# Should Future Variability Modeling Languages Express Constraints in OCL?

Don Batory
University of Texas at Austin
batory@cs.utexas.edu

## ABSTRACT

Since the mid-2000s, *Propositional Logic* (PL) has been the de facto language to express constraints in *Feature Models* (FMs) of *Software Product Line* (SPLs). PL was adequate because product configurations were formed by binary decisions including or not including features in a product. Inspired by both prior research and practical systems (*eg.*, SPLs that use KConfig), future FMs must go beyond PL and admit numerical (and maybe even text) variables and their constraints.

The *Object Constraint Language* (OCL) is a general-purpose declarative constraint language for *Model Driven Engineering* (MDE), which admits virtually any kind of variable and constraint in metamodels. We should expect future FMs to be examples of MDE metamodels. This raises a basic question: Should OCL be used to express constraints of future variability modeling language(s)?

In this talk, I outline the pros and cons for doing so.

**ACM Reference Format:**
Don Batory. 2019. Should Future Variability Modeling Languages Express Constraints in OCL?. In *23rd International Systems and Software Product Line Conference - Volume B (SPLC '19), September 9–13, 2019, Paris, France.* ACM, New York, NY, USA, 1 page. https://doi.org/10.1145/3307630.3342406

## 1 INTRODUCTION

I am no fan of the *Object Constraint Language* (OCL) and never have been. I find it inelegant and bloated. Using Eclipse OCL years ago, I recall different OCL implementations did not agree on syntax and covered the OCL standard (at that time) with varying fidelity. (Note: The current 2.0 standard is a behemoth 262 page document [5]!) I tried to teach OCL to my undergrads, and that was an unpleasant experience for both me and my students. OCL and related tools were simply too complicated. As mentioned in [1], Eclipse tools:

(1) Were unappealing—they were difficult to use even for simple applications.
(2) Fostered a medieval mentality in students to use incantations to solve problems. Point here, click that, something happens. From a student's perspective, this is gibberish. Although I could tell them what was happening, this mode of interaction left a vacuum where a deep understanding should reside.
(3) Have a steep entry cost to use, teach, and learn – too high for my comfort (and I suspect my student's as well).

I'm not alone with these opinions [2].

From a distance, I have also watched various attempts to generalize *Feature Models* (FMs) to address next generation *Software Product Line* (SPL) concerns – such as admitting replicated features, features with attributes, numerical features, and expressing constraints. I recoiled at the complexity of these attempts, and the use of OCL as the language to express constraints.

I do not profess to know what future SPL Variability Modeling Languages will be and how constraints in such languages will be expressed. But I do believe the answer will be guided by:

- **Simplicity!** PL was chosen for classical FM constraints because it was a simple mathematical standard. I'm not sure there is a formal grammar (language) to which all classical FM tools agree, but it is hard to screw-up writing PL constraints. Next-generation FM constraints should be equally straightforward to write.
- **Don't Invent, Reuse!** Do we really need a **new** constraint language for future FMs? Clearly we need more than PL. But are we good enough as language engineers to create a new constraint language without making a complete mess of it? Shouldn't we reuse existing languages or sub-languages of existing languages? Our expertise is in SPLs, **not** in language engineering. If you want an example of (IMO) a failed custom constraint language, it is OCL. Re-read the 2nd sentence of this Introduction.
- **Circularity Avoidance!** Generalizing beyond hierarchical relationships of classical FMs, we're not far away from UML class diagrams [4] and Model Driven Engineering *metamodels* [3] – which are class diagrams + constraints. And constraints for class diagrams beg the use of OCL.

In this talk, I offer and demonstrate a way out of this circular conundrum. My solution does *not* eliminate all problems, but it does diminish key problems about OCL standards, OCL tooling, reducing the need for yet-another-language, minimizing long-term tool maintenance, and keeping constraint languages both familiar and simple to SPL programmers, practitioners, and researchers.

**Acknowledgments.** Batory is supported by NSF grant CCF1212683.

## REFERENCES

[1] D. Batory and M. Azanza. Teaching model-driven engineering from a relational database perspective. *Softw. Syst. Model.*, May 2017.
[2] J. Cabot and M. Gogolla. Object constraint language: A definitive guide. https://www.slideshare.net/jcabot/ocl-tutorial.
[3] K. Czarnecki, Chang Hwan, P. Kim, and K. T. Kalleberg. Feature models are views on ontologies. In *SPLC*, 2006.
[4] M. Fowler and K. Scott. *UML Distilled: Applying the Standard Object Modeling Language.* Addison-Wesley Longman Ltd., 1997.
[5] About the object constraint language specification version 2.4. https://www.omg.org/spec/OCL/About-OCL/, 2019.


*SPLC '19, September 9–13, 2019, Paris, France*

ACM ISBN 978-1-4503-6668-7/19/09…$15.00
https://doi.org/10.1145/3307630.3342406