Towards Explanation Generation using Feature Models in Software Product Lines

Dean Kramer, Christian Sauer, and Thomas Roth-Berghofer

School of Computing and Technology, University of West London,
St Mary's Road, London W5 5RF, United Kingdom
{first.lastname}@uwl.ac.uk

Abstract. Dynamic Software Product Line (DSPL) Engineering has gained interest through its promise of being able to unify software adaptation whereby software can be configured at compile time and runtime. Just like conventional adaptive software, software dynamism can confuse the user, and lower user trust. Variability knowledge expressed in a feature model though may not be understandable to the end user. Explanations have been shown to improve intelligibility of the software, and improve user trust. In this work, we consider how explanations can be used in DSPLs, by adding explanatory knowledge to feature models that can be used to generate explanations at runtime.

Keywords: Explanation Generation, Dynamic Software Product Lines, Feature Models

1 Introduction

Smart phones in recent years have seen high proliferation, allowing more users to stay productive while away from the desktop. It has become common for these devices to have an array of sensors including GPS, accelerometers, digital compass, proximity sensors, sound etc. Using these sensors with other equipment already found in phones, a wide set of contextual information can be acquired.

This contextual information can be used in Context-Aware Self Adaptive (CASA) software. This software can monitor different contextual parameters and dynamically adapt at runtime to satisfy the user's current needs [8]. These behavioural variations can be seen to share similarities with features in Software Product Lines (SPL), where product commonality and variability is handled, providing higher asset reuse. Within SPLs, Feature Oriented Software Development (FOSD) has emerged as a method for modularising the features of a system [3]. The one fundamental difference between these two concepts is that while SPLs conventionally manage static variability which is handled at compile time, adaptive software requires dynamic variability to be handled at runtime.

Dynamic Software Product Lines (DSPL) enables the SPL to be reconfigurable at runtime [9]. By using DSPLs, variability can be static, adapted at compile time, or dynamic and adapted at runtime. This allows for greater reuse

as variability can be implemented for both static and dynamic adaptation, as different products may require the adaptation to be applied at different times [14].

Feature Modelling has become the *de facto* method of variability representation, used in software product lines. In feature models, the adaptation of the product, be it static, or dynamic, are modelled, enabling a wide variety of products and product behaviours. While feature modelling is of great use in the development, the dynamics within feature modelling can be confusing to end-users. To amend the seemingly unpredictable and thus confusing nature of the behaviour of a dynamic system and the results it produces, it is desirable to enable the system to explain its behaviour as well as the results it produces to the end-user. As we will detail further on in this paper explanations are very useful to justify results a system produces and thus help to rebuild the trust an end-user has in the systems behaviour and results. So explanations are useful to the end-user as they can counter the mentioned non-transparency of DSPL end-products and their dynamic behaviours.

In our previous work [18], on enabling a system we developed to generate explanations, we investigated the integration of explanations into a context acquisition engine, used for developing context-aware applications. We did this with regard to mobile applications were one has to adhere to many constraints. We developed a *ContextEngine* to easier deal with such limitations and situation-specific information across applications [12], thus easing the creation of context-aware, mobile systems. We noticed that with the increased adaptability and dynamics of context-aware applications came an increase in complexity of the application, which in turn made it harder to understand the behaviour of such applications. In our initial research on this topic we then described how we enhanced the *ContextEngine* platform with explanation capabilities. As we describe in this paper and as it was proven in a variety of other work on explanations, explaining can be seen as complex reasoning task on its own. In our initial work we focused on the use of *canned explanations*. *Canned explanations* are information artefacts, pre-formulated by the software engineer, that serve as explanatory artefacts stored in the system and delivered to the user on demand. We integrated storage facilities for such information artefacts, or explanatory knowledge artefacts within the code structure of the *ContextEngine* and thus were able to provide these stored canned explanations on demand to a software engineer working with the *ContextEngine*. After this early steps and relatively simple approach, based also on a further study into the matter of explanation provision in the feature model and especially in the automated analysis feature models (AAFM) domain, we decided to elaborate on our initial work.

The rest of the paper is structured as follows: We introduce the feature modelling background of our work in the following section and based on the technological possibilities described there motivate our approach to use an extended feature model for explanation generation in Section 3. We then interlink our approach with related work on feature modelling, explanation generation and the use of explanations itself in the following section. We then introduce our approach to explanation generation from explanatory knowledge stored in an

extended feature model and demonstrate our concept of explanation generation in Section 5 . After discussing the advantages and possible disadvantages of our approach in Section 6 a summary and outlook on future aspects of our work concludes the paper.

2 Feature Models

The purpose of a feature model is to represent all possible products from a SPL in terms of features, and the relationships between them. An example feature model for a content store application is shown in Figure 1. A *feature* of a system

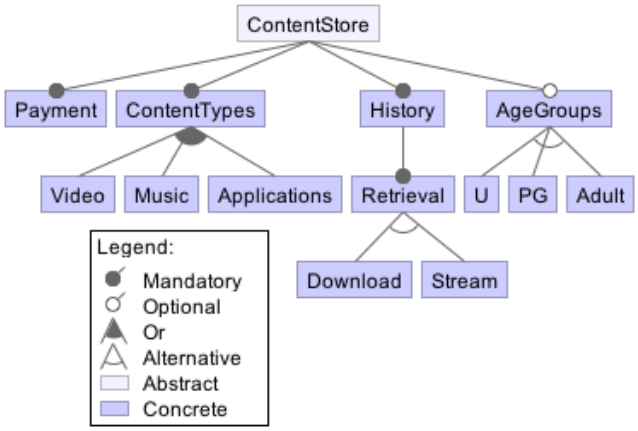

Fig. 1. Feature Model of a content store

has been described in a number of variations [2]. For this paper, we use the definition by Kang et al. [10] in that a feature is "*a prominent or distinctive user-visible aspect, quality, or characteristic of a software system or systems*". Feature models are modelled using hierarchical trees of features, with each node representing commonality and variability of its parent node. Relationships between each feature can be modelled using:

- Tree feature relationships between parent (compound) features and their child features (subfeatures) .
- Cross-tree constraints which typically apply feature inclusion or exclusion statements, normally using propositional formula. An example of this includes "if ABC is included, then feature XYZ must also be included."

Within feature models, different feature relationships can be applied including:

- **Mandatory**. A child feature is defined as mandatory in all products where its parent is also contained.

– **Optional**. A child feature is defined as optional when it optionally can be included or excluded when its parent is contained in a product.
– **Or**. A set of child features exhibit an or-relationship when one or more children are selected along with the parent of that set.
– **Alternative (XOR)**. A set of child features exhibit an xor-relationship when only a single child can be selected when the parent is included.

Feature models have been applied not only to modelling system features, but also context [1]. As DSPLs can be driven by context, modelling both contexts and the features that they affect in feature models allows for a single modelling language. The feature models introduced above represent what is known as *basic feature models*. There have been different additions to feature modelling, including *cardinality feature models* [7], and *extended feature models* [6].

Extended feature models extend basic feature models by the ability to attach additional information about features to the model. This additional information is included by the use of *feature attributes*, which are attached to features within the feature model. Feature attributes normally consist of a *name, domain,* and *value*. Feature attributes have been used in previous work for specifying extra-functional information [4]. We intend to also use feature attributes in our approach. We will employ additional feature attributes to store explanatory knowledge artefacts, see section 5.1 for details.

3 Motivation of our work

The GUI of an application, or even more intriguing, the behaviour of an application generated by the use of SPL can be rather dynamic. This dynamic behaviour can be confusing if not daunting to the end-user of the application. The end-user might not be aware of why the GUI has adapted and the factors influencing how it changes. Furthermore, the dynamic behaviour of the application, producing different results while receiving identical inputs just under different circumstances (for example a network being available or not), is a challenge to the trust the user develops towards the applications results. As the feature model, being the component responsible for the dynamic behaviour of the application, is a black box system to the end-user the need for explanations of this black box systems behaviour arises.

The benefits of being able to explain the behaviour of an application and subsequently its GUI are plenty. According to [17] there are a number of benefits explanations can provide to the end-user. The main benefits of interest with regard to our problem at hand are the generation of trust into the results the application generates and justification of and guidance on the changes in the GUI of the application.

As [6] have shown it can be a complicated process to apply abductive reasoning to generate minimal explanations from the logical representation of a feature model within an AAFM. To circumvent the effort involved in using abductive reasoning to generate a minimal explanations from the logical representation of the feature model our approach aims at integrating canned 'micro' or 'atomic'

explanations within the logical representation of the feature model. By doing so we aim to re-use the feature model itself in the same way it is used in the product configuration to also 'configure' or synthesise more complex explanations of the actual product generated from the 'atomic' building blocks given by the canned 'micro' explanations embedded in the feature descriptions themselves as well as in the representation of the relationships between these features described in the feature models logical representation.

3.1 Scenario Application

To illustrate our motivation, consider a DSPL example of a content store application for a mobile device. This application may provide different content for the user including applications, movies, music etc. Different content is organised into different categories. A simplified feature model of the DSPL can be seen in Figure 1. This application provides content for different age groups, and also the application can be tailored to suit these different groups.

In the feature model, we can see that the features *Payment, ContentTypes, History,* and *Retrieval* are required in every configuration of this DSPL. The *Payment* feature handles all payment transactions when content is bought or rented. The *ContentTypes* feature contains the different components for browsing, and buying different types of content. Because different regions in the world may require different content distribution licenses, it may not be possible to sell content in every region, so depending on the location of the user, different content type features including *Video, Music,* and *Applications* will be bound or unbound. In the *History* feature, all bought content is found, which can be retrieved in *Retrieval*. There are two primary methods in which content can be retrieved, downloaded or streamed. Certain content including video maybe downloaded or streamed. Depending on how much storage is available on the device, it may be not be possible to download the movie, so only the *Streaming* feature is bound. In Figure 2, we can see the variability of the screen according to content type features. If you consider the video feature, there is a button that takes the user to a set of screens for video content, and also a containership of widgets for advertising popular movies.

4 Related Work

Explanations and feature models have been used before, but more to aid the analysis process and error analysis of feature models [20] as well as in automated feature model analysis in general as Benavides et al. describe in [5]. As we already mentioned there are a number of goals that can be reached by providing explanations to the user of a, then, explanation aware system. An explanation aware system is a system that is able to provide explanations of the results it produces as well as of the means it employs to produce these results [11,13].

The goals pursued by enabling a system to provide explanations [19] are the following: Increase the transparency of a systems reasoning process to increase

Fig. 2. Variability of the main screen

the users trust into the system. Justifying results the system produces. This goal aims at explaining the quality and applicability of results the system produced to the end-user. Another goal of explanation provision is to provide relevance explanations of either question asked by the system or on information provided by the system. Conceptualisation, thus explanations of the concepts the system is working on, directly aids the last goal of providing explanations, learning. By explaining the concepts the system works on to the end-user the end-user is enabled to learn about the domain in which the system works.

Our approach to providing applications needs, next to the knowledge used by the feature model system, additional explanatory knowledge to create the explanations we want the system to be able to provide to the end-user. It is always necessary to provide explanatory knowledge in any system that is intended to provide explanations on its reasoning [16]. This additional explanatory knowledge is provided to and used by the explainer component of an explanation aware system, enabling it to provide explanations of its reasoning and results. The need for additional explanatory knowledge is also shown in [20] as the abduction process described there is relying also on additional explanatory knowledge.

In our approach the explanatory knowledge needed to generate explanations in our system will be broken down into 'atomic' canned explanatory knowledge artefacts that will be paired with each feature of the feature model as well as additional 'atomic' canned explanatory knowledge artefacts that will be attached to the relationship descriptors within our feature model. The aim of this 'enrichment' or 'dotation' of the feature model with 'micro' or 'atomic' explanatory

knowledge artefacts is it to reuse the artefacts 'bound' to the features and their relationship descriptors in the final product to synthesise complex explanations based on the available 'bound' atomic explanatory knowledge artefacts. We focus our approach especially on the issue of explaining a dynamic GUI to the end-user. As for example [15] described in their work the problems that can result from a dynamic and automatically assembled GUI, we aim to amend these problems by providing the end-user of an application with an insight into the GUI's changes by explaining them to her.

5 Our Approach

In our approach, we attempt to enable explanations in DSPLs. By adding explanations to DSPLs, we see two benefits. Firstly, explanations have been shown in other work to improve user understanding of a system which can be applied to DSPL systems [13]. Secondly, because a SPL enables many products to be produced using reusable common assets, we can then easily produce many explanation-ware applications, because we can leverage the reuse properties of the SPL with the explanations. The first part of our approach regards the modelling of the system.

5.1 Modelling

Just as the rest of the system is modelled using feature models, so too are the explanations. To add explanations to the feature model, we use extended feature models. As introduced earlier in the paper, with extended feature models, extra explanatory information can be attached to features using feature attributes. These feature attributes can be used for storing the specific explanation snippets for each feature. For each feature attribute there is a name, domain, and value. The domain of the attribute holds what type of explanation it is, with the value holding the explanation fragment.

Mapping explanatory knowledge fragments to features is not just enough; we also need to map explanatory knowledge fragments to feature model relationships. Examples of explanatory knowledge fragments mapped to relationships include:

- Mandatory - "in all configurations".
- Optional - "which is optional".
- Or - "which can be".
- Alternative - "which is either".

5.2 Composing Explanations

Once we have the explanations added to the feature model, we can compose complex explanations. These complex explanations are made up of the concatenation of explanations added to the features and the relationship explanations.

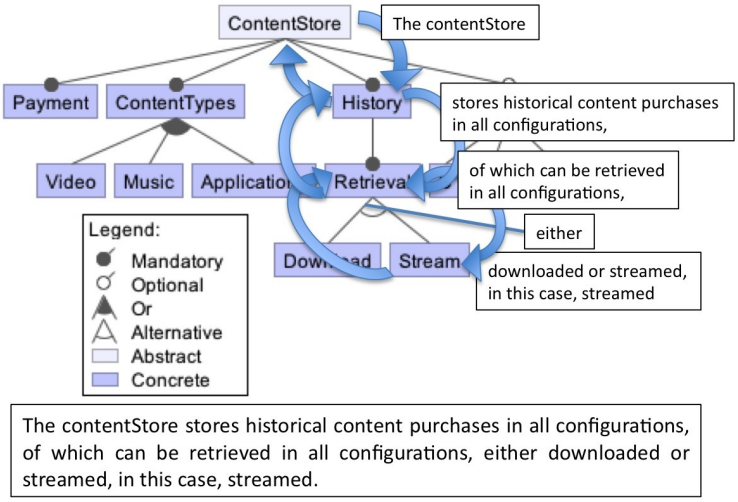

Fig. 3. Composing Explanation of why streaming is active

Lets take the example of considering a configuration where content is streamed to the user instead of downloaded as shown in Figure 3. If the user wanted to know *why* streaming is active, to generate an explanation we firstly follow the tree from the streaming feature to the root. We then get the conceptual explanation of the root, in this case "The content store". Next we add the conceptual explanation for the "History" feature, in this case "Stores historical content purchases", and because it is a mandatory feature, we add "in all configurations". Following this, we add the conceptual explanation for the "Retrieval" feature, in this case "of which can be retrieved", and add "in all configurations" because the feature is mandatory. Then because the sub features of "Retrieval" are alternatives, we add "either", and the two conceptual explanations joined with an "or". Lastly, because in the configuration, the "Stream" feature is active, we add "in this case, streamed". We therefore can 'reuse' the structural information encoded in the feature model representation directly for the composition of complex explanations by simply 're-tracing' the explanatory knowledge artefacts, stored in feature and relationship nodes, along a path in the feature model.

6 Discussion

The implementation effort is expected to be minor, given the fact that our approach just adds three additional feature attributes to the feature and relationship descriptions. The intention of 'piggyback' riding the inherent logical structure encoded in the feature model graph to also derive complex explanations from it is still open to be tested for its actual quality of generated explanations as well as for its scalability. With regard to the scalability of our approach we intend it to be limited. Once a feature model exceeds a certain complexity the

coherent concatenation of explanatory knowledge artefacts described in the feature and relation nodes along a path in such a model will fail or become too much of a computational effort. However we assume that for small to medium scale feature models our relatively 'practical' approach of concatenating explanatory knowledge artefacts stored in the models nodes is relatively efficient compared to existing more complex approaches of explanation generation explained, for example, in [6].

7 Summary and Outlook

In this paper we presented how the variability of SPL based products and their behaviours could be explained to their end-users using explanations composed from explanatory knowledge added to enhanced feature model nodes. Based on the feature modelling background of our work we motivated our approach to use an extended feature model for swift explanation generation. We reviewed and compared our approach with related work on feature modelling, explanation generation and the use of explanations itself, especially inspecting approaches employing relatively complex abductive reasoning. We then introduced our approach to explanation generation from explanatory knowledge stored in extended feature model nodes explaining features themselves and their relationships. By tracing an example graph in a feature model we showed the working of our approach. Given the fact that we are in the early stage of further examining our approach we then discussed its possible limitations but also its possible advantages given by the fact that our approach promises to be easily implemented base on existing work on enhanced feature models and is very easy to use for explanation composition in small to medium sized feature models, compared to more complex approaches like abductive reasoning.

As we cannot yet predict the effectiveness of our seemingly 'pragmatic' approach, we have to implement the enhanced node generation in our existing enhanced feature model and then perform a series of experiments on this enhanced feature model. The aim of these experiments will be to establish our approaches boundaries with regard to parameters such as quality and usability of generated explanations as well as the scalability of our approach. We also have to measure the computational effort necessary for explanation composition and then measure it against the gain in usability to establish the worthiness of further researching our approach of reusing extended feature model structures for explanation generation from explanatory knowledge artefacts stored in the nodes of the feature model.

An additional feature of the reuse of an enhanced feature models structure for explanation generation not yet investigated further is the reuse of propositional logic formulae derived from the feature model. We plan to investigate the possibilities of this reuse in our immediate follow up research.

References

1. Acher, M., Collet, P., Fleurey, F., Lahire, P., Moisan, S., Rigault, J.P.: Modeling Context and Dynamic Adaptations with Feature Models. In: 4th International Workshop Models@run.time at Models 2009 (MRT'09). p. 10 (Oct 2009)
2. Apel, S., Kästner, C.: An overview of feature-oriented software development. Journal of Object Technology 8(5), 49–84 (2009)
3. Batory, D., Sarvela, J., Rauschmayer, A.: Scaling step-wise refinement. IEEE Trans. Softw. Eng. 30, 355–371 (June 2004)
4. Benavides, D., Segura, S., Ruiz-Cortés, A.: Automated analysis of feature models 20 years later: A literature review. Inf. Syst. 35(6), 615–636 (Sep 2010)
5. Benavides, D., Segura, S., Ruiz-Cortés, A.: Automated analysis of feature models 20 years later: A literature review. Information Systems 35(6), 615–636 (2010)
6. Benavides, D., Trinidad, P., Ruiz-Cortés, A.: Automated reasoning on feature models. In: Proceedings of the 17th international conference on Advanced Information Systems Engineering. pp. 491–503. CAiSE'05, Springer-Verlag, Berlin, Heidelberg (2005)
7. Czarnecki, K., Helsen, S., Ulrich, E.: Formalizing cardinality-based feature models and their specialization. Software Process: Improvement and Practice 10, 7 – 29 (01/2005 2005)
8. Daniele, L.M., Silva, E., Pires, L.F., Sinderen, M.: A soa-based platform-specific framework for context-aware mobile applications. In: Aalst, W., Mylopoulos, J., Rosemann, M., Shaw, M.J., Szyperski, C., Poler, R., Sinderen, M., Sanchis, R. (eds.) Enterprise Interoperability, Lecture Notes in Business Information Processing, vol. 38, pp. 25–37. Springer Berlin Heidelberg (2009)
9. Hallsteinsen, S., Hinchey, M., Park, S., Schmid, K.: Dynamic software product lines. Computer 41, 93–95 (April 2008)
10. Kang, K., Cohen, S., Hess, J., Nowak, W., Peterson, S.: Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical Report CMU/SEI-90-TR-21 (1990)
11. Kofod-Petersen, A., Cassens, J.: Explanations and context in ambient intelligent systems. In: Modeling and Using Context, pp. 303–316. Springer (2007)
12. Kramer, D., Kocurova, A., Oussena, S., Clark, T., Komisarczuk, P.: An extensible, self contained, layered approach to context acquisition. In: Proceedings of the Third International Workshop on Middleware for Pervasive Mobile and Embedded Computing. pp. 6:1–6:7. M-MPAC '11, ACM, New York, NY, USA (2011), http://doi.acm.org/10.1145/2090316.2090322
13. Lim, B.Y., Dey, A.K., Avrahami, D.: Why and why not explanations improve the intelligibility of context-aware intelligent systems. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. pp. 2119–2128. ACM (2009)
14. Parra, C.: Towards Dynamic Software Product Lines: Unifying Design and Runtime Adaptations. Ph.D. thesis, INRIA Lille Nord Europe Laboratory (March 2011)
15. Pleuss, A., Hauptmann, B., Dhungana, D., Botterweck, G.: User interface engineering for software product lines: the dilemma between automation and usability. In: Proceedings of the 4th ACM SIGCHI symposium on Engineering interactive computing systems. pp. 25–34. ACM (2012)
16. Roth-Berghofer, T.R.: Explanations and case-based reasoning: Foundational issues. In: Funk, P., Calero, P.A.G. (eds.) Advances in Case-Based Reasoning. pp. 389–403. Springer-Verlag, Berlin, Heidelberg, Paris (September 2004)
17. Roth-Berghofer, T.R., Cassens, J.: Mapping goals and kinds of explanations to the knowledge containers of case-based reasoning systems. In: Case-Based Reasoning Research and Development, pp. 451–464. Springer (2005)

18. Sauer, C., Kocurova, A., Kramer, D., Roth-Berghofer, T.: Using canned explanations within a mobile context engine. Explanation-aware Computing ExaCt 2012 p. 26 (2012)
19. Sørmo, F., Cassens, J., Aamodt, A.: Explanation in case-based reasoning–perspectives and goals. Artificial Intelligence Review 24(2), 109–143 (2005)
20. Trinidad, P., Benavides, D., Durán, A., Ruiz-Cortés, A., Toro, M.: Automated error analysis for the agilization of feature modeling. J. Syst. Softw. 81(6), 883–896 (Jun 2008)