Towards a New Repository for Feature Model Exchange

José A. Galindo and David Benavides
Universidad de Sevilla
Sevilla, Spain
{jagalindo,benavides}@us.es

ABSTRACT

Feature models are one of the most important contributions to the field of software product lines, feature oriented software development or variability intensive systems. Since their invention in 1990, many feature model dialects appeared from less formal to more formal, from visual to textual, integrated in tool chains or just as a support for a concrete research contribution. Ten year ago, S.P.L.O.T. a feature model online tool was presented. One of its most used features has been the ability to centralise a feature model repository with its own feature model dialect. As a result of MODEVAR, we hope to have a new simple textual feature model language that can be shared by the community. Having a new repository for that language can help to share knowledge. In this paper we present some ideas about the characteristics that the future feature model repository should have in the future. The idea is to discuss those characteristics with the community.

CCS CONCEPTS

• **Software and its engineering → Software product lines**; *Re-quirements analysis*; *Software design engineering*; *Software imple-mentation planning*;

KEYWORDS

feature model repository,
characteristics,
variability,
requirements

1 INTRODUCTION

Feature models are one of the most important contributions since software product lines flourished [6] back in 1990. There are commercial and academic software product line development tools that use feature models as a cornerstone language. Most of these tools use their own feature modelling dialect. Often, the semantics are quite similar while there are differences among the concrete syntax used to describe feature models. There are many textual and graphical feature models dialects that can be found in the literature [5]. One of the ideas of the MODEVAR workshop is to find a common, simple feature model language that can be used to share knowledge among researchers.

Ten years ago S.P.L.O.T., a feature model online tool, was presented [8]. One of the most used characteristics of S.P.L.O.T was the feature model repository that allows to share models among researchers and practitioners. However, as other tools, S.P.L.O.T also defined its own feature modelling language and the interoperability with other tools was not always straightforward.

There is a strong effort around the world for what is called *Open Science* [9] that promotes – among other principles– the reproducibility of experiments and information sharing. We believe that the software product line community has to put some effort to follow this line. Although S.P.L.O.T. has played its role during these years, we think that a new feature model repository has to be set up with new characteristics, technologies and open science spirit.

In this paper we present some ideas about the elements that a feature model repository could have in the future. The idea is to discuss those elements among the community to join forces to set up a shared feature model repository for the next decades that can complement the new feature model language.

2 REPOSITORY CHARACTERISTICS

Following, we enumerate and explain some of the characteristics that we envision in a new feature model repository using the technique of *user stories* [2]. The characteristics were defined observing S.P.L.O.T. and other repositories such as Zenodo [1] and after some brainstorming discussions among the authors. We present the user stories following the template of *"As a [persona], I [want to], [so that]."* Then, we justify the potential importance of the user stories.

(1) *As a researcher, I want to upload models to the repository, so that my models are available for others.* This is one of the most basic features. It seems to be simple but some complexity can appear. For example, a syntax checker could be added to ensure that the uploaded models are syntactically correct. This raised an additional user story as described below. Also, a cross check validation could be implemented to approve or deny uploads according to some criteria. Also, depending on other user stories, some additional information to upload the models can vary.
(2) *As a researcher, I want to syntax check my models before uploading in the repository, so that my models are syntax error free.* This will clearly depend on the abstract and concrete syntax defined for the language.
(3) *As a researcher or user, I want to download models from the repository, so that I can replicate experiments or use existing models.* This is also a basic feature but could also face some

[1] https://zenodo.org/

complexity during its development. For instance, the way models are downloaded could vary from one by one to batch downloads. Designing a good solution for downloading models seems to be important to have a useful repository.

(4) *As a researcher, I want to generate a universal identifier for my models in the repository, so that my models are citable and easily identified.* A possible interesting characteristic of the repository could be to have a citable identified in the form of a DOI or something similar. The DOI could correspond with the DOI of a related research paper or could be a dedicated DOI just for the model or even a set of models. This is an interesting feature but may have a lower priority than the former ones.

(5) *As a researcher or user, I want to manage versions of my models, so that I can compare different versions.* This was a hardly discussed story among the authors. We see a potential of having models or benchmark versions but at the same time we see that this can be a characteristic that could make the repository more complex to maintain.

(6) *As a user, I want to search for models in the repository, so that I can find the models I am interested in.* A good search engine can help find models among the ones in the repository. Other search types could be implemented such as tag–based or author–based.

(7) *As a user, I want to display models in the repository, so that I can have a glance at the model in the case I want to use it.* Feature models of the repository could be used for benchmarking purposes but also for teaching or dissemination. In some cases, it could be useful to display the model. It has to be decided how to visualise the model. A textual visualisation will be available but other visual syntax could be defined/discussed.

(8) *As a user or researcher, I want to know some indicators about the models in the repository such as ratings or number of downloads, so that I can compare models.* This can allow to see how many times a model has been downloaded or rated. This could also help on the search user story described above.

(9) *As a developer or researcher, I want to have an API that can interact with the repository, so that I can programmatically access repository's information.* An interesting feature of a repository would be that the content could be accessed programatically in the form of an API. This could allow a developer to call a method to download some models with some characteristics for example such as a given number of features.

(10) *As a user, I want to have recommendations according to my profile, so that I can do better model selections.* If metrics of the models are stored according to downloads or user ratings, we could envision a recommendation system based on user profiles and ratings to better select models. This seems to be a more long term feature to be considered.

(11) *As a researcher, I want to create communities, so that I can have a common space to manage my models.* This characteristic has been discussed also in depth among the authors and no consensus has been reached. The idea is to allow the creation of communities inside the repository that could allow to group people around a community. It could be the case of a research group or university that has its own space to share their models.

(12) *As a user, I want to see model's metrics, so that I can have extra information about the models.* There are some implicit information in feature models such as the number of features, the number of relationships or other structural metrics [1]. Having the possibility to obtain this information available in the repository would be of help for people using it. Some of the information can be computationally hard to calculate (such as number of dead features) so a trade-off among information availability and computation capacity has to be defined.

3 INFORMATION REQUIREMENTS

To implement the features described in Section 2, there is some information that would be useful to store among the model:

- File. The concrete file of the model. This is, the serialization of the model itself so it can be used across different platforms and tools (e.g. FaMiLiar, FaMa or Feature IDE)
- Author. The author of the model. Information of the creator of the model so it can be used to credit authors.
- Owner. The owner of the model that can be different of the author. It might happen that different users took the time to translate and transcribe the model from other languages or visual formats.
- Hash. An identifier of the model that can be used also as a checksum. This is useful for the integration within other tools so we can avoid wrong downloads.
- Description. A brief description of the model.
- Organisation/Community. In the case Communities are allowed, this information has to be stored.
- Language level. In the case different language levels are defined for the language, the model has to define at which level it belongs to. This is, if we ended up having a level supporting attributes, we would need to specify if the model contains such information.
- Rating. In the case models ratings are allowed. We can think of popularity of models and ratings from the community.
- Tags. To associate free tags to a model that can serve as a way to search and filter the models
- Version. In the case that different versions of the same model are allowed. This is intended to allow the upload of different versions of the same model. For example, showing how a model has evolved.
- Model's metrics. In the case that these metrics are allowed, this information has to be stored. To know in advance some characteristics of the models such as te number of features or cross-tree relationships ratio.

4 DEPENDENCIES WITH LANGUAGES ELEMENTS

There are several elements of the language that will affect the development of the repository. Note that these aspects should be discussing during the workshop:

4.1 Concrete syntax

The serialization for the selected feature model concepts can impact how we store the models. In the literature we can find multiples styles [4] of serializations for feature models. However, there are three main styles: *i*) XML based, multiple tools such as FaMa, SPLOT and Feature IDE rely in such representation. This representation has as main drawback the size of storage of each model and how hard to read visually can be. The main benefit is that we can rely on multiple available tools to read and write XML files; *ii*) Bracelet-based format: There are also tools that moved towards a more json alike formats (e.g. FaMiLiar) in which the hierarchical relationships are developed based on brackets. This ends up resulting in lower size which is beneficial for transmission and storage of the models. Those models are also more easy to read by a human. *iii*) Plain text Formats: These file formats have defined custom syntax to define each element of the feature model notation they work with. Some tools that provide support for this formats are FaMa and Feature IDE.

4.2 Abstract syntax/Model levels

In the literature we can find multiple variants of feature models that are capable of representing different facets of variability. Usually, two different groups of relationships are defined: *i*) hierarchical relationships to define the options enabled by a variation point in a product line and *ii*) cross-tree constraints to define restrictions on features that do not share a common parent in the feature model tree. We can define up to five flavours of feature models:

Basic feature models. These set of models are the most constrained ones and its first introduced in the FODA study [7] paper. They only encode Boolean features; four types of structural relationships. Namely, mandatory, optional, set and or. Finally, two types of cross-tree relationships requires and excludes. The basic feature model defines four kinds of hierarchical relationships: *i*) **mandatory**; *ii*) **optional**; *iii*) **set** which includes **alternative**; and **or** relationships; In addition to these hierarchical constraints, two cross-tree constraints are defined: *i*) **requires**, and; *ii*) **excludes**.

Cardinality based feature models. This representation is an attempt to unify feature models and UML constraint notations. This feature models definition, replaces the definition of basic feature model relationships by cardinality-based relationships.

Complex-constraints feature models. This representation extends previous levels by allowing the definition of any complex cross-tree constraint.

Attributed feature models. Attributed feature models is a feature modelling extension that includes non-Boolean information about features. Complex constraints are allowed between features and attributes such as “Every attribute called cost must have a value greater than 10." There are a variety of approaches to describe feature model attributes, however, most of them share some characteristics such as name and value.

Non-traditional feature models. This last group of feature models extends the previous approaches by adding functionality to encode more complex variability. For example, in this group we can find models containing multi-features [3], when a feature can be present more than one time in a product.

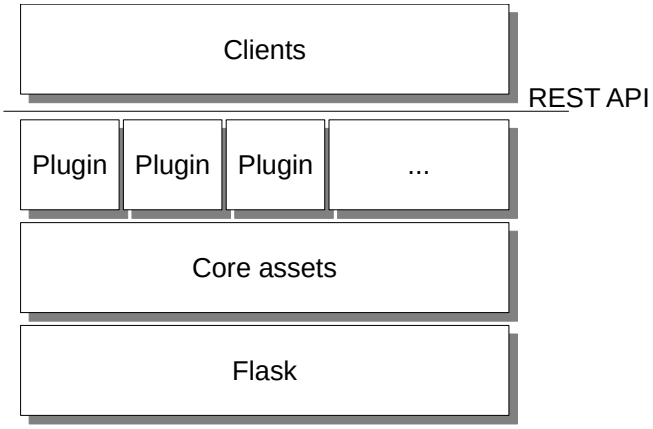

Figure 1: Possible architecture for our repository.

Our repository aims at supporting multiple flavours of the same model (e.g. with and without attributes) we would need to define the extension points of our language and the mechanisms to compose it before uploading it to the repository. That is, we would need to support versions of the same model but compatible with different levels of the language.

4.3 Technological stack to build the language infrastructure

We can summarise that the stack needs to fulfil the following requirements given the requirements shown in Section 2:

- Extensible. This repository feature responds to the need of providing rest interfaces, support for multiple attributes per model as well as the creation of communities.
- Light. Due the amount of optional features identified it would be interesting to rely on a small core.
- Reliable. There are several repositories that were built during the last years. It would be advisable that the core component of our repository is backed up by a larger project.

Figure 1 presents a possible architecture for our repository. First, we chose to select the micro framework Flask [2] which provides a light and extensive core for a web application and it is implemented in Python. This will enable the support of basic functionality such as accepting requests and build up our application. Second, a set of core assets providing the basic functionality of hosting and organising the models is needed. Then, we envision a plugin system to provide other functionality such as the evaluation of metrics for models or enable the rating of them. Finally, an API is provided to integrate the repository in different tools. Also, an HTML front-end will be built consuming such API.

Given those considerations, we propose to reuse and promote open-science previous efforts. Concretely, we found a solution stack

[2] http://flask.pocoo.org/

provided by Invenio [3] which was initially developed at CERN and holding an open-source licence.

5 CONCLUSIONS

We expect that the number of requirements would require some extensiveness in the repository so we can keep on adding new functionality as time goes by. For example, functionality such as the execution of analysis operations can be kept out of an initial version.

Also, this modular architecture would enable to adapt the repository for different levels of variability or serialisations. Another required main feature is the REST API so its easily interoperable with various tools and frameworks.

We think that the Invenio solution can serve as a base for building up a feature model repository that fulfills all our requirements. During the workshop, we aim to retrieve such information and reach consensus on the needs the community has.

ACKNOWLEDGEMENTS

This work has been partially funded by the EU FEDER program, the MINECO project OPHELIA (RTI2018-101204-B-C22); the Juan de la Cierva postdoctoral program; the TASOVA network (MCIU-AEI TIN2017-90644-REDT); and the Junta de Andalucia METAMORFOSIS project.

REFERENCES

[1] Ebrahim Bagheri and Dragan Gasevic. 2011. Assessing the maintainability of software product line feature models using structural metrics. *Software Quality Journal* 19, 3 (2011), 579–612. https://doi.org/10.1007/s11219-010-9127-2

[2] Mike Cohn. 2004. *User stories applied: For agile software development.* Addison-Wesley Professional.

[3] Maxime Cordy, Pierre-Yves Schobbens, Patrick Heymans, and Axel Legay. 2013. Beyond boolean product-line model checking: dealing with feature attributes and multi-features. In *Proceedings of the 2013 International Conference on Software Engineering*. IEEE Press, 472–481.

[4] Holger Eichelberger and Klaus Schmid. [n. d.]. Textual Variability Modeling Languages: An Overview and Considerations. In *Proceedings of the 1st International Workshop on Languages for Modelling Variability (MODEVAR 2019).*

[5] Holger Eichelberger and Klaus Schmid. 2015. Mapping the design-space of textual variability modeling languages: a refined analysis. *STTT* 17, 5 (2015), 559–584. https://doi.org/10.1007/s10009-014-0362-x

[6] José A. Galindo, David Benavides, Pablo Trinidad, Antonio-Manuel Gutiérrez-Fernández, and Antonio Ruiz-Cortés. 2019. Automated analysis of feature models: Quo vadis? *Computing* 101, 5 (01 May 2019), 387–433. https://doi.org/10.1007/s00607-018-0646-1

[7] Kang. 2010. FODA: Twenty Years of Perspective on Feature Modeling.. In *Proceedings of Fourth International Workshop on Variability Modelling of Software-Intensive Systems, ICB-Research Report, volume 37, Universität Duisburg-Essen, page 9.*

[8] Marcilio Mendonca, Moises Branco, and Donald Cowan. 2009. SPLOT: software product lines online tools. In *Proceedings of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications.* ACM, 761–762.

[9] Brian A Nosek, George Alter, George C Banks, Denny Borsboom, Sara D Bowman, Steven J Breckler, Stuart Buck, Christopher D Chambers, Gilbert Chin, Garret Christensen, et al. 2015. Promoting an open research culture. *Science* 348, 6242 (2015), 1422–1425.

[3] http://inveniosoftware.org/