6
Final Remarks and Future Work

In this thesis we presented a point of view that the existing support for engineering enterprise software product lines can be significantly improved by taking a domain-specific modeling perspective. We presented domain knowledge modeling languages that formalize domain-specific concepts and their instantiation constraints, in addition to express the way in which domain-specific concept instances implement features.

The view of a software product line as a composite of domain-specific concepts was critical for improving feature visualization. The set of mappings to source code instantiating concepts is an important contribution as it enables easy navigation from models to source code and also projecting features assignments on physical assets. We showed that both capabilities are fundamental to correctly comprehend the configuration knowledge specification.

Another important contribution is the demonstration that DKMLs can be used for many different purposes. We explored the automatic interpretation of the metamodel to provide guidance and ensure consistency of the configuration knowledge in terms of the semantic expressed by the feature model. Disciplined assignment of features to domain knowledge model's elements restrict assignments such that no violation of the programming interface can occur during product derivation. Instead of assigning features to physical assets, disciplined assignment in our case map features to virtual elements of an instantiated framework. On top of correctly built domain knowledge models, a constraint programming modeling of the configuration knowledge detects errors in the entire product line in a single step, instead of the need of compiling and execute each product in isolation. Our constraint programming model checks the configuration knowledge against the feature model semantics. Among other, it ensures that a framework concept instantiation cannot be assigned to a optional feature and removed if it is still used by concept instances not removed from the same product.

The proposed technique for engineering framework-based software product lines was designed to be an uniform way to assign feature to source code and a reusable-programming model that can be easily adopted in existing

projects. It is easy to build DKMLs for different domains of application using the tool support provided by GenArch$^+$. We showed using numerous examples of DKMLs for several frameworks that this is valid assumption. The structure based on classes and references decorated withthe knowledge about the configurability semantics was enough to model the programming interface provided by all investigated frameworks. We also demonstrated that the closing mapping between DKMLs abstract syntaxes and the framework's programming interfaces supports the efficient projection of domain knowledge models from existing source code.

Finally, we compared domain knowledge models to other code-oriented techniques regarding some criteria: modularity, complexity, effort required to correct comprehend the product line. Concerned with modularity, we realized that the use of domain knowledge modeling languages, in general, reduces feature scattering and tangling. However, these benefits depends on some properties of the features. For instance, there is not way to modularize features not associated to framework-provided concepts in domain-knowledge models. Moreover, we observed that domain-knowledge models substantially increase the number of elements in the configuration knowledge, which might represent a negative impact on the effort need to specify and comprehend the configuration knowledge. However, we have initial evidences that most of the effort required to specify the configuration knowledge can be avoided by projecting models from existing source code automatically.

This later assumption is in accordance with our finding regarding the effort to correctly comprehend the configuration knowledge. In fact, we observed that GenArch$^+$ does not require less time to comprehend a product line, than CIDE and pure::variants. Nevertheless, the results clearly illustrate domain-knowledge models usefulness for correct configuration knowledge comprehension. We also observed that our proposed product line engineering technique is also useful for the case of traditional configuration knowledge comprehension tasks, such as "Identifying all files in which source code of a feature occurs" and "Identifying all features that occurs in a certain file". In the case of specific tasks, associated to comprehend the configuration knowledge related to framework-provided concepts, GenArch$^+$ out-performed the traditional code-oriented techniques.

The results presented in this thesis may impact the way product lines are engineered and documented. Taking a language-oriented perspective brings many benefits, such as improved visualization of features and automation. The need of DKMLs arises due to the inability of the current code-oriented techniques to adequately express framework-provided concepts and programming

interface defined over programming languages. We believe that in the future programming languages and programming interface construction mechanisms will evolve so that concepts instances will be easily identified in the source code and instantiation constraints will be checked by the compiler. However, until that happens DKMLs can provide the support required to properly comprehend and safely built framework-based software product lines.

6.1 Limitations and Future Work

There are some possible directions for future works. One important attempt is the formalization of the DKML concept and a deeper investigation of its desired properties. This formalization would allow us to describe all elements of DKMLs and the generic infrastructure provided by GenArch$^+$ more precisely. In particular, three new areas would benefit from formalization: integrating DKMLs and programming language semantics, product line refactoring, and dynamic analysis of code customization.

In future work, we want to develop an environment that better integrates and unifies the development mechanisms, low level (programming languages) and high level (DKMLs). First it will provide a more precise verification of the configuration knowledge, that is, verifications that do not only take into account the knowledge expressed by the domain knowledge models. This formalization will also support small-step refactorings that preserve the behavior of all products regarding the semantics defined by the framework's programming interfaces. Finally, the impact of assigning features to code customization may also be completely checked against violations of the feature model semantics considering a more appropriated dynamic analysis of concept instance usage.

Application of DMKLs in practice is needed to ultimately confirm the usefulness and value of the proposed engineering. To that end, more languages should be built to address current problems with a few widely used frameworks, and the effectives of DKMLs should be evaluated in large scale software product lines. Moreover, in this case, we believe that future work on views and feature visualizations is need to improve the comprehension of large scale configuration knowledge. A new research endeavor in this direction would be to empirically assess the impact of more recent domain-specific based techniques, such as Clafer (Bak 2010) and Projectional Workbenches (Völter and Visser 2011), on product line comprehension.