Combining Context-Oriented Programming and Feature Diagrams: Towards a Position Statement

Arnaud Hubaux, Andreas Classen*, Patrick Heymans
PReCISE Research Centre,
Faculty of Computer Science,
University of Namur
5000 Namur, Belgium

E-mail: {ahu,acs,phe}@info.fundp.ac.be

1 Introduction

Context-Oriented Programming (COP) is an emerging programming paradigm which aims to alleviate the management of behavioural variations resulting from context changes [6]. The basic idea is to allow the developer to focus on the business code without having to hard-code alternatives to variations in the context. A context-oriented program is defined as a set of layers dynamically activated and deactivated at runtime according to the evolution of the context.

An issue barely tackled in COP is the modelling, at a high level, of the context layers and the conditions of their activation. Two sorts of constraints on layer activation and deactivation require special attention: (1) the static constraints independent of the context status, e.g. exclusivity between the UDP and TCP protocols, and (2) the dynamic constraints evaluated at runtime, e.g. the loss of the wifi connection automatically disables the voice over IP feature.

A rather straightforward way to express the constraints of type (1) is to model each layer as a feature. The resulting feature diagram (FD) [7] represents the hierarchy of layers along with their cross-layer constraints. The constraints of type (2), however, require further investigations because they involve dynamic configuration changes and can vary from one product to another. Some approaches have already been proposed to deal with these two issues.

Fernandes *et al.* [4] present UbiFEX, a modelling notation extending existing FD languages with contextual information. The general FD generated with UbiFEX is composed of an FD, a context FD with the associated activation expressions, and context rules binding the context and feature models together. UbiFEX also comes with a simulation tool checking the consistency of the produced models.

Desmet *et al.* [3], propose the Context-Oriented Domain Analysis (CODA) which is heavily inspired by the original Feature-Oriented Domain Analysis (FODA) [7] used in product-line development. It enforces software engineers to think of context-aware systems as pieces of basic context-unaware behaviour which can be further refined by means of context-dependent adaptations at certain variation points. A context-dependent adaptation is a unit of behaviour that adapts a subpart of a software system only if a certain context condition is satisfied.

Hartmann *et al.* [5] introduce context variability models (CVM) to represent high-level context information of software supply chains. A CVM is a general classifier of the context of products that is expressed in a FODA like notation. The combination of the CVM and the SPL feature model results in a Multiple Product Line Feature Model where the dependencies between both models are expressed with `requires` and `excludes` links. They do not explicitly present their work as suited to self-adaptive or dynamic systems. Once adopted, the context configuration choices are immutable and do not lead to self-adaptive behaviours. Conversely, Desmet *et al.* [3] and Fernandes *et al.* [4] consider the dynamic evolution of the context and its impact on the model.

In this paper, we identify fundamental challenges existing at the modelling side of COP. Specifically, we focus our attention on the use of FDs to represent context layers and their extension to cope with context-dependent configuration changes.

*FNRS Research Fellow.

2 Position statement

It is acknowledged that FDs are best suited to represent the variability of software product lines prior to runtime. Nevertheless, as observed in [2], it is common that the configuration of an SPL product changes at runtime. Although being a valuable solution to deal with configuration change at runtime, COP is restricted to code-level adaptations. The link between the static representation of variability and its dynamic management at the realisation level is thus missing.

The core challenges one must address to fill this gap are the following.

- *Define the type of context the model should deal with.* The context of an SPL product can be characterised as static or dynamic. The static context is the environmental information to take into consideration before running the program. The dynamic context is the environmental information entailing dynamic changes of the running system.

- *Find the most appropriate way to express context depending on its type.* Due to their very different natures, the model chosen to represent them is very likely to differ significantly. The type of context addressed will thus have to be precisely specified.

- *Define ways to scope the context to model.* Although a traditional activity of software engineering, explicit guidelines have to be defined as extensions of existing work in this area.

- *Extend the semantics of FD with contextual models.* The well known static semantics of FD [8] will have to be extended with the language used or designed to account for the context. Future work will tell whether the dynamic semantics of FD [1] should be used a basis for such extended model.

- *Map the features in the FD to the layers in COP.* At the implementation level, each feature from the FD will have to be mapped to context layers.

- *Map context constraints to layer activation/deactivation in COP.* As for features, all the elicited context constraints will have to be mapped to the running code.

- *Diagnose layer activation/deactivation.* Besides enhancing the understandability of large problems, the purpose of the model is to ease the programming task and to widen automated reasoning opportunities at runtime. Part of this reasoning will be dedicated to the monitoring of the activation and deactivation of layers, and the cancellation of the action if necessary.

- *Provide runtime diagnosis information.* Ideally, not only the achievability of the activation/deactivation should be reported to programmer, but also its proof. This way, the proper reaction to the illegal action is made possible. Note also that solution proposals could come along with the proof to facilitate problem resolution.

3 Conclusion

In this paper, we have listed several challenges one faces when linking context-oriented programming with a feature diagram used in software product line engineering. The next step of our research is to systematically address each of these points to ultimately come up with a comprehensive solution proposal.

Acknowledgements

This work is sponsored by the Interuniversity Attraction Poles Programme of the Belgian State of Belgian Science Policy under the MoVES project and the FNRS.

References

[1] A. Classen, A. Hubaux, and P. Heymans. A formal semantics for multi-level staged configuration. In *Proceedings of the Third International Workshop on Variability Modelling of Software-intensive Systems (VaMoS'09)*, Sevilla, Spain, January 2009. University of Duisburg-Essen.

[2] A. Classen, A. Hubaux, F. Sanen, E. Truyen, J. Vallejos, P. Costanza, W. De Meuter, P. Heymans, and W. Joosen. Modelling Variability in Self-Adaptive Systems: Towards a Research Agenda. In *1st International Workshop on Modularization, Composition, and Generative Techniques for Product Line Engineering (McGPLE08)*, Nashville, USA, October 2008.

[3] B. Desmet, J. Vallejos, P. Costanza, W. De Meuter, and T. D'Hondt. *Modeling and Using Context*, chapter Context-Oriented Domain Analysis, pages 178–191. Springer Berlin / Heidelberg, 2007.

[4] P. Fernandes and C. Werner. Ubifex: Modeling context-aware software product lines. In *2nd International Workshop on Dynamic Software Product Lines (DSPL08)*, Limerick, Ireland, 2008.

[5] H. Hartmann and T. Trew. Using feature diagrams with context variability to model multiple product lines for software supply chains. In *12th International Software Product Line Conference*. IEEE Computer Society, 2008.

[6] R. Hirschfeld, P. Costanza, and O. Nierstrasz. Context-oriented programming. *Journal of Object Technology (JOT)*, 7:125–151, 2008.

[7] 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.

[8] P.-Y. Schobbens, P. Heymans, J.-C. Trigaux, and Y. 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.