AtlasSPL - A Web-Based Tool for Feature Modeling

Marcelo Schmitt Laser[1], **Elder Macedo Rodrigues**[1], **Cristiano Moreira Martins**[1], **Flávio Oliveira**[1]

[1]School of Computer Science (FACIN)
Pontifical Catholic University of Rio Grande do Sul (PUCRS)
Postal Code 90.619-900 – Porto Alegre – RS – Brazil

{marcelo.laser, cristiano.martins}@acad.pucrs.br

{elder.rodrigues, flavio.oliveira}@pucrs.br

Abstract. *This paper presents a graphical web-based feature modeling tool called AtlasSPL that aims to facilitate the training of new professionals in basic Software Product Line design concepts. AtlasSPL provides an intuitive graphical interface for the construction and visualization of feature models, is an accessible solution, gives access to a repository of previously-validated feature models, and supports five different feature model notations. We believe that these qualities define it as a highly accessible solution for both educational and small-scale industrial settings.*

1. Introduction

Software Product Line Engineering (SPLE) is a technique that has grown considerably in recent years. Due to allowing for a great degree of systematic reuse, as well as decreasing long-term costs and time-to-market, it has proven itself as a valuable technique in the design, development and management of software where a core of commonalities can be identified between different products [Clements and Northrop 2001] [Pohl et al. 2005]. A key concept of a Software Product Line (SPL) is the definition of variability, this being the characteristics (or features) that vary from one product to another within a single family of software products.

The clear and expressive representation of variability during the design of an SPL is invaluable to its continued success [Pohl et al. 2005]. While there are different techniques to aid in this task, Feature Modeling [Kang et al. 1990] [Czarnecki and Eisenecker 2000] is one of the most widely used [Berger et al. 2013], allowing for a graphical representation of variability that is recognisable by all stakeholders. Although there are several tools that aid in feature modeling, none of them meet our usability and availability requirements.

Various notations have been proposed to represent Feature Models, most of them based on FODA [Kang et al. 1990]. Some of the most notable ones in the literature are the Czarnecki-Eisenecker base [Czarnecki and Eisenecker 2000] and extended [Czarnecki et al. 2005] notations, the FeatuRSEB notation [Griss et al. 1998] and the Gurp-Bosch-Svahnberg notation [van Gurp et al. 2001]. Despite having important differences in presentation, they share a common set of semantics that can be exploited in the construction of a feature modeling tool.

Over the course of our work, we have often run into difficulties when having to build and present feature models, for reasons ranging from lack of knowledge from our collaborators to lack of support in the tools available to us. To resolve this issue, we have defined a set of requirements, based on our expertise in this field, for a tool to support feature modeling with the purpose of basic education in software variability and feature modeling, as well as facilitating the collaboration between professionals who may not be acquainted with abstract structures such as trees.

In this paper we present AtlasSPL, a web-based feature modeling tool built to meet these requirements. AtlasSPL was created with the primary purpose of training our collaborators in the use of feature models, and should therefore be a suitable tool for educational environments. We believe that it is also sufficiently complete to allow for the basic feature modeling activities of managing SPL variability, and is therefore suitable for small-scale industrial use.

The rest of this paper is organized as follows: Section 2 presents some background information regarding feature models and their applicability in Software Engineering, as well as some tools that aid in the feature modeling process; Section 3 presents our in-house feature modeling tool (AtlasSPL), along with a summary of the requirements on which this tool's design is based and an example of its application through the modeling of a pedagogical SPL; Section 4 presents some lessons learned from the development of these tools as well as some proposals of future work; Section 5 presents a summary of our conclusions based on the material presented in this paper.

2. Background

In this section we briefly present the basic concepts of feature modeling. This is followed by the descriptions of four tools that support the feature modeling process, along with the merits and flaws that we have perceived in them. Finally, we provide a contextualization of our own experiences in this field of research, along with the motivations for the development of a new feature modeling tool.

According to [Czarnecki and Eisenecker 2000], "a feature is an important property of a concept instance", representing any commonality or difference between the different products of an SPL. A model that groups the representation and relationships between the features of an SPL is called a Feature Model, which can be comprised of one or more Feature Diagrams. Feature Models are commonly used for the representation of variability in SPLs, both for their simplicity and expressiveness, and for the ease with which they can be explained to stakeholders, specially those from outside the field of Software Engineering.

A feature model is typically comprised of a root feature, representing the domain of the SPL, and several child features representing variation points of this domain. These variation points may in turn have child features of their own, either to further subdivide them into smaller variation points or to represent variants available to the SPL.

2.1. Related Feature Modeling Tools

Although a number of tools exist that aid in the feature modeling process, the better portion of them appears to be experimental or incomplete. Prior to our decision to develop a tool of our own, as well as to guide us in its design, we have identified three major tools that stand out as being in a more mature stage. We have also used a tool previously designed by our own research group during previous projects as a basis for the conception of our work.

FeatureIDE is a framework based on Eclipse and developed to support Feature-Oriented Software Development (FOSD) [Thüm et al. 2014]. Although FeatureIDE is undeniably a powerful tool, used primarily for the purposes of teaching and research, its target user is a professional who already has a certain knowledge of feature modeling and SPL concepts. Conversely, our target users are students and professionals who are entirely unacquainted with these concepts, and our target activities include the design and visualization of SPL features.

FeaturePlugin is a feature modeling plug-in for the Eclipse IDE, and proposes to integrate feature modeling with a complete and established development environment [Antkiewicz and Czarnecki 2004]. It provides support to feature modeling through the use of tree structures and logical statements, and integrates feature modeling and feature-based configuration. Though it is a complete solution to feature modeling, it is restricted to a single environment (Eclipse) and does not provide a graphical editor, while we seek greater intuitiveness and compatibility.

The Software Product Line Online Tools (S.P.L.O.T.) are a collection of web-based tools that include a feature model editor with advanced constraint definition and validation techniques [Mendonça et al. 2009]. S.P.L.O.T. contains a vast repository of models which are available to the public, making it a substantial information base for feature modeling practices. The tool does not, however, offer any means of graphical construction of feature models, being based instead on a tree structure. The use of this kind of structure could be inadequate for presentation purposes where the audience is unacquainted with certain basics of Computer Theory, such as tree structures.

PlugSPL, the in-house tool on which AtlasSPL is largely based has been described extensively elsewhere [de M. Rodrigues et al. 2014]. Regardless, it is important to assert those characteristics of it that are specific to the SPL Design activity, both in regards to what is being adopted from it as to what components had to be developed or adapted for reuse. It is also important to highlight that PlugSPL is a stand-alone tool that supports several phases of the SPL development cycle, and was therefore built around very different requirements from AtlasSPL.

Our previous approach to the development of the SPL Design module, and indeed for the entire tool, had been to develop an extensible plugin-based tool written in C# and run directly from the user's machine. Our experience with this previous project has shown us that for the purposes of accessibility and ease of collaboration, it would be advantageous to turn instead to a web-based User Interface (UI). Though the original architecture had been made in a modular manner, such a drastic change to the mode of input/output creates certain issues that have to be addressed.

Another important difference between PlugSPL and AtlasSPL is the decision to have native support of a number of feature model notations, allowing each user to decide at the time of a model's creation which one to base it on. We believe that this allows for greater accessibility, as well as enabling the user to learn about the more common notations found in the literature.

Finally, we have chosen to reuse some of the assets created for PlugSPL, *viz.* the persistence structures and the validation code executed in their construction. Adapting these structures for use in a broader scope (more than one notation) and to operate with web-based technologies was yet another challenge in the design and development of our new tool.

2.2. Context

Over the course of our group's research into Software Testing[1], we have developed a family of Software Testing Tools by use of SPL practices [de M. Rodrigues et al. 2010] [Silveira et al. 2011] [Costa et al. 2012]. Given the volatile nature of our research environment, we have had to repeat certain processes several times, particularly in what relates to product configuration, product generation [de M. Rodrigues et al. 2014].

In order to eliminate these repeated efforts, we have proposed and developed an in-house tool (PlugSPL) that supports several phases of the SPL development cycle [de M. Rodrigues et al. 2014]. Among the requirements of that tool we identified the support for graphical-based notation for designing feature models. Though the resulting tool was satisfactory in resolving the requirements we had at the time, it was made primarily for use by a core group of the research team that already had experience with feature modeling. Over the course of the following years, we have found that PlugSPL could be difficult to use by new collaborators, particularly in what refers to the SPL Design activity.

Given that feature modeling is a crucial activity during SPL design, we sought out means by which to train our collaborators in their use. Furthermore, it is widely accepted that feature models are of easier presentation to stakeholders who are not versed in variability analysis than other similar representations [de M. Rodrigues et al. 2014]. For this reason, we also sought tools that would allow collaboration during the design of feature models.

Despite their merits, the results that we turned up in regards to actual tools were scarce. Some major qualities that were lacking in these tools were simplicity, ease of use and accessibility, which rendered them inadequate to our needs in feature modeling practices. We also found the existing tools lacking in terms of presentation due to their complexity. In order to facilitate the understanding of variability management, as well as to provide a shared and collaborative environment for feature modeling, we decided to create an in-house web-based tool for graphical feature modeling.

3. AtlasSPL - Feature Model Editor Web-Tool

AtlasSPL is a web-based feature modeling tool that was designed and developed with the qualities of ease-of-use and versatility at the forefront of our team's goals. It is the culmination of six years of study and experience with SPL design, working with a volatile team whose members were constantly being shifted in and out of projects, with new members coming into the team often without any knowledge of SPL or feature modeling concepts. With AtlasSPL, we hope to greatly facilitate the training of new professionals in feature modeling and design, as well as basic notions of Software Product Lines.

This section presents: the design decisions taken for the development of AtlasSPL, as well as the requirements that drove them; a complete description of the functionalities

[1]`www.cepes.pucrs.br`

of the tool as well as the ways to access it; a brief description of the feature model notations supported by the tool, and; an explanation of the validation process applied by AtlasSPL.

3.1. Requirements and Design Decisions

In order to present the current state of our studies in feature modeling practices, as well as the needs we perceived for the training of our collaborators, we have defined some requirements that must be addressed by a feature modeling tool. These requirements refer primarily to the usability and functionality of the tool, and do not largely concern themselves with architectural or performance constraints.

In our context, the first and most important requirement is ease of access and portability. If the tool is dependent on a particular machine configuration, operating system, language, development environment or framework, or if it requires prior installation and preparation, it could be limited in its applicability. This requirement has been the driving reason to our decision of making a web-based tool, resolving all of these dependencies so long as the client machine has a connection to the Internet.

Another requirement is the availability of multiple feature modeling notations, which is particularly important for processes of training and collaborative work. We judge that training in more than one feature model notation is essential to the comprehension of the basic feature model semantics, enabling the trained professional to design more complete and expressive models. Furthermore, the collaborative nature of SPLs demands that professionals be able to apply different views of the same concept in order for a better exchange of ideas to be reached.

The tool must also possess an intuitive user interface based on the graphical representation of feature modeling objects. Given that feature models are ultimately a graphical artifact, we find it important for the user to be given a “canvas” with which to represent them, rather than depending on textual input and tree structures for their design.

It is valuable, particularly in a tool directed at training and education, that there be feature model validation functionalities. Given that our aim is to provide the very first steps of training in feature modeling practices, we have opted to provide support to structural validation. To achieve this we have largely reused one of our research team’s previous assets.

Finally, we believe it advantageous to have an easily accessible repository of reference models. We have opted to give the user access to previously-validated models, as well as the option to add new models to the public repository. The repository also has the option of allowing users to create private access groups, permitting the collaboration between closed teams without the support of file-sharing solutions.

3.2. Tool Functionalities

AtlasSPL provides the user with the most basic operations required for feature modeling, these being: adding, moving and removing features and relationships; setting feature properties; renaming features; creating, saving and loading feature models, and; validating feature models. These functions were built into the tool in the way the authors judged to be the most intuitive for the user. Figure 1 shows the user interface for AtlasSPL and presents an example feature model in construction, drawn from the Arcade Game Maker Pedagogical Product Line (AGM) and using the FODA notation.

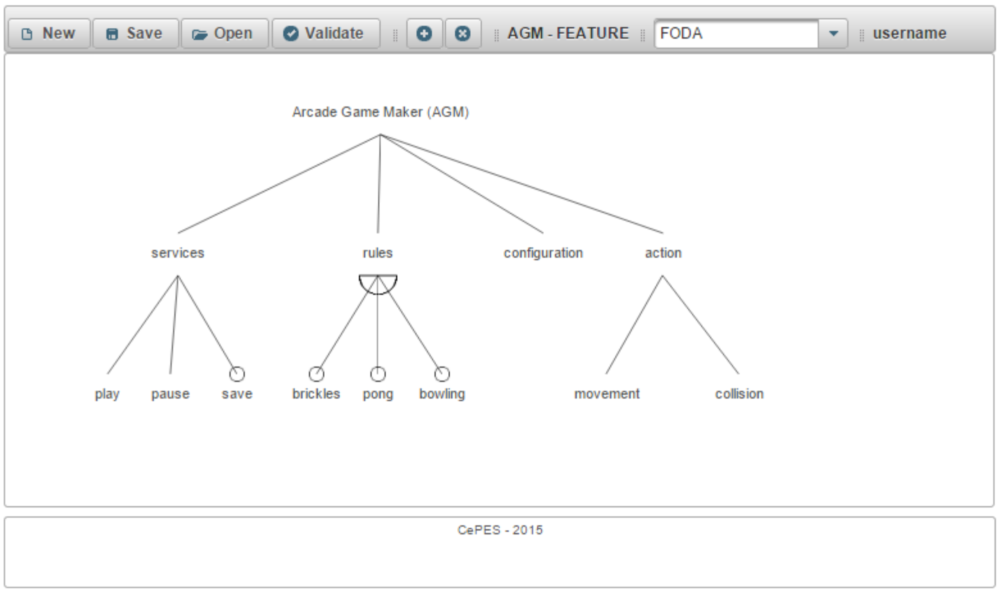

Figure 1. AtlasSPL Webtool - AGM Feature Model using FODA

Upon clicking the "New" button, a dialogue menu is opened (see Figure 2) where the user may name the new feature model and select the feature model notation to be used (refer to Subsection 3.3 for the available notations).

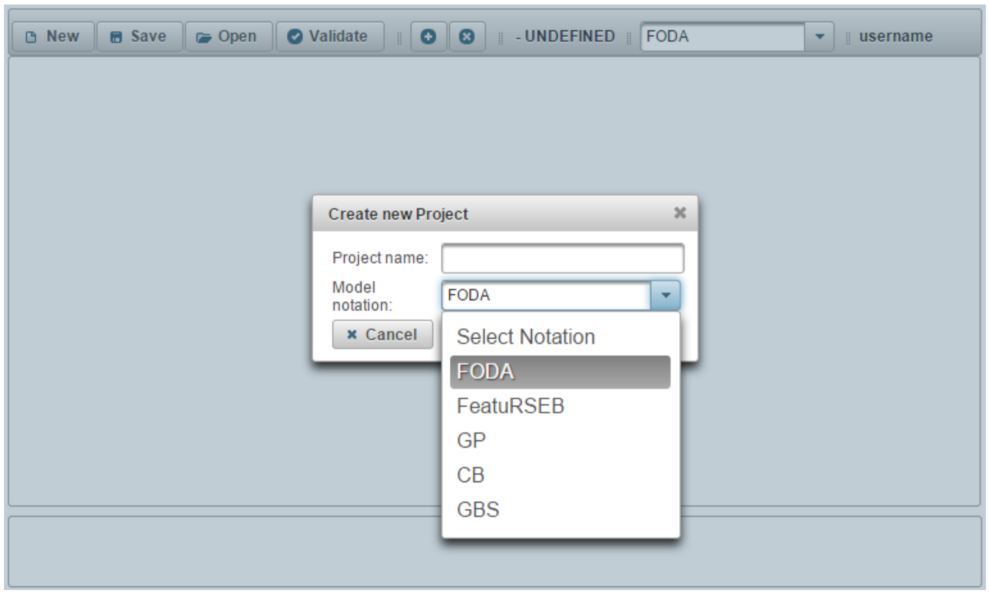

Figure 2. New feature model dialogue menu

There are three ways for the user to add a new feature to the model, allowing for greater efficiency. The user may click the "+" button and then click anywhere inside the canvas. The user may also right-click anywhere inside the canvas and select "New Feature" from the context menu. Finally, the user may double-click anywhere inside the canvas. Any of these actions will result in a feature object being created at the current cursor position and a dialogue menu being opened to name it.

To remove a feature, the user must right-click on that feature and select “Remove Feature” from the context menu. In future versions of AtlasSPL, it will also be possible to select a feature by clicking it and then deleting the selected feature. To move a feature, the user may click it and drag it over the canvas. To rename it, the user must right-click the feature and select “Feature Name” from the context menu.

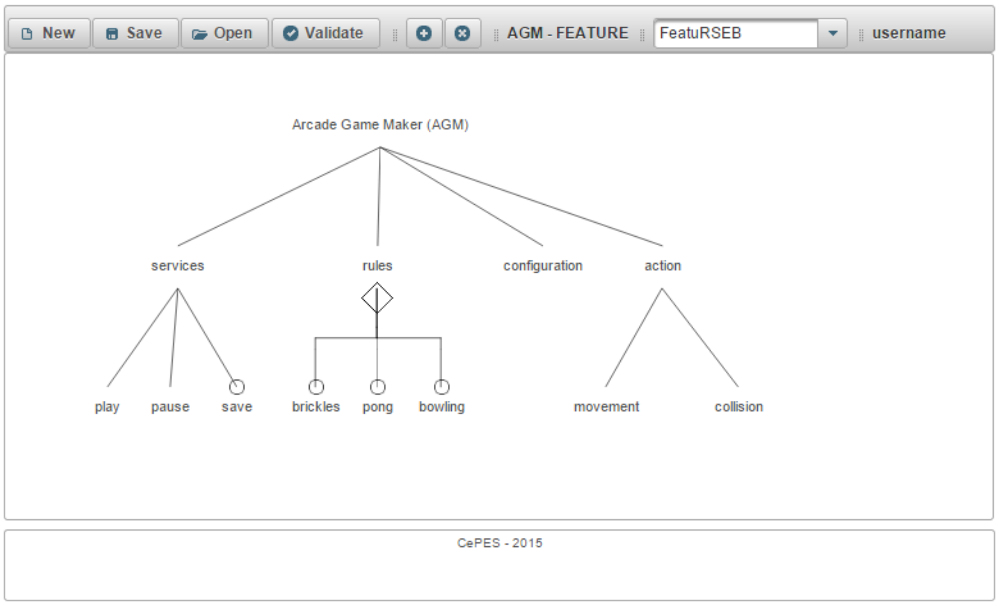

Figure 3. AtlasSPL Webtool - AGM Feature Model using FeatuRSEB

To add a relationship, the user must click anywhere upon the lower edge of the desired parent feature (the area will be highlighted when moused over) and drag the cursor towards the child feature. By clicking on an existing relationship and dragging the cursor over to another feature, the relationship’s child feature may be changed. By clicking on an existing relationship and dragging the cursor to an empty area of the canvas, the relationship is removed.

The appearance of a relationship is dependent on the properties of its child feature and the selected feature model notation. For example, Figure 1 shows the alternative relationship between the subfeature *rules* and its variants *brickles*, *pong* and *bowling* as a semi-circle, in accordance to the FODA notation. Meanwhile, Figure 3 represents this as an XOR relationship by using the unfilled diamond shape and straight lines, in accordance with FeatuRSEB.

To set a feature’s properties one must right-click it and select the desired relationship type from the context menu. The available relationship types vary according to the feature model notation in use.

Finally, to conduct the structural validation of a feature model, the user must click the “Validate” button, which will attempt to reconstruct the diagram currently on the canvas. After the validation process is completed, a dialogue box will open to present the results, either stating “Valid Diagram” or “Invalid Diagram”. Figure 4 shows an example of the tool declaring a feature model as invalid.

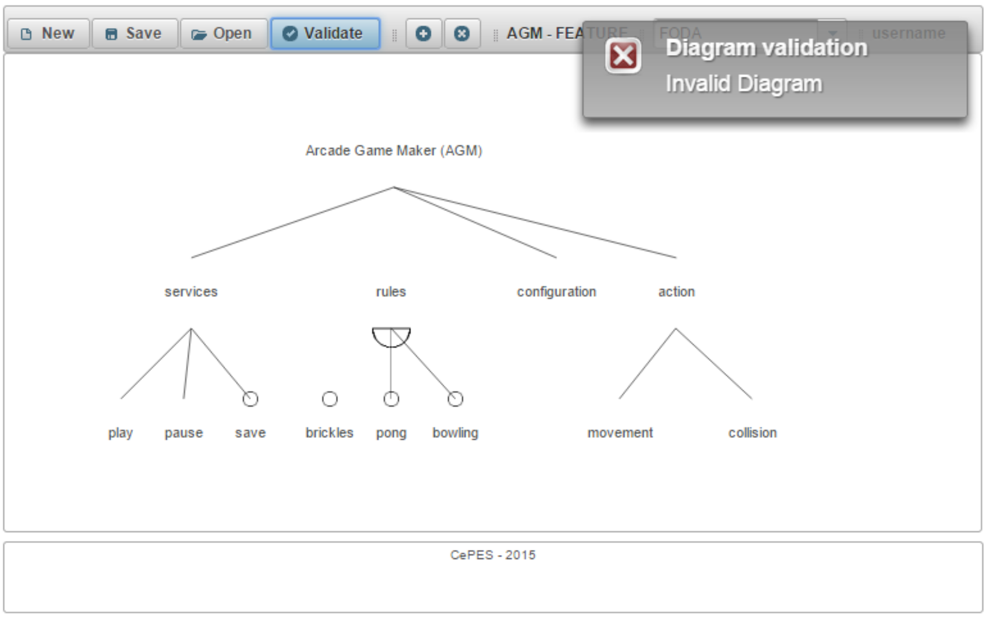

Figure 4. Validation dialogue box

For a more thorough validation, presenting data such as the number of valid configurations that may be derived from the feature model and the number of dead features within that model, AtlasSPL allows the user to export a feature model in the S.P.L.O.T. native XML format.

3.3. Adopted Feature Model Notations

We have selected five feature model notations for which to provide native support. This selection was based on our own experience, as well as brief literature searches by the authors into surveys of and comparisons between feature model notations. The selected notations are:

- FODA [Kang et al. 1990];
- Czarnecki-Eisenecker Base Notation [Czarnecki and Eisenecker 2000];
- Czarnecki-Eisenecker Extended Notation [Czarnecki et al. 2005];
- FeatuRSEB [Griss et al. 1998];
- Gurp-Bosch-Svahnberg Notation [van Gurp et al. 2001].

The Feature-Oriented Domain Analysis notation (FODA), which is the original feature modeling notation, can be used as a basis for the study of feature model semantics. It presents all of the main components of a feature model, these being features themselves, relationships, the notions of mandatory (must be present in all product configurations) and optional (may or may not be present in a product configuration) features, and the notion of alternative (mutually exclusive) features.

The Czarnecki-Eisenecker Base notation (also known as the Generative Programming notation, hereby referred to as GP) has been broadly used as the primary reference notation for feature models. Building on FODA, GP permits all of the original components of its predecessor using different graphical representations, and proposes new components to allow for greater expressiveness. The alternative features are now called an XOR feature group, to differentiate them from the new OR feature group (one or more may be present in a product configuration).

The Czarnecki-Eisenecker Extended notation (also known as the Cardinality-based notation, hereby referred to as CB) is an extension of the GP notation, presenting a new and more precise form by which to graphically represent relationships between features. By allowing the user to define a specific cardinality for each relationship, it is possible to precisely determine the occurrence of features in product configurations.

FeatuRSEB proposes to integrate FODA with Reuse-Driven Software Engineering Business (RSEB), an UML-based process, and presents feature models primarily from the point of view of the developer, or "reuser". FeatuRSEB deals with binding time directly by specifying whether an alternative relationship is resolved statically (XOR) or dynamically (OR).

The Gurp-Bosch-Svahnberg notation (GBS) proposes a framework to organize the approaches that existed at the time into. This notation is also primarily characterized by the definition of binding time within relationships, as well as presenting the idea of external features, features that are provided by a certain configuration's target platform and are therefore important to that configuration, but which are not part of the system itself.

3.4. Feature Model Validation

AtlasSPL provides structural validation of feature models by use of a number of procedures that check the consistency of a model by reconstructing it in the form of a restrictive data structure. This data structure was designed to only allow the instantiation of feature models if they are built in accordance to the guidelines of a given notation. Any structural inconsistencies in a feature model will result in a failure to instantiate this data structure, which AtlasSPL responds to by declaring the feature model as invalid.

While the algorithms by which this structural validation is executed are beyond the scope of this work, we present now the particular inconsistencies that are validated by AtlasSPL.

- There must be only one root feature within a feature diagram. Should there be more than one root feature in a single diagram, the feature model is declared invalid.
- All features must be connected by relationships. Should there be any features that have not been connected to the rest of the diagram, the feature model is declared invalid.
- All features must be reachable from the root feature through a single set of relationships. If AtlasSPL identifies more than one path from the root feature to any other given feature, the feature model is declared invalid.
- All features must be given a name (label) and all names must be different from one another. Should any two features share the same name, or should any one feature have the empty string for a name, the feature model is declared invalid.

4. Future Work

This section presents the lessons learned during the development of this research, as well as prospects for future research based on them.

- Given the successful application of feature modeling concepts within a web environment that was achieved by this project, the authors believe it would be advantageous to expand upon this initiative and build modules to support other parts of the SPL development cycle. While feature modeling has proven to be one of the most difficult topics in training new collaborators, it is also important to acquaint them with other SPL concepts.
- The extension of AtlasSPL to deal with the design and validation of logical constraints may prove useful for more advanced training. Furthermore, it would enable the tool to fully support the SPL Design activity within a production setting.
- The creation of a common data structure for multiple feature model notations may enable AtlasSPL to serve as a centralizing tool for feature modeling. It is likely that we may approach this matter through the analysis of semantic commonalities between notations, as well as the creation of parsers for existing feature modeling tools.
- We are aware that the tool must be improved to meet the requirements of different environments, especially in regards to providing better support for Feature Model analysis and validation. Moreover, the tool must provide detailed feedback to the user regarding any modeling mistakes so as to be usable by inexperienced users without the presence of an experienced tutor.
- Studies to empirically evaluate the success of AtlasSPL in achieving its proposed goals would solidify it as an educational tool. The authors are particularly interested in studying the intuitiveness of its UI, as well as the degree by which it aids in the teaching of basic feature modeling concepts.

5. Conclusion

AtlasSPL is a feature modeling web-tool proposed for the purpose of easing the training of professionals entirely unacquainted with feature model and SPL concepts. It aims to achieve this goal by providing a simple and accessible interface, based on graphical components to aid in the construction of feature models as well as their visualization. Furthermore, AtlasSPL was designed to be sufficiently complete so as to serve as an initial tool for the inception and presentation of new SPLs.

The tool offers native support to five different feature modeling notations, these being the Feature-Oriented Domain Analysis notation, the Czarnecki-Eisenecker base and extended notations, the FeatuRSEB notation and the Gurp-Bosch-Svahnberg notation. By allowing the user to choose between these notations, AtlasSPL enables different degrees of precision and simplicity, as well as different semantics to facilitate collaboration between users familiar with different notations.

Acknowledgements

Study partially developed by the Research Group of the PDTI 001/2012, financed by Dell Computers of Brazil Ltd. with resources from Law 8.248/91. We thank Dell for the support in the development of this work.

References

Antkiewicz, M. and Czarnecki, K. (2004). FeaturePlugin: Feature Modeling Plug-in for Eclipse. In *Proceedings of the 2004 OOPSLA Workshop on Eclipse Technology eXchange*, pages 67–72, New York, NY, USA. ACM.

Berger, T., Rublack, R., Nair, D., Atlee, J. M., Becker, M., Czarnecki, K., and Wasowski, A. (2013). A survey of variability modeling in industrial practice. In *Proceedings of the Seventh International Workshop on Variability Modelling of Software-intensive Systems*, New York, NY, USA. ACM.

Clements, P. C. and Northrop, L. (2001). *Software Product Lines: Practices and Patterns*. SEI Series in Software Engineering. Addison-Wesley.

Costa, L. T., Czekster, R., Oliveira, F. M., Rodrigues, E. M., Silveira, M. B., and Zorzo, A. F. (2012). Generating Performance Test Scripts and Scenarios Based on Abstract Intermediate Models. In *24th International Conference on Software Engineering and Knowledge Engineering*, pages 112–117.

Czarnecki, K. and Eisenecker, U. (2000). *Generative Programming: Methods, Tools, and Applications*. Addison-Wesley, Boston, MA.

Czarnecki, K., Helsen, S., and Eisenecker, U. (2005). Staged configuration through specialization and multi-level configuration of feature models. In *Software Process Improvement and Practice*.

de M. Rodrigues, E., Passos, L., Teixeira, F., Zorzo, A. F., and Saad, R. (2014). On the Requirements and Design Decisions of an In-House Component-based SPL Automated Environment. In *26th International Conference on Software Engineering and Knowledge Engineering*, pages 483–488.

de M. Rodrigues, E., Viccari, L. D., Zorzo, A. F., and Gimenes, I. M. (2010). PLeTs-Test Automation using Software Product Lines and Model Based Testing. In *22nd International Conference on Software Engineering and Knowledge Engineering*, pages 483–488.

Griss, M., Favaro, J., and d'Alessandro, M. (1998). Integrating feature modeling with the RSEB. In *Proceedings. Fifth International Conference on Software Reuse*, pages 76–85.

Kang, K., Cohen, S., Hess, J., Novak, W., and Peterson, A. (1990). Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical Report CMU/SEI-90-TR-021, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA.

Mendonça, M., Branco, M., and Cowan, D. D. (2009). S.P.L.O.T.: software product lines online tools. In Arora, S. and Leavens, G. T., editors, *OOPSLA Companion*, pages 761–762. ACM.

Pohl, K., Böckle, G., and van der Linden, F. J. (2005). *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer, 1 edition.

Silveira, M. B., Rodrigues, E. M., Zorzo, A. F., Vieira, H., and Oliveira, F. (2011). Model-Based Automatic Generation of Performance Test Scripts. In *23rd International Conference on Software Engineering and Knowledge Engineering*, pages 1–6, Miami, FL, USA. Knowledge Systems Institute Graduate School.

Thüm, T., Kästner, C., Benduhn, F., Meinicke, J., Saake, G., and Leich, T. (2014). FeatureIDE: An Extensible Framework for Feature-oriented Software Development. *Science of Computer Programming*, 79:70–85.

van Gurp, J., Bosch, J., and Svahnberg, M. (2001). On the notion of variability in software product lines. In *Proceedings Working IEEE/IFIP Conference on Software Architecture*, pages 45–54.