Poster: On Extracting Relevant and Complex Variability Information from Software Descriptions with Pattern Structures

Jessie Carbonnel, Marianne Huchard, Clémentine Nebut
LIRMM, Université de Montpellier, CNRS, Montpellier, France
{jcarbonnel,huchard,nebut}@lirmm.fr

ABSTRACT

The migration from existing software variants to a software product line is an arduous task that necessitates to synthesise a variability model based on already developed softwares. Nowadays, the increasing complexity of software product lines compels practitioners to design more complex variability models that represent other information than binary features, e.g., multi-valued attributes. Assisting the extraction of complex variability models from variant descriptions is a key task to help the migration towards complex software product lines. In this paper, we address the problem of extracting complex variability information from software descriptions, as a part of the process of complex variability model synthesis. We propose an approach based on Pattern Structures to extract variability information, in the form of logical relationships involving both binary features and multi-valued attributes.

CCS CONCEPTS

• **Information systems → Information extraction**; • **Software and its engineering → Software product lines**; **Software reverse engineering**;

KEYWORDS

Software Product Line, Reverse Engineering, Variability Extraction

ACM Reference format:
Jessie Carbonnel, Marianne Huchard, Clémentine Nebut. 2018. Poster: On Extracting Relevant and Complex Variability Information from Software Descriptions with Pattern Structures. In *Proceedings of 40th International Conference on Software Engineering Companion, Gothenburg, Sweden, May 27-June 3, 2018 (ICSE '18 Companion),* 2 pages.
https://doi.org/10.1145/3183440.3194982

1 INTRODUCTION

Software Product Line (SPL) Engineering [4] is a development paradigm that seeks to develop a set of similar software systems through systematic artefact reuse and mass-customisation. The core of this approach is based on the development of a generic software architecture, on which variable and reusable software artefacts can be plugged depending on given requirements. The key of SPL implementation is variability modelling, i.e., representing what can vary in the software systems to be developed, and how it can vary. Variability is usually expressed in terms of binary features, which are distinguishable variant characteristics. A recent survey shows that a significant part of companies opt for an extractive adoption strategy of SPL [1]; it is an arduous task that implies to extract variability information based on descriptions of the existing software systems, in order to build a variability model [3].

Pattern Structures are a mathematical framework for hierarchical clustering of a set of objects described by complex data called *patterns*, allowing to extract all logical relationships that are true for the considered set of objects. In this paper, we propose a method that uses Pattern Structures to extract logical relationships involving both features and multi-valued attributes from complex variability descriptions, i.e., depicting not only binary features. It is a first step towards the automated synthesis of complex variability models from existing software systems. The aim of this research is to ease the transition of more and more complex software variants that have been individually developed, to software reuse and mass-customisation approaches. Our method is sound and complete, and offers theoretical foundations to complex variability information extraction.

2 METHOD OVERVIEW

The proposed extraction method is based on Pattern Structures (PS) [2], a mathematical framework used for knowledge discovery on a set of objects described by complex data, e.g., numerical values, graphs, sets of values. PS describe each object of a set O according to a *pattern*, that can be of any type of data on which one can define a *similarity operator* $\sqcap$. Applied to a set of patterns, this operator returns a pattern representing their most specific common generalisation. We applied our method on existing multi-valued matrices taken from the software comparison category of wikipedia. Here, the objects represent the software variants, and the patterns represent the variant descriptions. Table 1 presents an excerpt of a multi-valued matrix depicting softwares and their characteristics. It describes 5 software variants depending on 5 characteristics: *Open Source*, *Corporate* and *Personal* are binary characteristics, *First Release* is a characteristic having 4 possible Date-typed values, and *Language* is a characteristic with 3 possible String-typed values. Our method can be decomposed in the 4 steps of Figure 1.

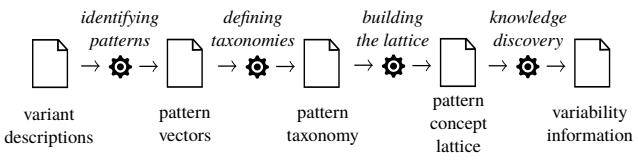

Figure 1: Extracting complex variability information from variant descriptions using Pattern Structures.

ICSE '18 Companion, May 27-June 3, 2018, Gothenburg, Sweden

ACM ISBN 978-1-4503-5663-3/18/05.
https://doi.org/10.1145/3183440.3194982

Table 1: Excerpt of a multi-valued matrix representing software variant descriptions taken from wikipedia

		Open Source	Corporate	Personal	First release	Language
1	Confluence		X	X	2004	Java EE
2	GrokOla		X		2014	JavaScript
3	Jive		X		2006	Java EE
4	Wagn	X	X	X	2006	Ruby
5	Wiki.js	X	X	X	2017	JavaScript

Identifying patterns. Patterns can be values of *atomic types*, e.g., a Date, a String or a number. But, one can choose to compose several patterns to form a *vector of patterns*, that can be considered a pattern as well. In our case, a variant description may be represented as a pattern vector depicting the values of each characteristic of the matrix. This first step consists in identifying in the variant descriptions the set of component patterns that will form the pattern vectors. As binary characteristics have a $\{true, false\}$ domain, the set of binary characteristics having a $true$ value of a variant can be considered as one multi-valued characteristic. This representation choice does not alter the extracted information and permits to simplify the processed data.

The pattern vectors for the software variants of Table 1 are thus of the form $v = \langle P_f, P_{fr}, P_l \rangle$, where P_f is the set of features (i.e., binary characteristics), followed by the value of the attribute *First Release* (*FR*) (P_{fr}) and then the value of the attribute *Language* (P_l). The two following pattern vectors respectively represent the softwares *Confluence* and *GrokOla*:

$$v_1 = \langle \{Corporate, Personal\},\ FR = 2004,\ Language = Java\ EE \rangle$$
$$v_2 = \langle \{Corporate\},\ FR = 2014,\ Language = JavaScript \rangle$$

Defining taxonomies. Now that we have specified the pattern vectors, we have to define similarities between the elements of each component pattern. A similarity operator is associated to a subsumption relation $\sqsubseteq$, allowing the set of patterns P to be partially ordered in a taxonomy, in which each pair of elements has a unique most specific common generalisation pattern. In other words, it structures the patterns by specialisation, and $p_1 \sqsubseteq p_2 \iff p_1 \sqcap p_2 = p_1, \forall p_1, p_2 \in P$. For example, let us consider P_f the first component of our vector, representing the binary features owned by the variants. The similarity between a variant described by the feature set $\{Corporate, Personal\}$ and another variant described by $\{Corporate\}$ is that they are both described by the feature set $\{Corporate\}$ (denoted $\{Corporate, Personal\} \sqcap_f \{Corporate\} = \{Corporate\}$). Also, a way to define the similarity of a variant first released in 2004 and a variant first released in 2014 is that they have both been released between 2004 and 2014 ($2004 \sqcap_{fr} 2014 = [2004, 2014]$). Once we have defined the similarity operators of the component pattern characterising our software variants, we can compute similarity between these pattern vectors:

$$v_1 \sqcap v_2 = \langle \{Corporate\}, FR = [2004, 2014], Language = \neg Ruby \rangle$$

Building the lattice. The set of objects O, the pattern taxonomy $(P, \sqcap)$ and the mapping $\delta : O \to P$ associating each object of O to its pattern in P, form a Pattern Structure. Given a Pattern Structure $PS = (O, (P, \sqcap), \delta)$, a *pattern concept* of PS is a tuple (O', p) such that $O' \subseteq O$ and $p \in (P, \sqcap)$, which represents a maximal set of objects and the most specific pattern corresponding to these objects. This means that there is no other object than the ones in O' that corresponds to the pattern p, and that there is no other pattern more specific than p which corresponds to all the objects from O'. These pattern concepts can be partially ordered: given two pattern concepts $C_1 = (O_1, p_1)$ and $C_2 = (O_2, p_2)$ of PS, $C_1 \leq_{ps} C_2$ if and only if $O_1 \subseteq O_2$ and $p_1 \sqsubseteq p_2$. The set of all pattern concepts of PS provided with the order $\leq_{ps}$ forms a lattice structure called a *pattern concept lattice.*

Knowledge discovery. The pattern concept lattice emphasises several types of variability information, i.e., logical relationships between features and/or attribute values, that state how these elements can be combined to compose a software of the software family. The most represented and studied variability relationship types in the literature are implications between elements, co-occurring elements, mutually exclusive elements and feature-groups. All these types of variability information can be extracted from PS using the proper algorithms [5]. By theoretical definition, variability extraction with Pattern Structures is complete, in the sense where they allow to extract all logical relationships (among the 4 aforementioned types) that are true for the considered set of variants.

3 CONCLUSION AND PERSPECTIVES

We presented an approach to extract complex variability information from software variant descriptions based on Pattern Structures. It allowed us to extract variability information by considering not only the software variants' features, but also multi-valued attributes. This work is a first step toward a more generic approach to assist practitioners into extracting complex variability models.

Our extraction method is complete and therefore extracts an important number of relationships. It appears that a significant part of them are "accidental", i.e., true for the considered set of variants but not regarding the domain. In future work, we plan to study existing techniques to lower the number of considered extracted relationships by deepening the separation of the meaningful ones from the accidental ones. Another future work will be to consider the extraction of other kinds of variability information that could be useful to synthesise complex variability models from software descriptions. Particularly, relationships between several independent but connected software families are to be studied, allowing applications in the field of multiple software product lines.

REFERENCES

[1] Thorsten Berger, Ralf Rublack, Divya Nair, Joanne M. Atlee, Martin Becker, Krzysztof Czarnecki, and Andrzej Wasowski. 2013. A survey of variability modeling in industrial practice. In *Proc. of the 7th Int. Workshop on Variability Modelling of Software-intensive Systems.* 7:1–7:8.

[2] Bernhard Ganter and Sergei O. Kuznetsov. 2001. Pattern Structures and Their Projections. In *Proc. of the 9th Int. Conf. on Conceptual Structures.* 129–142.

[3] Charles W. Krueger. 2001. Easing the Transition to Software Mass Customization. In *Proc. of the 4th Int. Workshop on Software Product-Family Engineering.* 282–293.

[4] Klaus Pohl, Günter Böckle, and Frank J van der Linden. 2005. *Software Product Line Engineering: Foundations, Principles, and Techniques.* Springer Science & Business Media.

[5] Uwe Ryssel, Joern Ploennigs, and Klaus Kabitzsch. 2011. Extraction of feature models from formal contexts. In *Workshop Proc. (Vol. 2) of the 15th Int. Conf. on Software Product Lines.* 4:1–4:8.