members frequently play multiple stakeholder roles. These issues suggest a need for lightweight, accessible techniques for system definition and development. E-Science projects need methods that are readily utilised by individual team members, regardless of their background and experience.

In these respects, EGSO (*European Grid of Solar Observations)* [1] typifies an e-Science project. EGSO project partners are geographically widespread and have varying backgrounds and experience. The practicalities of communication and co-ordination have demanded simple, effective solutions for requirements analysis and management, and architectural definition. In this paper, we highlight some techniques and approaches we have applied to address these concerns. We also note that the 'Twin Peaks' model of software development serves as a useful reference for e-Science projects. The model describes the relationship between requirements and architecture, and suggests how development can proceed in a complex, dynamic environment.

In section 2, we discuss techniques used for requirements analysis and management, and in section 3 our approach to architectural definition is outlined. Section 4 introduces the 'Twin Peaks' model, and in section 5 we briefly describe one project initiative used to align the requirements and architecture.

2. Requirements Analysis and Management

The definition of workable system requirements for EGSO began at an early stage in order to add detail to the ideas that had lead to the inception of the project. This culminated

in the production of several documents, including a set of scientific use cases derived from community consultation, and a 'System Concepts' document that reflected the result of discussions within the project team. Whilst these artefacts proved useful, they were not sufficiently refined to form a basis for system development. The use cases were highly specific and technical, and it proved difficult to draw useful generalizations from them, whilst the System Concepts document took a mechanistic approach to the problem of describing required functionality, resulting in a set of requirements that were heavily interwoven with design assumptions.

To overcome these problems the various documents were used to produce a focussed statement of system functionality. This took the form of an indexed System Requirements Document (SRD) that consolidated the existing information into a discrete series of functional and non-functional requirements. This provided scope for the isolated management of individual requirements and the association of meta-information, such as a MoSCoW[1] priority, and comments that helped to retain their context. Whilst this was judged to be successful in separating the concerns of '*what*' the system should do from '*how*' it should do it, it also highlighted some hidden weaknesses in the way requirements had been described; for example labels and definitions had been used inconsistently. Also, it became apparent that the requirements needed to be more formally traced to the real world problems that had inspired them.

In order to provide the desired traceability, a reverse engineering process was undertaken. This involved a self-referential analysis of the SRD in which we related requirements to each other on the basis that specific low-level requirements might have a collective responsibility for delivering higher-level generic requirements. Practically, this involved organizing the requirements in a simplified tree structure, where children are related to their parents by either satisfying or constraining them [2]. Due to the qualitative nature of many of the requirements, it was also necessary to consider the partial satisfaction of one requirement by another. The results of this analysis provided a second formal view of the requirements, which allowed useful management possibilities such as the ability to apply sanity checks to priorities and to hide detail by collapsing tree branches. More significantly it again revealed areas of weakness; for example implicit but un-stated high-level requirements and seemingly arbitrary constraints were identified and explored. Thus, the process of reconstruction provided a more precise understanding of the requirements.

To avoid the problem of double maintenance the derived relationship information was folded back into the original SRD. This was achieved by adding pointers to 'child' requirements as an item of meta-information. At this point the SRD was also marked-up using XML, allowing associated technologies to be leveraged in order to constrain the structure of the document between editing sessions, and transform it to provide different views. For the purpose of constraining the document structure a schema was defined which included constructs to aid version control. XSLT style-sheets were then written to effect the transformation of the document: to HTML with hyperlinks to navigate the relationships (Figure 1) and to a graph specification that could be interpreted by the DOT program [3] in order to generate a more intuitive view of the tree (Figure 2).

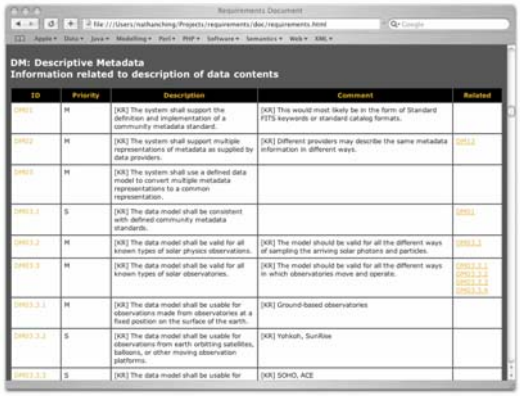

Figure 1: HTML View of the SRD

Utilising 'best practice' techniques and developing effective tools and measures has enabled EGSO to deal with the problem of requirement instability. In particular the application of the analysis technique outlined lead to the identification of a core set of high-level requirements for the system, which have been safely 'frozen' to minimize costs associated with changes to them. Other peripheral requirements have not been frozen,

[1] This refers to a simple four level prioritization scheme in which 'M' denotes a requirement that *must* be met, 'S' a requirement that *should* be met, 'C' a requirement that *could* be met and 'W' a requirement that we *would like* to meet.

but are open and amenable to fine grained development.

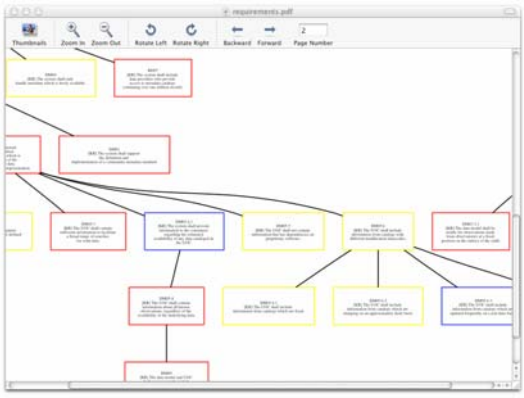

Figure 2: Tree View of the SRD

3. Architectural Definition

In designing large software systems, many factors other than formal requirements can direct, or otherwise impact upon an emerging architecture. Within EGSO, the previous experience of project partners, the 'vision' of the system as described by early project documents, and technology surveys conducted within the project all suggested possible design solutions. As previously mentioned, early design ideas such as these can tend to be mechanistic in nature, suggesting details that effectively constrain the problem solving space.

An approach is required that minimises predisposition to particular implementation, and focuses explicitly on the functionality required of the system. Accordingly, the high-level architecture for EGSO has been developed in line with the principles of the Model Driven Architecture (MDA) [4], promoting clarification of '*what*' the system is required to do, rather than '*how*' it is to do it. The essential business logic of the system is captured in a UML (Unified Modelling Language) [5] model that is independent of any possible implementation technology or platform.

In this way, the potential for re-use of the EGSO architecture has been significantly enhanced, representing a considerable advantage for a system evolving within the dynamic e-Science domain. The high-level architecture also serves as an essential shared view of the system under development. This shared view serves both as an artefact for effective communication between team members, and as a baseline for further architectural refinement. Being modular in nature, the architecture also suggests a convenient breakdown of further development work that is appropriate to a distributed team.

4. The Twin Peaks Model of Software Development

The 'Twin Peaks' model of software development [6] is a partial view of the well-known Spiral development model. It focuses explicitly on requirements and architecture, allowing for their concurrent and independent evolution. (Figure 3) Development proceeds with successive iterations of both these concerns, leading to increasingly detailed requirements and refinement of the system architecture.

The Twin Peaks model addresses many development issues that are characteristic of e-Science projects: the need for rapid prototyping that can facilitate exploration and elaboration of requirements, the need to be able to evaluate candidate architectural components, and the need to be able to accommodate requirements that are subject to change. Further, the model recognises the reality of most development projects, where time and resource constraints do not allow for a lengthy requirements elicitation process before proceeding with system design. We have described how, within EGSO, the development of the requirements and architecture formed two semi-independent tracks of activity. The project is progressing according to the guidelines offered by the 'Twin Peaks' model.

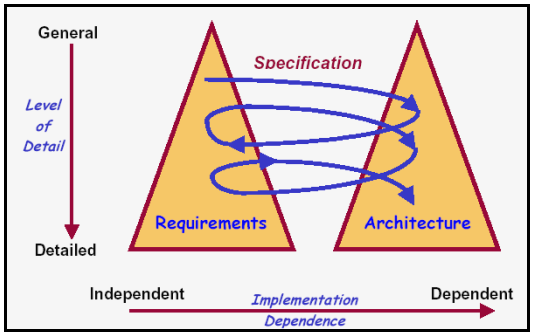

Figure 3: The Twin Peaks Model [6]

5. Bridging the Gap between Requirements and Architecture

The challenge presented by the Twin Peaks model is that of bridging the gap between requirements and architecture in systems

development. This is particularly so during the early stages of development when both specifications lack the necessary detail to make them fully coherent. To bridge this gap, the EGSO project developed several simple 'Scenario Animators' to provide a user view of the architecture that could both be measured against, and used to inform the system requirements.

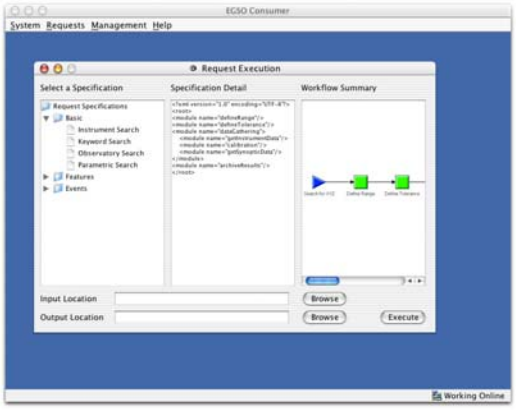

Figure 4: Consumer Role Scenario Animator

Three applications were developed that envisioned user perceived functionality associated with the EGSO system architecture. These corresponded to the three system roles defined by the architecture, that of the Consumer (Figure 4), Provider and Broker, and reflected the underlying components associated with each. Whilst these applications did not seek to 'spike' implementation risks (they lacked any real underlying logic) they facilitated a high level exploration of the solution space, helping to identify emergent requirements whilst informing architectural development.

The project has also undertaken research that aims to match recognised 'architectural styles' to a set of derived general requirements for the data-grid domain. (To be published).

6. Conclusions

In conclusion, we have identified some development challenges faced within the EGSO project, which we believe are characteristic of the e-Science domain. These include a shifting technological landscape, changing application requirements, and complex organisational characteristics where traditional stakeholder boundaries are blurred. We assert that long-term project success depends on the selection of appropriate development methods that can promote effective communication, and support such a dynamic development environment.

We have emphasised the importance of separating the key concerns of requirements and architecture in order to minimise the constraints they place upon each other during the early stages of a project. Further, we have demonstrated how both can be developed concurrently, in a way that addresses the challenges identified within the e-Science domain. In the case of requirements, we have noted some lightweight techniques that can ensure they are manageable, focussed and adaptive to change. In the case of architecture, we propose that the adoption of MDA principles is an appropriate strategy for the definition of system architecture in this context; further addressing the challenge of adaptability, in addition to aiding task decomposition within a distributed development environment.

Finally, we believe that we have identified a useful development model for e-Science projects, the 'Twin Peaks' model, that offers a framework in which development of requirements and architecture can proceed semi-independently. We have also noted one of the key challenges this model poses, that of ensuring that requirements and architecture ultimately cohere, and have briefly described a practical solution to address this.

References

[1] The EGSO Project website, http://www.egso.org/

[2] A. van Lamsweerde, "*Requirements engineering in the year 00: A research perspective*". In Proc. 22nd International Conference on Software Engineering. June 2000.

[3] E. Gansner et. al., "*Drawing graphs with dot*". http://www.research.att.com/sw/tools/graphviz/dotguide.pdf. 2002.

[4] Object Management Group. "*The Model Driven Architecture*". http://www.omg.org/mda/

[5] Object Management Group. "*Unified Modelling Language Specification v1.4*". http://cgi.omg.org/docs/formal/01-09-67.pdf. September 2001.

[6] B. Nuseibeh. "*Weaving the Software Development Process Between Requirements and Architecture*". In Proc. of ICSE2001 Workshop STRAW-01, Toronto, May 2001.