Feature-Guided Architecture Development for Embedded System Families

T. J. Brown, R. Bashroush, C. Gillan, I. Spence, P. Kilpatrick
School Of Computer Science, Queen's University of Belfast.
{tj.brown, r.bashroush, c.gillan, i.spence, p.kilpatrick}@qub.ac.uk

Abstract

Software product-line engineering aims to maximize reuse by exploiting the commonality within families of related systems. Its success depend on capturing the commonality and variability, and using this to evolve a reference architecture for the product family. With embedded system families, the possibility of variability in hardware and operating system platforms is an added complication. In this paper we outline a strategy for evolving reference architectures from bi-directional feature models. The proposed strategy complements information provided by the feature model with scenarios that help to elaborate feature behavior.

1. Introduction

Software Product-line Engineering [1] has emerged as a major strategy for maximizing reuse when a family of related software systems is to be fielded. In this approach, an initial phase of commonality-variability analysis, often using feature modeling [2][3][4], is followed by the design of a reference architecture for the family. In this work we address the issue of deriving reference architectures from feature models. In the case of feature models for embedded system families, it can be useful to model both hardware and software features and capture the relationships between the two. Our scheme of feature modeling does this. It is inspired by earlier approaches to feature modeling, including FODA[2] and FORM[3][4], and incorporates concepts from other work [5][7]. Construction of such models is only justified if they can be used as a significant aid to the architecture development stage. We have evolved an outline methodology for architecture design that is guided by and structured from the product family feature model, but makes use of additional information derived from scenarios designed to exercise features.

2. Rationalised Feature Modelling

Our feature modeling schema partitions the feature model into capability, operating environment and domain technology feature layers, as does FORM [3][4]. Also included is an optional inverted feature tree that captures features provided by the hardware and operating systems. When this platform layer is present there can be relationships across the hardware/software boundary, including feature dependencies and alternatives. The latter indicates product functionality which may be provided as software on some products, but as hardware on others.

2.1 Capturing Feature Behavior

Our architecture design methodology entails the iterative use of scenarios to elicit details of related behavior. To capture this information in a completely architecture independent way, we employ the Use Case Maps (UCM) path notation [7][8]. Our feature modeling tool allows UCM paths to be attached to features where this is appropriate.

3. Feature Model Evolution

In the first instance a feature model will be created using requirements information assembled from a variety of sources. To evolve towards an architecture, we begin within the capability feature layer and consider each capability feature as a candidate sub-system. To help to decide on the viability of this emerging partitioning we consider the degree of cross-coupling likely to arise between sub-systems. It may be necessary to develop some initial test scenarios or use cases to do this. We need to find a partitioning in which interaction traffic across the tree is generally modest compared with interaction up and down the tree. We select a level in the feature tree at which there is an acceptably low level of cross-coupling. Sub-systems are then centered on these capability features.

For each feature selected as the basis of a sub-system we now consider whether that sub-system should form a passive software component that communicates synchronously and executes sequentially, or an active task that may execute concurrently using asynchronous communication. It may be necessary to group multiple sub-systems within the same task to ensure that communication can be asynchronous. Other factors like required response times, scheduling policy, and issues relating to resource sharing may need to be taken into consideration. Finding an optimal model for a family of products may not be easy, but the feature model remains an important source of guidance.

At this point the model's top-level structure reflects what are really early architectural decisions. Before proceeding through the feature tree, we turn our attention to the platform features and their relationships with features that we expect to provide as software.

4. Platform Independence

The main reason for modeling the features provided by the platform is to help with the evolution of a software design that is significantly decoupled from the hardware and O/S on which it operates. The key stratagem for achieving this decoupling is the design of an *Adaptable Platform Abstraction Layer.* The complexity of this layer may vary considerably from one situation to another, but it must be able to present a fixed interface to software layers above, irrespective of the specific hardware devices below. Multiple implementations of some interface elements may be needed and a modular structure for this layer will frequently be appropriate. Having a model of the features provided by the platform, and their relationships to those to be provided in software, is a key enabler at this stage.

5. Elaborating Behavioral Detail

With a sub-system and task model and a platform abstraction layer defined, the design is 'bounded' at the top and bottom levels. The next task is to elaborate detail of intermediate layers, by working down developing scenarios that help to expose feature behavior and variability. Our approach is influenced by the PuLSE-DSSA strategy [9]. Features are prioritized and scenarios that exercise them are applied in order. The understanding gained is captured using UCM path representations linked to features. We exploit the UCM concepts of stubs and plug-ins [7][8] to capture behavioral variability. Thus a feature at one level with alternative child features can have a dynamic stub within its path. Its child alternatives can then contribute alternative plug-in paths.

Application of a scenario should typically elaborate detail of issues such as task interactions, the identification of components and sub-components and the services they must provide. Where a high level feature has optional or alternative sub-features, we may need multiple scenarios. Using multiple scenarios aids understanding of the way communication patterns change with feature combinations.

6. Conclusions

To support the design of reference architectures for embedded system families, we have evolved bi-directional feature modeling with behavior capture, and a related architecture development methodology. Experience with these techniques is limited, but we have explored their use in the design of optical network products. [10]. More remains to be done to refine both the notation and the process techniques.

7. References

[1] L. M. Northrop, "A Framework for Software Product Line Practice – version 3", *Software Engineering Institute*, 2001.

[2] Kyo C. Kang, G. C. Shalom, J. A. Hess, W. E. Novak and A. S. Pettersen, "Feature-Oriented Domain Analysis (FODA) Feasibility Study", *Technical Report CMU/SEI 90-TR-21*, 1990.

[3] K. Lee, Kyo C. Kang, W. Chae and B.B. Choi, "Feature-based approach to object-oriented engineering of applications for reuse", *Software Practice and Experience*, Vol. 30, pp. 1025 – 1046, 2000.

[4] Kyo C. Kang, S. Kim, J. Lee and K. Lee, "Feature-Oriented Engineering of PBX Software for Adaptability and Reusability", *Software Practice and Experience*, vol. 29, pp. 875 – 896, 1999.

[5] K. Czarnecki and U. W. Eisenecker, "*Generative Programming: Methods Tools and Applications, - Chapter 4*", Addison-Wesley, 2000.

[6] D. Fey, R. Fajta and A. Boros, "Feature Modeling: A Meta-model to Enhance Usability and Usefulness", *Proceedings of the 2nd International Conference on Software Product Lines (SPLC2),* Springer, LNCS 2379, 2002. pp. 198 – 216.

[7] D. Amyot, "Use Case Maps as a Feature Description Notation", in *Language Constructs for Describing Features: Proceedings of the FIREworks workshop*, Eds. Gilmore and Ryan, Springer-Verlag, 2001.

[8] Web reference: http://www.usecasemaps.org/index.shtml

[9] M. Anastasopoulos, J. Bayer, O. Flege and C. Gacek, "*A Process for Product Line Architecture Creation and Evaluation: PuLSE-DSSA – version 2*", Fraunhofer Institute for Experimental Software Engineering IESE-Report 038.00/E, 2000.

[10] C. Gillan, I. Spence, T. J. Brown, P. Kilpatrick, R. Bashroush, R Smith, "Applying Product Line Engineering Techniques to SDH Multiplexing, submitted to The International Journal of Embedded Systems, September 2005.