

FEATURE MANAGEMENT APPLIED TO ON BOARD SOFTWARE BUILDING BLOCKS

Arnaud Bourdoux⁽¹⁾, Laurent Demonceau⁽¹⁾, Paul Parisis⁽¹⁾
Andreas Classen⁽²⁾, Quentin Boucher⁽²⁾

⁽¹⁾ *SPACEBEL s.a., Liège Science Park, B4031 Angleur, Belgium, +32 4 361 81 11*
arnaud.bourdoux@spacebel.be, laurent.demonceau@spacebel.be, paul.paris@spacebel.be

⁽²⁾ *Facultés Universitaires Notre Dame de la Paix, Namur, Belgium,*
andreas.classen@info.fundp.ac.be

1. KEY WORDS

Software Product Line Engineering, Feature Management, On Board Software, Component, Reference Architecture, Building Blocks, Reuse, CFDP.

2. ABSTRACT

This paper devises the Software Product Line Engineering and more particularly the Feature Management as an efficient way of managing the variability of generic on board software building blocks. The principles of Feature Management are sketched along with its impact on the software development lifecycle. A use case of Feature Management applied to CFDP developed by Spacebel [14] illustrates the method. Other possible applications in the context of the reuse of on board software building blocks in reference architecture are discussed.

3. CONTEXT

Reference architecture, building blocks and reuse are becoming increasingly important subjects in on board software development. In order to reduce engineering and development costs, reusable components are becoming more and more sought after.

However, software building blocks cannot always be taken off the shelf and reused as is. They must often be modified to match specific mission needs, unless reuse is taken into account at building block design time through sufficient genericity to embrace potentially variable needs.

The reuse of generic building blocks can then lead to significant overweight in terms of memory footprint and processor load. It can also result in dead code. These issues are typical of

flight software, where memory and computing resources are limited and reliability is critical.

Software genericity and flight code constraints therefore tend to go against each other. Reused component could reveal less efficient than those that would have specifically been developed for the mission.

An industrial process for managing and producing generic components while mastering the resulting overweight is thus necessary. It must allow the selection of the required functionality while the unnecessary functionality is wiped out.

4. FEATURE MANAGEMENT OVERVIEW

4.1 Software Product Line Modelling

Software Product Line Engineering (SPLE) is an emergent software engineering paradigm institutionalising reuse throughout the software lifecycle. A software product line (SPL) can be defined as "a set of software-intensive systems that share a common, managed set of features satisfying the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way" [5]. By adopting SPLE, one expects to benefit from economies of scale and thereby to improve the cost, productivity, time to market, and quality of developing software.

"Central to the SPLE paradigm is the modelling and management of variability, i.e. the commonalities and differences in the applications in terms of requirements, architecture, components, and test artefacts" [10]. This variability is commonly expressed using features, which appear to be first class abstractions that shape the reasoning of the

engineers and other stakeholders [4]. A set of features is a product of the SPL.

Features can be grouped in feature diagrams (FDs), which model the variability of the SPL at a high level of granularity: an FD expresses the set of products of the SPL. They are generally used (i) to capture commonality and variability, (ii) to represent dependencies between features, (iii) to determine combinations of features that are allowed and disallowed in the SPL, and (iv) to guide the configuration process.

Basically, FDs are trees¹ whose nodes denote features and whose edges represent top-down hierarchical decomposition of features. Each decomposition indicates that, given the presence of the parent feature in a product, some combination of its children should also be present in the product. In the same idea, for a feature to be selectable for a product, its parent must also be selected. In addition to their tree-shaped backbone, Feature Diagrams can also contain additional constraints, usually specified in propositional logic.

Among the standard analysis tasks for FDs are (i) satisfiability checking, i.e. to check whether the FD is not overconstrained and admits no product, (ii) product checking, i.e. to check whether a particular product is part of the product line, (iii) dead feature search, i.e. to uncover features that never appear in a product or simply (iv) to calculate the number of products in the product line.

A number of FD notations have been proposed in the literature (see for example [2], [6], [7], [8], [9] or [13]) and at the moment, there is no unified and universally accepted one. However, most of these notations are very similar and the preceding description applies to all of them. FDs have a formal semantics, which can be easily implemented in propositional logic. Many analysis tasks can thus be automated using off-the-shelf satisfiability solvers.

4.1 Feature Management Implementation

A binding mechanism has been developed with the objective of managing such a large number of features while keeping the code easily readable and maintainable. It relies on a simple way of tagging the source code. A tag consists in a formatted comment that includes a feature

name. Its scope extends over the next functional block so that there is no need to tag the end of the block. A block can be a simple declaration or statement, a more complex construct or even a complete function or file.

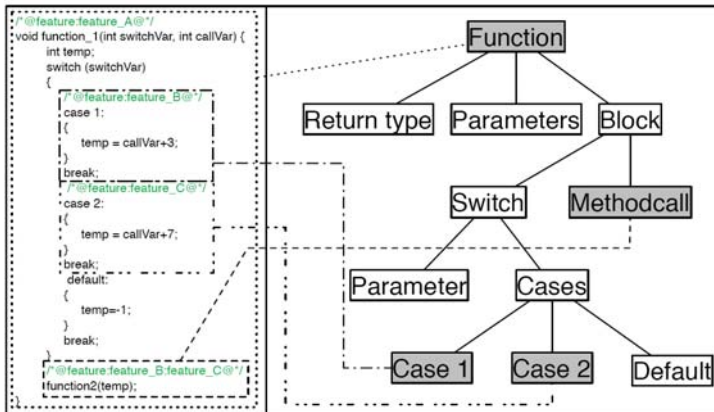
Complete files or functions can be attached to features during software engineering, using the usual UML tool. Tags are then automatically introduced in the source code generated by the tool. When only particular sections of code are being used by given features, tags may however need to be manually added at design and coding time.

To create a final product, also called Featured Version, the complete library is first passed, together with the selected feature set, through a parser developed in Flex and Bison that removes the code corresponding to not selected features. The resulting code is then compiled normally using the standard C compiler. As the feature tags are also present in the UML models, the generated documentation also only contains information relevant to the features that are actually part of the build.

An important characteristic of the code tagging approach is that functional blocks correspond to structural elements of the C source code i.e. elements of the Abstract Syntax Tree (AST) of the language. This implies that the pruned code will always be syntactically correct as it is only possible to remove groups of statements that belong together. This is illustrated in Figure 1 which contains an example of tagged code on the left and its associated AST on the right. There, the highlighted nodes of the AST correspond to tagged portions of code. `/*@feature:feature A@*/` is an example of tag covering a whole function, in this case function 1. If feature_A is not part of a product, the whole function will be removed, i.e. the source code associated to the Function node of the AST (as well as all its sub-nodes) will be pruned. `/*@feature:feature B@*/` and `/*@feature:feature C@*/` each cover a case block.

This means that if, for example, feature_B is not part of a product, the case block where `switchVar` equals 1 will be removed of the source file. This deletion keeps the code syntactically correct as it removes the whole case block (highlighted in Figure 1). Finally, the `/*@feature:feature B:feature C@*/` tag covers a function call that will be removed only if none of feature B and feature C are selected in a product.

¹ Sometimes, directed acyclic graphs (a node can have several parents) are used, too.



Once again, the deletion of this statement keeps the code syntactically correct as it corresponds to a node of the AST.

Figure 1 presents only a few examples of functional blocks but the same holds for all other functional blocks such as loops, groups of statements enclosed by braces, ...

5. FEATURE MANAGEMENT IN THE SOFTWARE PROCESS

Feature Management is not limited to source code. It concerns the complete software development and also the associated documentation and tests. It has to be considered during the initial product development, and each time the product is going to be reused in a particular project.

5.1 Development Time

Feature management must be taken into account very early in the software development life-cycle. The feature diagram elaboration must be part of the engineering effort. It actually benefits to the engineering as it obliges to identify all the features, the relation between them and their variability before the software implementation. At the end of the development, every feature must be carefully characterized in terms of its resources consumption.

Creating the features and elaborating the feature diagram must be seen as a crucial activity. One aspect of the problem that needs to be carefully thought about is the granularity of the features, i.e. the depth of the feature tree. Too fine granularity indeed raises the development costs, as each new feature introduces an overhead in the engineering process, while too coarse granularity misses the goal of using feature management, as additional development could then still be needed at reuse time.

5.2 Reuse Time

At reuse time, only the requirements related to the selected Features are applicable to the product. The Feature selection is therefore a major activity driving the software requirements. The Feature Set to be proposed for a particular project is the minimum set for which the associated software requirements fully cover the user requirements. As the memory footprint and processor load impact of each feature is known, the resource budget corresponding to the Feature Set can be directly obtained.

From there, the requirements relative to the selected Feature Set are flowed down to the architecture, to the code and to the tests. Following this approach, documentation produced using a feature set does not contain irrelevant information, and only the applicable tests are executed on a featured version build.

6. CFDP USE CASE

Spacebel has been developing a flight-qualified implementation of the CCSDS File Delivery Protocol (CFDP) [14]. As described here below, the CFDP is a very versatile protocol that offers a wide range of options, from simple one-way, best-effort point-to-point transfers to acknowledged exchanges transmitted via several waypoints. Because of the wide spectrum of possibilities offered by the protocol, a CFDP implementation has to be highly modular, in order to easily switch from one option set to the other, depending on specific missions' constraints and needs.

This modularity requirement actually makes the CFDP library development a use case of choice for the deployment of feature management. Indeed, it would not be reasonable to propose a complete, generic version of the library to an integrator, as only a subset of the functionalities of the protocol are likely to be used for a specific mission.

6.1. Overview of the CFDP

The CCSDS has developed the CFDP standard to complement the existing packet standards and to anticipate the needs of future missions. The CFDP proposes a file transfer protocol that efficiently copes with characteristics that are typical of the space data systems, missions and environment. The protocol can operate according to various profiles suited to specific mission needs and system constraint.

In its simplest form, the protocol provides Core Procedures with file delivery capability operating across a single link (i.e. point-to-point).

For more complex mission scenarios, where no direct link is available between file source and destination, the protocol offers Extended procedures and Store and Forward Overlay (SFO) procedures; both allowing end-to-end file transfers which can span multiple CFDP waypoint nodes thus providing end-to-end accountability through multiple hops capable of automatic store & forward operations.

In addition to the purely file delivery-related functions, the protocol also includes file manipulation commands and Filestore management services that provide control over the storage medium and management of the remote file systems.

The CFDP can therefore be used in various ways and support a wide range of options. Among others:

- Space-to-ground, ground-to-space and space-to-space directions of transfer are possible across an arbitrary network.
- The network can contain multiple links with disparate availability, as well as underlying subnetworks with heterogeneous protocols.
- The communication link can be unidirectional, half duplex, full duplex and can exhibit near-Earth or deep space delays.
- The file delivery path may contain one or several links and waypoints.

- The relaying of files at waypoints can be immediate or deferred upon complete file reception at waypoint.
- The file transfer can be reliable or on a best effort basis.
- Various retransmission strategies can be selected from unacknowledged to various flavours of acknowledged mode: positive, negative, deferred, asynchronous, immediate and prompted.
- Checksum can optionally be enabled on the individual packets and on the entire file.
- Proxy operations can be used to initiate the delivery of a file from a remote CFDP entity to some other user, either to the local user itself or to the user of some third CFDP entity.

6.2. Feature Management Applied to CFDP

Depending on the mission, each of the aforementioned capabilities can either be a mandatory feature or a useless waste of resources. Feature Management has thus been applied to the CFDP Flight library development to yield a generic building block that contain all the features but that can easily be tailored to strictly fit a particular projects needs.

The various protocol options have first been expressed as features, and the library feature diagram created. Roughly sixty features have been identified, spread on three hierarchy levels. Each feature has then been attached to software items such as packages, functions or even sections of code.

The code tagging approach described in section 4 has also been applied. It proved well integrated in the standard UML based development environment. It allowed automatically and optimally building versions of the CFDP Library corresponding to a selected and coherent set of features.

7. PROSPECTIVES

The Packet Utilization Standard is another typical example of an on board software building block. It also shows a high degree of variability from mission to mission.

One could therefore easily imagine applying the feature management process to a PUS implementation, probably with the services and

subservices as a starting point of the feature diagram, then refined with implementation options relevant to each of the subservices.

Subsequent projects relying on the PUS would then benefit of having a reusable building block tailorable to the mission with minimum effort and adapted to the available resources. The development effort would thus be limited to the interfaces with the lower levels of the software (accessing the avionics), which by definition tend to be more specific.

More generally, any software component which use is widespread in space missions can be a good candidate for a generic building block. Among those, components displaying a large number of options are the ones that should be targeted by a feature management approach. Examples of such components could be high-level FDIR applications, communication protocols, or equipment managers.

8. CONCLUSIONS

Feature Management is an efficient solution for managing the variability of generic software components that offer various options, profiles, capability sets or alternative implementations of a same function. It is an enabling technology for the efficient reuse of On Board Software Building Blocks.

9. REFERENCES

[1] D. Batory, D. Benavides, and A. Ruiz-Cortes. Automated analysis of feature models: Challenges ahead. *Communications of the ACM*, December 2006.

[2] Don S. Batory. Feature Models, Grammars, and Propositional Formulas. In *Proceedings of the 9th Int. Software Product Line Conference (SPLC)*, pages 7-20, 2005.

[3] David Benavides, Pablo Trinidad Martín-Arroyo, and Antonio Ruiz Cortés. Automated reasoning on feature models. In *Advanced Information Systems Engineering, 17th International Conference, CAiSE 2005*, pages 491-503, Porto, Portugal, June 2005.

[4] Andreas Classen, Patrick Heymans, and Pierre-Yves Schobbens. What's in a feature: A requirements engineering perspective. In José Luiz Fiadeiro and Paola Inverardi, editors, *Proceedings of the 11th International Conference on Fundamental Approaches to Software Engineering (FASE'08), Held as Part of the Joint European Conferences*

on Theory and Practice of Software (ETAPS'08), volume 4961 of LNCS, pages 16-30. Springer, 2008.

[5] Paul C. Clements and Linda Northrop. *Software Product Lines: Practices and Patterns*. SEI Series in Software Engineering. Addison-Wesley, August 2001.

[6] Ulrich W. Eisenecker and Krzysztof Czarnecki. *Generative Programming: Methods, Tools, and Applications*. Addison-Wesley, 2000.

[7] M. Griss, J. Favaro, and M. d'Alessandro. Integrating feature modeling with the RSEB. In *Proceedings of the Fifth International Conference on Software Reuse (ICSR)*, pages 76-85, Vancouver, BC, Canada, June 1998.

[8] K. Kang, S. Cohen, J. Hess, W. Novak, and S. Peterson. Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical Report CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie Mellon University, November 1990.

[9] Kyo C. Kang, Sajoong Kim, Jaejoon Lee, Kijoo Kim, Euseob Shin, and Moonhang Huh. Form: A feature-oriented reuse method with domain-specific reference architectures. *Annales of Software Engineering*, 5:14-168, 1998.

[10] Klaus Pohl, Gunter Bockle, and Frank van der Linden. *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer, July 2005.

[11] Pierre-Yves Schobbens, Patrick Heymans, Jean-Christophe Trigaux, and Yves Bontemps. Feature Diagrams: A Survey and A Formal Semantics. In *Proceedings of the 14th IEEE International Requirements Engineering Conference (RE'06)*, pages 139-148, Minneapolis, Minnesota, USA, September 2006.

[12] Pierre-Yves Schobbens, Patrick Heymans, Jean-Christophe Trigaux, and Yves Bontemps. Generic semantics of feature diagrams. *Computer Networks (2006)*, doi:10.1016/j.comnet.2006.08.008, special issue on feature interactions in emerging application domains, page 38, 2006.

[13] A. van Deursen and P. Klint. Domain-Specific Language Design Requires Feature Descriptions. *Journal of Computing and Information Technology*, 10(1):1-17, 2002.

[14] Laurent Demonceau, Paul Parisi, Massimiliano Ciccone, Gianluca Furano, Robert Blommestijn, CCSDS File Delivery Protocol for Future ESA Missions, DASIA 2008

[15] CCSDS File Delivery Protocol (CFDP), CCSDS 727.0-B-4, Recommendation for Space Data System Standards, Blue Book, Issue 4, January 2007

[16] CCSDS File Delivery Protocol (CFDP), CCSDS 720.1-G-2, Part1: Introduction and Overview, Green Book, Issue2, September 2003

[17] CCSDS File Delivery Protocol (CFDP), CCSDS 720.2-G-2, Part2: Implementer guide, Green Book, Issue2, September 2003

[18] CFDP Notebook of Common Inter-Agency Tests for Core Procedures, November 2001