Variability Extraction from Simulator I/O Data Schemata in Agriculture Decision-Support Software

Thomas Georges[1,2], Marianne Huchard[1], Mélanie König[2], Clémentine Nebut[1], and Chouki Tibermacine[1]

[1] LIRMM, Univ Montpellier, CNRS, Montpellier, France
{firstname.lastname}@lirmm.fr
[2] ITK, Montpellier, France
{firstname.lastname}@itk.fr

Abstract. The context of this work is the development of software systems that help in decision making in the agriculture domain at our industrial partner, ITK. These software systems include simulators which help farmers to understand and predict plants life cycle. Each plant and each kind of prediction has its own parameters. For example, yield prediction for wheat is very specific and different from vine disease prediction. There are however some common characteristics, like the fact that these simulators take as input weather data. The goal of the project on which we work is to build a software product-line in order to: i) enable an easy derivation of new products (by IT teams) with new simulators (built by agronomist teams), and ii) simplify the maintenance of the existing large code base of our industrial partner. The construction of this product-line passes through the extraction of variable and common characteristics of all existing products at ITK. The extraction process may be laborious and time consuming. We study in this work the automation of this process, by focusing on the schemata of data received as input and produced as output by simulators. We hypothesize that Formal Concept Analysis (FCA) is a useful tool for extracting software variability, i.e. highlight commonalities and specifics for assisting IT/agronomist teams in software construction. In this paper, we propose a process for variability extraction. This process is based on a set of pre-processing steps to prepare data for FCA tools. These tools build at the end of the process an AOC-Poset, i.e. a conceptual structure derived from the concept lattice in which we can identify common and variable characteristics. We implemented this process and experimented it on a set of six simulators. We obtained promising results towards the construction of the software product-line.

Keywords: Formal Concept Analysis · Software engineering · Software Product Line · Variability Extraction · Knowledge Extraction

1 Introduction

Many software companies face the problem of developing and maintaining a portfolio of products with some common purpose and context. Capitalizing knowledge acquired on the domain and on the previously developed software may be a help for developing new ones in the same business domain, and rationalizing the different activities around software. When software systems are sufficiently similar, migrating to the software product line paradigm may be appropriate for that capitalization. For example, our industrial partner ITK[3] provides a decision-support software systems platform for agriculture. It develops simulators for different purposes as yield expectation or disease prediction. The platform can be seen as a set of similar software systems, which allows a migration to a software product line. The expected benefit is to assist the agronomist team in the development of a new product, and speed up simulator integration by the IT team. This requires extracting and organizing knowledge from the code base and all existing documents. As a first step in that direction, this paper focuses on extracting knowledge from part of this description, which is composed of an input and an output data schema, with a tree structure embedding the data hierarchical organisation. We use Formal Concept Analysis to highlight input and output variability among the different simulators. We call variability the ability of a software to be configured, customized, extended, or changed for a specific context [3]. In our case, it concerns the variation in the data schemata of simulators. In this paper, we explain the process that leads us from raw data to a conceptual structure, here an AOC-Poset and how to exploit it. Concretely, the contribution of this paper is an application of Formal Concept Analysis to identify variability. This paper is organized as follows. Section 2 gives an overview of the approach. Section 3 addresses the dataset presentation. Section 4 explains how the preprocessing is performed on Data schemata. Section 5 presents the Formal Concept Analysis processing. Section 6 shows the results. Section 7 exposes the related work and Section 8 concludes the paper with a summary of the contribution and a few perspectives.

2 Overview of the approach

Research Question. The main question studied in this paper is: *How to extract software variability from simulator data schemata?* To answer this question, we need to use simulator data schemata to build a formal context usable with FCA, and more specifically AOC-Poset building algorithms. This is performed by a process, presented in the following subsection, which is able to exploit simulator data schemata to get as much knowledge as possible.

Process. Figure 1 illustrates the process to get variability from raw data schemata. First a pre-processing is performed, by cleaning, enriching and formatting data to obtain a formal context. Then from the formal context, an FCA-based structure,

[3] https://www.itk.fr/en/

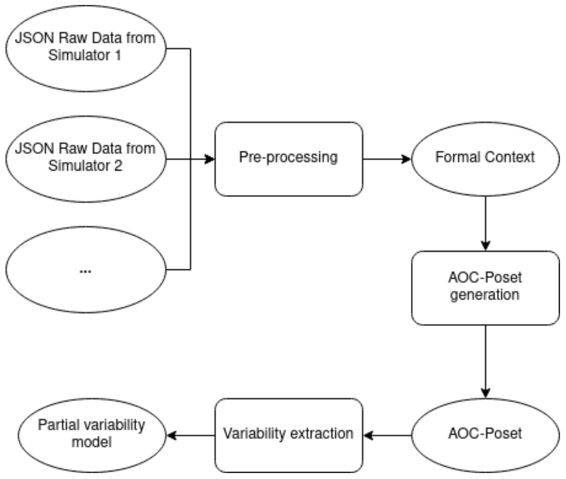

JSON Raw Data from Simulator 1
JSON Raw Data from Simulator 2
...
Pre-processing
Formal Context
AOC-Poset generation
AOC-Poset
Variability extraction
Partial variability model

Fig. 1. Process from simulators to variability

i.e the AOC-Poset [10] is used to highlight commonalities and specifics. So far, only a partial variability model is created, with common terms and simulator-specific terms. We have not yet studied the identification of logical rules relating these terms, like implications, co-occurrences or mutual exclusions between two terms, but we are quite confident that this is possible by applying techniques proposed in previous work from our team [8].

An AOC-Poset has been chosen instead of a usual concept lattice because of the simplicity of this model which makes it easily readable and understandable in the context of variability analysis. The second theoretical reason is the complexity of its construction, which is polynomial in contrast to the construction of concept lattices that is exponential. The size of simulator data schemata is relatively small for the moment in our study, but in the future if the process is to be used with larger and numerous schemata, the construction of the variability model would be made more efficient.

3 Simulator description

Each simulator used in ITK products has its own purpose. It receives some data as input, like weather information or soil type. It produces some data as output, like predictions of yield or disease. All ITK products are defined as Web applications with simulators written mainly in Python. Input and output data are defined in *JSON* format[4] and have schemata defined in *JSON* too.

Output data schemata are in general less complex than input data ones and can be absent in some cases, if there is only a number or a string as the output from a simulator.

[4] https://www.json.org/json-en.html

Our study is based on six different simulators:

- *Cropwin simulator* to estimate yield from annual culture as wheat or corn.
- *Disease simulator* to predict plant's risk to contract a disease.
- *Grapes simulator* to estimate yield from grapes cultures.
- *nferti simulator* to predict plant's stress, as the lack of nitrogen.
- *Orchard simulator* to estimate yield from sustainable culture as apricots or walnuts.
- *Vine disease simulator* to predict vine's risk to contract a specific disease.

Simulator description Each simulator has a documentation which is provided by the agronomists who developed the simulator. This documentation includes a wiki with a description of the simulation model (the mathematical model), the API for using the simulator, its dependencies and its technical documentation, among other elements. The simulation model provides the schema of the data needed for running the simulator (input data) and also the schema of the data provided as an output. These schemata are defined using a JSON dialect. We collected the available schemata for the selected six simulators.

Input/output data schema. Each data schema is a tree of terms. From these six simulators, we have six input data schemata and four output ones. Input data schemata include from 31 to 127 different terms and output data schemata include 22 to 95 different terms (see Table 1 for details).

Simulator	nferti	Cropwin	Disease	Grapes	Orchard	Vine disease	**total**
Inputs terms	119	126	127	31	50	11	**464**
Outputs terms	95	32			53	22	**202**

Table 1. Simulator input and output description size

4 Pre-processing Data schemata

Raw data schemata cannot be exploited directly in our process. They need to be sanitized, formatted and prepared to be exploited by FCA. For this, we build a dictionary to exclude unwanted/technical terms, a dictionary to associate new terms to replace existing acronyms and abbreviations. In order to maximize variability extraction, we choose to exploit tuples of terms. The results have been formatted as a formal context.

The construction of our dictionaries by removing or associating more terms was an iterative manual work. An analysis was made after each iteration. Redundant and inappropriate terms were removed and new terms explaining existing abbreviations were added.

In Figure 2, we outline the complete process that goes from raw data to a formal context. First we transform raw data schemata in tree (*Treeify*) in order

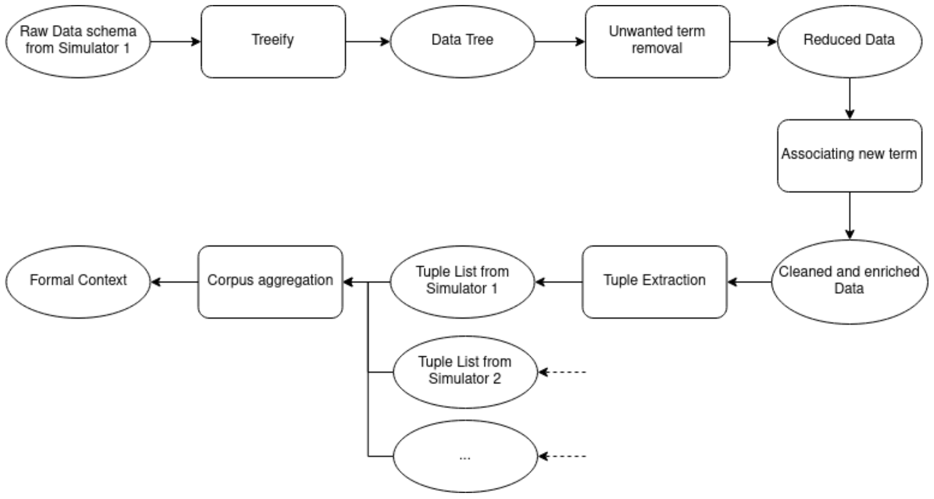


Fig. 2. Process to transform raw data into formal contexts.

to make the following processing steps on a tree and not on a text document. Then, we remove useless terms (*Unwanted term removal*). Next we replace abbreviations and acronyms (*Associating new term*) and we extract tuples from the tree (*Tuple Extraction*). At the end, for all simulators, data schemata of each kind (Input or Output) are merged and formatted as a formal context (*Corpus aggregation*).

4.1 Excluding unwanted terms

Stopwords are useless terms, which do not need to be kept in our variability extraction result. We choose to remove them to limit unnecessary too frequent terms. Without removing unwanted terms, variability extraction would be less relevant due to the introduced noise. To build the dictionary, we ranked all terms by their frequency, in order to select and remove all too frequent unwanted terms. We have used a well-known metric for that which is TF-IDF [16]. We built a dictionary with 30 different terms to be removed.

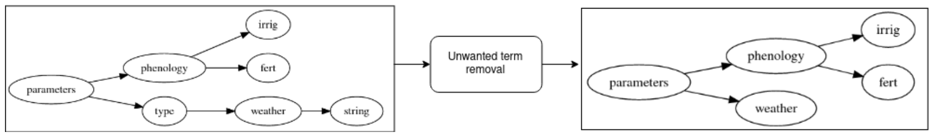


Fig. 3. Removing unwanted terms example

Figure 3 depicts an example of this processing. Each term in a data schema is searched in the unwanted terms dictionary and removed from the tree if present (e.g. *type* and *string*). If the removed term is not a tree leaf, the subtree linked to this removed node is linked to its parent directly.

4.2 Associating new terms

The goal of building this second dictionary is to extend abbreviations and replace acronyms by the complete terms. The construction of this dictionary has been done manually. We checked each term to decide if it was an acronym or an abbreviation. If this was the case, we added it to the dictionary together with its complete name. For example, *irrig* has been added and associated to the term *irrigation*. The built dictionary includes around 30 different terms, 9/10 of them coming from the agronomic domain and 1/10 from IT domain.

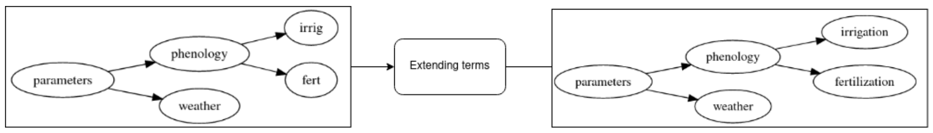


Fig. 4. Extending terms example

Figure 4 shows an example of this processing. Each term is compared with the associated term in the dictionary. If it matches, the current term is replaced by its complete version.

4.3 Tuple extraction

Extracting only single terms makes us loose the information about relationships provided by the tree structure. To keep information from data we need to refine the extraction. We extract each node alone, but also all father-son's pairs following a depth-first search method. The size of the extracted tuples is one or two terms, and this is enough for our process. After empirical observations, we indeed concluded that the use of tuples of size three or more terms does not help in identifying more commonalities in our data, because they are present in only one simulator. For example, the following tuple `(parameters,phenology,irrigation)` is specific to a single simulator, which is *nferti*.

Besides this, we did not base our process on raw text data, and choose to transform it into our own tree representation, in order to be independent from any kind of data structure format, such as *XML* or *JSON* in our case.

Figure 5 depicts an example of tuple extraction. In this example, we can observe the extraction of five tuples with a single term and four tuples with two terms, starting from a tree of five nodes and four father-son edges.

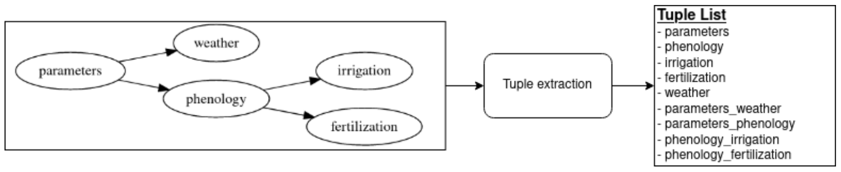


Fig. 5. From cleaned and enriched data to formal attributes

4.4 Data Formatting

To use Formal Concept Analysis, a last transformation is required. We use here FCA [12] as a knowledge engineering method, for its capacity to build *formal concepts* from a formal context (FC) $K = (G, M, I)$ that associates objects from a set G to attributes from a set M through relation $I \subseteq G \times M$. Object sets and attribute sets are associated thanks to two operators, both denoted by $'$. For $O \subseteq G$, the set of attributes shared by the objects of O is $O' = \{m | \forall g \in O, (g, m) \in I\}$. For $A \subseteq M$, the set of objects that share the attributes of A is $A' = \{g | \forall m \in A, (g, m) \in I\}$. A formal concept $\mathcal{C} = (Extent(\mathcal{C}), Intent(\mathcal{C}))$ is a maximal object group (extent) associated with their maximal shared attribute group (intent), i.e. $Extent(\mathcal{C}) = Intent(\mathcal{C})'$ (and equivalently $Extent(\mathcal{C})' = Intent(\mathcal{C})$). The concept order, denoted by $\preceq_{\mathcal{C}}$ is defined as follows: $\mathcal{C}_1 \preceq_{\mathcal{C}} \mathcal{C}_2$ if $Intent(\mathcal{C}_2) \subseteq Intent(\mathcal{C}_1)$ (and equivalently $Extent(\mathcal{C}_1) \subseteq Extent(\mathcal{C}_2)$). The concept lattice is the set of all concepts, provided with $\preceq_{\mathcal{C}}$. The lowest (w.r.t. $\preceq_{\mathcal{C}}$) concept owning one object is its introducer concept. The highest (w.r.t. $\preceq_{\mathcal{C}}$) concept owning one attribute is its introducer concept. The suborder of the concept lattice restricted to these introducer concepts is called the AOC-Poset (Attribute-Object Concept poset). In the following, we use the AOC-Posets, which are a scalable alternative to concept lattices, as the conceptual structures to highlight variability, as they contain all the information we need.

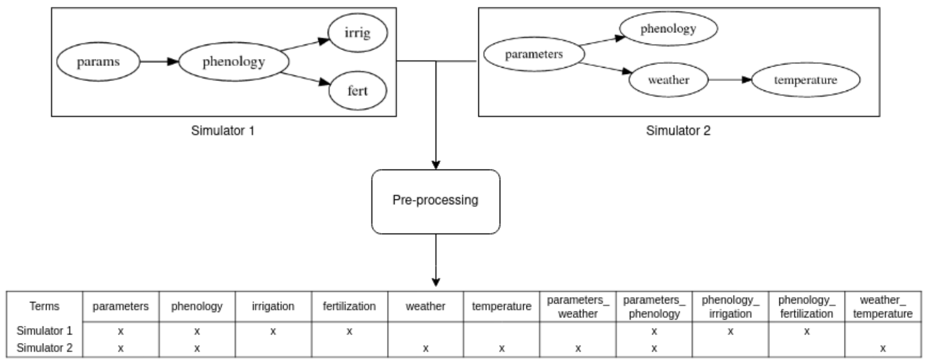
We need to generate two formal contexts from the extracted tuples. One for the input and the other for the output data schemata. In each formal context, G is the set of simulators, M is the set of tuples, i.e. 1-tuples for nodes or 2-tuples for edges. $(g, m) \in I$ if the tuple m exists in the data schema of the simulator g.

Figure 6 shows an example of the transformation. It starts from two simulators and generates a formal context. A cross means that the simulator has the tuple.

5 FCA processing

The cleaned data is now in the appropriate format to be processed by FCA tools. Extracting the variability using FCA is efficient and the obtained conceptual structures are a useful way to express this variability [6]. We used the RCA plugin of Cogui[5] to generate AOC-Posets [10]. For the input data schemata,

[5] http://www.lirmm.fr/cogui/



Terms	parameters	phenology	irrigation	fertilization	weather	temperature	parameters_ weather	parameters_ phenology	phenology_ irrigation	phenology_ fertilization	weather_ temperature
Simulator 1	x	x	x	x				x	x	x	
Simulator 2	x	x			x	x	x	x			x

Fig. 6. Example of raw data transformed into a formal context

we obtained the AOC-Poset depicted in Figure 7. This conceptual structure is discussed in the following section.

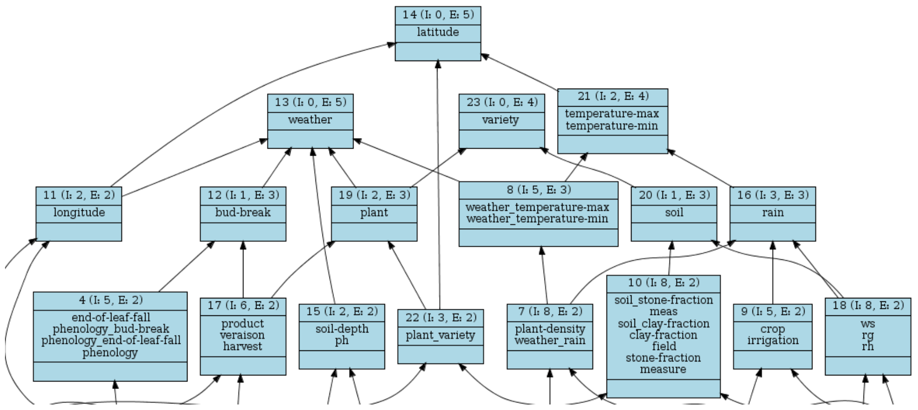


Fig. 7. Excerpt of the AOC-Poset from input data schemata

6 Evaluation

The variability extracted from AOC-Posets is used to understand how close the simulators are to each other.

6.1 Results

The excerpt of the AOC-Poset, presented in Figure 7, shows which simulators parts are specific and which are common. Precisely, beginning from the most common parts (from the top concepts of the AOC-Poset), we have:

– The concept 14 introducing attribute *latitude*, common to five simulators out of six.
– The concept 13 introducing attribute *weather*, common to five simulators out of six.
– The concept 23 introducing attribute *variety*, common to four simulators out of six.
– The concept 21 introducing attributes *temperature-max* and *temperature-min*, common to four simulators out of six.

Temperature and *weather* have a significant impact on cultures, each *variety* has its own specificity and *latitude* is useful for sunshine impact. Looking on agronomic domain, we can easily see why these are the most common terms.

We have 36 common terms to at least two simulators and 374 specific terms over 410 in total. This means that simulators have eight per cent of terms in common, all other concepts are only present once. These are specific to their own simulator. The bottom part of the AOC-Poset is not shown in Figure 7 because it contains useless information, which corresponds to six concepts that match with the six simulators and all their specific attributes (not common with other simulators). We can observe that the grouping of attributes in the AOC-Poset concepts is variable and ranges from one in the top concepts, discussed above, to seven in concept 10 of Figure 7. We can also observe that the latter concept corresponds to an abstraction of simulators that process soil information, *i.e.* there is a semantic cohesion between these seven attributes. Concept four (at the left of the figure) groups four attributes semantically related too, and which correspond to plant phenology.

For the outputs (the AOC-Poset is not shown in the paper for the sake of space), we found only three common terms:

– The concept introducing attribute *daily*, common to three simulators out of four.
– The concept introducing attribute *phenology*, common to three simulators out of four.
– The concept introducing attribute *yearly*, common to three simulators out of four.

We observed that there are only three common terms to at least two simulators and 195 terms specific over 198 in total. This means that simulators have one per cent of terms in common, all other concepts are only present once. This low degree of commonality was predictable since each simulator has its own purpose and returns only useful data targeting that purpose.

As a final step in this evaluation work, we plan in the near future to initiate a discussion with the agronomist teams in order to validate and potentially improve the extraction result (AOC-Posets).

6.2 Discussion

Our work has multiple purposes. The first goal is to assist agronomists in the design of new models and simulators, then support the simulator integration by the IT team and allow a better migration towards a software product-line.

The second purpose is to provide a common vocabulary. In addition, the association dictionary helps to understand which terms are used including acronyms, abbreviations or written in different ways and offer a new standardization. We are quite confident that this will help agronomists when a new model has to be built by providing a standardized vocabulary and naming conventions.

When the IT team has to develop a new application, its first task is the simulator integration. This is a fastidious and error prone task based on manually cloning an existing integration. Based on the generated variability, which will be linked to code artefacts, cloning can be automated and simplified.

The last envisaged use is the migration of ITK software products to a Software Product Line, considering the AOC-Poset as a step towards the production of a complete feature/variability model. This migration will not be based on the source code only but will be completed by ontologies [5]. The dictionaries and AOC-Posets are relevant artifacts to this aim.

7 Related work

FCA has already been used for variability extraction in the domain of software product lines, to synthesize a feature model by exploring the AOC-Poset, the AC-Poset (Attribute-Concept Poset) or implicative systems [15, 17, 1, 7]. In these works, the formal context associates software product configurations to features. The feature model is a kind of logical tree exposing mandatory and optional features, feature groups (Or, Xor), and feature refinement through tree edges [13]. Cross-tree constraints (such as binary implication or mutual exclusion) may accompany the description. We ground the variability extraction on the same principles, using FCA as the revealer of commonalities and specifics. Compared to these works, the difference is that we do not focus on feature variability, but on input/output data variability. This provides a complementary view on the future product line.

Dealing with tree description could have been dealt taking inspiration from genterms (labelled trees provided with a generalization relation) [9] or using the pattern structure paradigm [11, 14]. The pattern structure paradigm is a way we will explore. Nevertheless, it was initially not clear how to determine the similarity and subsumption operators for our labelled directed (rooted) trees. In this work, we preferred to conduct a first study using local information, based on common nodes and edges, that can be encoded with basic formal contexts. A drawback in our approach is that tree portions will have to be rebuilt in a post-processing operation if we want to have a global view, but this has the merit of providing a simple initial solution.

Finally, this work also shares similar objectives and techniques with database schema integration [4], ontology merging [18], and common model extraction [2], including the need for linguistic analysis, and designing an integrated view, here on inputs or outputs of the simulators.

8 Conclusion

Linking software engineering and artificial intelligence with Formal Concept Analysis provides new tools and methods to improve current practices. We proposed to extract variability of simulators data schemata to improve future development of new simulators and to assist their integration in a new application.

In the short term, we have to integrate these in the software product line migration process, and to share this knowledge with the agronomist/IT team. All existing simulators do not have a data description schemata, especially outputs. This causes a lack of details about variability and reduces result impact. Including more data schemata coming from other simulators and asking the agronomist team to detail the outputs when they are absent will improve the quality of the results allowing us to give more assistance to ITK teams.

We plan in the future to work on variability extraction from source code of existing applications and linking this variability with what we extract from input/output data schemata. The ultimate goal is to provide a complete feature model which will enable agronomists and IT teams to easily configure new products by working together on common assets and using a unified vocabulary.

References

1. Al-Msie'deen, R., Huchard, M., Seriai, A., Urtado, C., Vauttier, S.: Reverse engineering feature models from software configurations using formal concept analysis. In: Proceedings of the Eleventh International Conference on Concept Lattices and Their Applications, Košice, Slovakia, October 7-10, 2014. pp. 95–106 (2014)
2. Amar, B., Guédi, A.O., Miralles, A., Huchard, M., Libourel, T., Nebut, C.: Using formal concept analysis to extract a greatest common model. In: Maciaszek, L.A., Cuzzocrea, A., Cordeiro, J. (eds.) ICEIS 2012 - Proceedings of the 14th International Conference on Enterprise Information Systems, Volume 1, Wroclaw, Poland, 28 June - 1 July, 2012. pp. 27–37. SciTePress (2012)
3. Bachmann, F., Clements, P.: Variability in software product lines. Tech. Rep. CMU/SEI-2005-TR-012, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA (2005), http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=7675
4. Batini, C., Lenzerini, M., Navathe, S.B.: A comparative analysis of methodologies for database schema integration. ACM Computer Survey **18**, 323–364 (1986)
5. Bécan, G., Acher, M., Baudry, B., Ben Nasr, S.: Breathing ontological knowledge into feature model synthesis: An empirical study. Empirical Software Engineering **21** (03 2015). https://doi.org/10.1007/s10664-014-9357-1
6. Carbonnel, J.: L'analyse formelle de concepts : un cadre structurel pour l'étude de la variabilité de familles de logiciels. PhD Thesis, Université de Montpellier (2018)
7. Carbonnel, J., Huchard, M., Nebut, C.: Modelling equivalence classes of feature models with concept lattices to assist their extraction from product descriptions. J. Syst. Softw. **152**, 1–23 (2019). https://doi.org/10.1016/j.jss.2019.02.027, https://doi.org/10.1016/j.jss.2019.02.027

8. Carbonnel, J., Huchard, M., Nebut, C.: Towards the Extraction of Variability Information to Assist Variability Modelling of Complex Product Lines. In: Proceedings of the 12th International Workshop on Variability Modelling of Software-Intensive Systems - VAMOS 2018. pp. 113–120. ACM Press, Madrid, Spain (2018). https://doi.org/10.1145/3168365.3168378, http://dl.acm.org/citation.cfm?doid=3168365.3168378
9. Daniel-Vatonne, M., Hemce, C.: On a tree-like representation for symbolic-numeric data and its use in galois lattice method. In: Proceedings of 18th International Conference of the Chilean Computer Science Society (SCCC '98), November 12-14, 1998, Antofagasta, Chile. pp. 48–57. IEEE Computer Society (1998). https://doi.org/10.1109/SCCC.1998.730782, https://doi.org/10.1109/SCCC.1998.730782
10. Dolques, X., Le Ber, F., Huchard, M.: AOC-posets: a scalable alternative to Concept Lattices for Relational Concept Analysis. In: CLA: Concept Lattices and their Applications. pp. 129–140. La Rochelle, France (Oct 2013), https://hal.archives-ouvertes.fr/hal-00916850
11. Ganter, B., Kuznetsov, S.O.: Pattern Structures and Their Projections. In: 9th Int. Conference ICCS'01, Stanford, CA, USA. pp. 129–142 (2001)
12. Ganter, B., Wille, R.: Formal Concept Analysis - Mathematical Foundations. Springer (1999)
13. Kang, K., Cohen, S., Hess, J., Novak, W., Peterson, A.: Feature-Oriented Domain Analysis (FODA) Feasibility Study. Tech. Rep. CMU/SEI-90-TR-021 (1990)
14. Leeuwenberg, A., Buzmakov, A., Toussaint, Y., Napoli, A.: Exploring pattern structures of syntactic trees for relation extraction. In: Baixeries, J., Sacarea, C., Ojeda-Aciego, M. (eds.) Formal Concept Analysis - 13th International Conference, ICFCA 2015, Nerja, Spain, June 23-26, 2015, Proceedings. Lecture Notes in Computer Science, vol. 9113, pp. 153–168. Springer (2015). https://doi.org/10.1007/978-3-319-19545-2_10, https://doi.org/10.1007/978-3-319-19545-2_10
15. Loesch, F., Ploedereder, E.: Restructuring variability in software product lines using concept analysis of product configurations. In: 11th European Conference on Software Maintenance and Reengineering, Software Evolution in Complex Software Intensive Systems, CSMR 2007, 21-23 March 2007, Amsterdam, The Netherlands. pp. 159–170 (2007). https://doi.org/10.1109/CSMR.2007.40
16. Ramos, J.: Using tf-idf to determine word relevance in document queries (01 2003)
17. Ryssel, U., Ploennigs, J., Kabitzsch, K.: Extraction of feature models from formal contexts. In: Software Product Lines - 15th International Conference, SPLC 2011, Munich, Germany, August 22-26, 2011. Workshop Proceedings (Volume 2). p. 4 (2011). https://doi.org/10.1145/2019136.2019141
18. Stumme, G., Maedche, A.: Ontology merging for federated ontologies on the semantic web. In: International Workshop for Foundations of Models for Information Integration (FMII-2001). pp. 413–418 (2001)