Views in Support of Architectural Evolution

Alessandro Maccari
Software Architectures Group
Nokia Research Center
P. O. Box 407, FIN — 00045 NOKIA GROUP
Tel.: +358 40 749 9074
Fax: +358 9 4376 6308
alessandro.maccari@nokia.com

Galal H. Galal
School of Informatics and Multimedia Technology, University of North London
166-220 Holloway Road — London N7 8DB
Tel.: +44 20 7753 7001
Fax: +44 20 7753 7009
galal@acm.org

Abstract

Managing evolution of complex software architectures is a continuous challenge in industry. Systems such as cell phones undergo a continuous increase in complexity, while the fast market evolution impose quick integration of new features. Being able to easily manage software architecture evolution is the basis for quicker time-to-market and faster product evolution. The notion of architectural viewpoint has been used in Nokia for a long time, largely based on the classical 4+1 view model. Emphasis on architectural evolution generates the need to add specific viewpoints to that basic model. We list the most important ones and provide basis for discussion and further research on the matter.

1. Introduction

Developing software for systems such as mobile phones means being involved with the maintenance and evolution of complex architectures (which takes time) and, at the same time, having to integrate new features in existing systems at market (i.e. fast) speed. Proper architectural modelling plays an essential role in such tasks: having good architectural documentation is the basis for successfully managing the evolution of the software architecture and roadmapping of system features.

Architectural modelling based on different views has been used in Nokia for a long time. The basics of the discipline was set back in 1995 by Philippe Kruchten s paper on the 4+1 view model [Kruchten]. We believe that the model is still valid, and will refer to it throughout this short paper.

We have elaborated that model to make it fit the specific needs posed by real-time, embedded telecom systems. In a workshop at the last ICSE conference [ICSEworkshop] we have presented our flavour of the 4+1 view model. In brief, we model architectures by means of four basic viewpoints, resulting in the following views:

a) requirements view (which includes a domain model and a requirements model, mainly in the form of use cases);
b) conceptual view (made of architecturally significant entities, stereotypes, constraints and interaction patterns);
c) logical view (major logical components and relevant provide service/ require service relationships between them);
d) implementation view (source code or target modules that implement the logical view elements).

Our model does not include a deployment view (yet), since it has proved to be rather trivial in the case of our mobile handset software architecture.

The main additions we make to Kruchten s model are:

a) the requirements view contains not only use cases (engineered functional requirements) but also non-functional requirements and, most importantly, a domain model [Jackson], which we find of invaluable help in modelling the reality and the problem domain;
b) the conceptual view, containing architecturally significant constraints and patterns, is not explicitly present in Kruchten s model (we believe it s meant to be implicit in the logical view, although the author is the best person to ask!);

We briefly elaborate on the requirements and conceptual views in sections 2 and 3 respectively.

We believe that the above-listed views provide a fairly good static descriptions of software architecture. However, when it comes to managing evolution, it can be very hard to extrapolate the necessary information from the views we have used so far. A workshop recently held with the ECOOP conference [ECOOPworkshop] focused on architectural evolution; its participants debated on how to model architecture so to be able to manage and even predict evolution. Partially inspired from the conclusions thereby reached, we present the idea of an architectonics view (section 4), which classifies software components according to different layers of change.

Moreover, we present other views that we need to use to model the architecture of our systems (section 5). Telecom systems, and mobile handsets in particular, have complex runtime behaviour due to events from the environment and the mobility of the devices. Understanding such behaviour (at both user and operating system level) is crucial for writing effective test cases, and this is the reason why we propose the dynamic view and task view.

Finally, we introduce the notion of the organizational view, which models the structure of the developing organization. While this view is trivial in most cases, when working in large, multi-site organizations such as Nokia it becomes crucial. (The theory that asserts that the structure of a software system mirrors that of the developing organization can be recognized as true by anyone who has worked in such a company).

We conclude the paper with a discussion about the IEEE standard 1471, and propose topics for further research.

2. Requirements view

The requirements view contains requirements (functional, expressed in terms of use cases, and qualitative) and a domain model. This contrasts with the +1 part of Kruchten s model, which contains only use cases. Following Michael Jackson s reasoning [Jackson], in order to find a good solution we must first understand the problem. In our case, part of the problem has to do with modelling the reality.

An example from the mobile phone domain is a message. Digital mobile phones have always been capable of sending and receiving messages. Until very recently, messages were simply made of (almost) ASCII text, up to 160 characters long. In the past two years, the usage of messages has boomed, and Nokia has launched products that support picture messages, where a picture can be attached to the message. In the future, with the availability of the so-called third-generation networks, higher bandwidth will allow the transmission of multimedia messages, which we guess will be very similar to today s emails and include attachments of various kinds (documents, pictures, sounds), that can be viewed or played by certain applications.

Clearly, the concept of message is no more as simple as it used to be. Understanding what a message is (and how it can evolve in the future, see section 4) is crucial for architecting the messaging software in mobile handsets. The answer to the question what is a message? (and to similar, even more complicated questions, such as what is a call?) lies in the domain model.

We feel that a well-defined domain model is an essential part of the architecture, since it allows the understanding of the system s architectonic nature and its implications on the evolution of the software system (see section 4).

It is to be noted that even with the addition of a domain model and of qualitative requirements (the *bilities), our requirements view does not contain any description of the system, and thus still qualifies to be a +1 view in the Kruchten sense.

3. Conceptual architecture view

We created the conceptual view after realising that the logical view in the Kruchten model did not explicitly contain a number of things:

a) the constraints on the types of components that can exist and on the allowed relationships between them (usually part of the architectural style);
b) the architectural patterns (or design patterns that have been applied at the architectural level);
c) the main system-level rules that software developers and architects must follow when building new parts of the system.

We felt that such information should be part of a view of its own. That view contains what everyone who works with a particular software architecture needs to know, and should be included in the basic training for developers and, most importantly, chief designers and architects.

An example from the mobile phone domain is the Symbian [Symbian] user interface architecture. Symbian is an operating system platform which is targeted to high-end mobile handsets with a rich functionality and am advanced graphical user interface (currently used in Nokia s 9210 Communicator and in Ericsson s R380). The Symbian platform is developed independently of the target products.

Applications are built using a flavour of the module-view-controller design patterns, which allows an application engine to run independently of its UI appearance. This way, an application can be adapted to different user interface styles by rewriting only the view code, without touching the engine code. This may sound like a requirements for applications, but we believe it has architectural implications, since all applications that run on Symbian platforms must conform to this rule.

4. The Architectonic view

In a book about how buildings learn and adapt over time, Brand [Brand] refers to the layers of change that comprise buildings. Brand identifies six layers in a building that change at different rates. From the slowest to the fastest these are: Site, Structure, Skin, Services, Space Plan, and Stuff (meaning things like furniture, decorations, light fixtures and appliances). Figure 1 below illustrates this view.

Figure 1: Shearing Layers of Change (Courtesy of Phoenix Illustrated)

This view is fundamentally *normative*, i.e. it is based on a study of the types of changes that typically affect buildings, after construction and delivery to clients, as a result of adaptations by their users. Buildings that accommodate such changes gracefully, are the ones that please their users most and remain useful for longer. Brand cites a few buildings that have been adapted by their users over many decades, even centuries, at manageable costs, and remain in use today. They are loved and cherished by their past and present users alike. Such buildings are capable of such accommodation of unforeseen uses because the layers that make them up are loosely-coupled. These layers *slip past* each other: changes to one layer do not necessitate changes to others. Note here that the low coupling is not at the level of individual bricks and other individual constructional elements: rather, the de-coupling referred to is at the level of categories, or layers, of such elements. The constructional elements are categorised according to the degree of susceptibility to, or speed of, change that they share. The categorisation also relates to the degree in which each layer constrains others, and to the scale of disruption that the change of each entails.

Our reading of architectural literature has identified another view congenial with Brand s; that is of the architectural theorist and critic Kenneth Frampton [Frampton]. Frampton uses the term Architectonic attributes to refer to the *light* versus *heavy* characteristics of constructional elements. He uses vernacular examples, such as the Greek temple, to show how the architectonic attributes of various construction technologies have been used to retain traditional symbolic values that are germane to certain *cultures*. Frampton goes on to show how various *cultural* expressions manifest continuation of the architectonic attributes of building material throughout their various histories. By comparison, Frampton observes how the architectonic role of constructional material can reversed from one culture to another: the relative lightness or heaviness of constructional elements is a function of the culture, its particular condition, and the expressions that it tries to achieve. So again, we encounter another view of architecture that demarcates categories of building blocks according to their relative stability characteristic, this time with the role of *cultural specificity* and variances spelled out.

Our conclusion at this point is that the way in which architecture constraints an artefact of any sort is very much dependent on the culture that spawns it. What is *heavy* is more constraining than what is *light*. The choice lies with the culture tradition that uses or indeed develops the building, or in our case, software. There is therefore a need to direct investigative attention to these cultural choices, and to the way in which they allow or constraint the evolution of the software artefact. For example, the Conceptual architecture view that we reported that Nokia has developed above, is also a cultural and business choice, which leads to certain allowances to and certain constraints on how the software can be feasibly evolved. In the Nokia example therefore, the conceptual architecture is more akin to the site or structure layer in Figure 1 above. This is resonant with the view that the aforementioned ECOOP workshop converged upon: namely that software architecture is primarily an expression of constraints!

We would like to submit here that there is certain validity to the view that software architecture should be less concerned with structural elements and more with categories of software system s components, in the large-grain. Stratifying such categories according to their relative rigidity, scale and speed, of change can help our architecting effort by making the architectonic nature of the software object as whole clearer. This clarity means that the impact of various architectural decisions can be studied more carefully, and in conjunction with the relevant stakeholders. The aim is also to support the understanding and consequent design of systems, so that adaptability properties are maximised, but with respect to the particular situation (read culture) that we are referred to. We refer to this view of systems as the architectonic view. We hasten to add here that there is not much that is fundamentally new here in this type of differentiation: the SPARC database model, modern operating systems, and the ISO OSI reference model follow the same principle. What is new is our reference to the way in which the specific problem domain can affect, and indeed spawn, such categorisation.