

SOVA – A Tool for Semantic and Ontological Variability Analysis

Nili Itzik and Iris Reinhartz-Berger

Department of Information Systems, University of Haifa, Israel
nitzik@campus.haifa.ac.il, iris@is.haifa.ac.il

Abstract. Variability analysis in Software Product Line Engineering (SPLE) utilizes various software-related artifacts, including requirements specifications. Currently, measuring the similarity of requirements specifications for analyzing variability of software products mainly takes into account semantic considerations. This might lead to failure to capture important aspects of the software behavior as perceived by users. In this paper we present a tool, called SOVA – Semantic and Ontological Variability Analysis, which introduces ontological considerations to variability analysis, in addition to the semantic ones. The input of the tool is textual requirements statements organized in documents. Each document represents the expectations from or the characteristics of a different software product in a line, while each requirement statement represents an *expected behavior* of that software product. The output is a feature diagram representing the variability in the input set of requirements documents and setting the ground for behavioral domain analysis.

Keywords: Software Product Line Engineering, Variability Analysis, Domain Analysis, Requirements Specifications, Ontology, Semantic Similarity

1 Introduction

As the complexity and variety of software products increased, the need to reuse software-related artifacts became very important. Software Product Line Engineering (SPLE) suggests an approach to systematically reuse artifacts, such as requirements specifications, design documents and source code, among different, yet similar, software products [3], [14]. Such reuse of artifacts often raises a significant challenge of variability management. *Variability* in this context can be defined as “the ability of an asset to be efficiently extended, changed, customized, or configured for use in a particular context” [7].

Viewing software requirements as the drivers of different development activities and methods, several studies have suggested using requirements specifications for variability analysis of software products. In these studies, requirements are operationalized or realized by features, and variability is mainly represented as *feature diagrams* – tree or graph structures that describe the characteristics of a software product line and the relationships and dependencies among them [8]. The current studies

commonly apply only semantic similarity metrics, which focus on similarities of terminology, in order to identify and analyze variability. As we will elaborate later, using only semantic considerations might lead to failure to capture important aspects of the software behavior, such as its triggers, pre-conditions, and post-conditions.

In [16], we suggest combining semantic and ontological considerations for calculating similarity. In particular, a behavior is described in terms of the initial state of a system before the behavior occurs, the external events that trigger the behavior, and the final state of the system after the behavior occurs. We use semantic metrics to evaluate the similarity of related behavioral elements and utilize this similarity to analyze variability. To support this approach, we have developed a tool, called SOVA – Semantic and Ontological Variability Analysis. This tool gets requirements documents written in plain text. Each document represents a different software product in the line and is divided into requirements statements. Each requirement statement, which may be composed of several sentences, reflects a use case, a user story, or any unit that represents a single expected or existing behavior of a software product. The variability of requirements is then analyzed, yielding a feature diagram. The resultant feature diagrams are behavior-driven and set the ground for behavioral domain analysis.

The rest of this paper is structured as follows. Section 2 reviews related work, exemplifying limitations of current approaches. Section 3 presents the main processes of the approach and their support in the SOVA tool. Finally, Section 4 summarizes and refers to future development plans.

2 Related Work

In the context of analyzing software products variability, different studies have suggested ways to use textual requirements to generate variability models, such as feature diagrams or Orthogonal Variability Models (OVM) [14].

In [19], a tool, named ArborCraft, is presented. This tool creates feature diagrams by grouping similar requirements using a hierarchical agglomerative clustering algorithm and semantic similarity measures – Latent Semantic Analysis (LSA) [10]. Feature variants are then identified using a Requirements Description Language and semantic considerations. In [4-5], publicly available repositories of product descriptions are utilized. Based on these repositories and the conditional probabilities between features occurrences, a probabilistic feature diagram is created using an incremental diffusive clustering algorithm. In [13], a semi-automatic method for constructing OVM diagrams is introduced. This method extracts functional requirements profiles (FRPs), represented as "verb-direct object" pairs, using expert knowledge and linguistic clues. The variability model is created using heuristic rules, such as: "If diverse values are identified for a case, then alternative choice(s) should be made."

All the above methods employ only semantic considerations. In particular, they may result with high similarity values for requirements that use similar terminology, even if the pre-conditions, the triggers, and the post-conditions of the corresponding behaviors are different. For example, the requirements "The system should be able to

report on any user update activities” and “Any user should be able to report system activities” may result in a very high value of semantic similarity, since both refer to “system”, “user”, and “report”. In fact, LSA [10] results in a similarity value of 1 for these requirements, implying that their semantic meanings are identical. However, these requirements are quite different: the first requirement represents behavior that is internal and likely aims at detecting suspicious user update activities. The second requirement, on the other hand, represents a behavior triggered by an external user who intends to report his/her system activities.

Another limitation of current studies is that they take into consideration the full text of a requirement statement. Such statements might include aspects (e.g., intermediate outcomes) that are less or not relevant for analyzing variability from an external perspective of a user or a customer. Such a view of the *expected behaviors* of software systems is important for reaching different reuse decisions, e.g., when conducting feasibility studies, estimating software development efforts, or adopting SPLE.

To overcome the above limitations, we proposed in [16] to combine semantic and ontological considerations when calculating similarity and analyzing variability. We further demonstrated that our approach outperforms LSA when examining the similarity of functional requirements. Here we present the tool we have developed to support that approach. The tool is named SOVA – Semantic and Ontological Variability Analysis.

3 The SOVA Tool

Fig. 1 presents the main processes supported by the SOVA tool, namely requirements parsing, behavioral similarity calculation, and feature diagram creation. Next we elaborate on each process and its support in the tool. Additional material can be found at <http://mis.hevra.haifa.ac.il/~iris/research/SOVA/>.

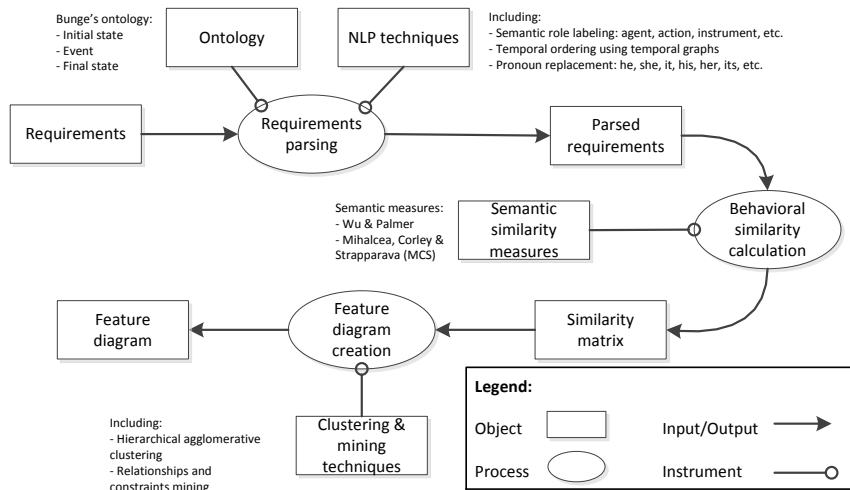


Fig. 1. An overview of the processes and flows supported by the SOVA tool

3.1 Requirements Parsing

During the first step, the input requirements are parsed. This is done by two main instruments: natural language processing (NLP) techniques and an ontological model.

First, a semantic role labeling (SRL) approach [6] is used to associate the parts of a requirement statement with their specific semantic roles. Five semantic roles are currently supported due to their special importance to requirements in general and functional requirements in particular: (1) Agent – Who performs? (2) Object (a.k.a. Patient) – On what object is it performed? (3) Instrument – How is it performed? (4) Temporal modifier (AM-TMP) – When is it performed? And (5) Adverbial modifier (AM-ADV) – In what conditions is it performed? A sixth label – Action – is handled to answer the question: What is performed? This label holds the sentence's predicate or verb.

Considering those labels and applying temporal order [11] and coreference resolution¹ [15] techniques, the tool identifies behavioral vectors, each representing an action or a pre-condition. Using concepts taken from Bunge's ontological model [1-2], the behavioral vectors are then classified into initial states that represent pre-conditions of the behavior, external events that trigger the behavior, and final states that represent post-conditions or outputs of the behavior. These three “types” of behavioral elements (namely, initial states, external events, and final states) were suggested in [17-18] for defining an external view of behavior. The classification of the vectors to these behavioral elements is mainly done by analyzing the *agent* and the *action* parts of the vectors and using the temporal order of the vectors [16].

The screenshot presented in Fig. 2 exemplifies the outcome of the parsing requirements activity. The field at the top of this screen enables choosing a particular requirements file and browsing its requirements statements (in the middle part of this screen). Each requirement statement includes one or more sentences. Each sentence appears in a separate row, where the number to its left indicates the requirement to which it belongs. Requirement 2, for example, is composed of two sentences. Choosing a particular sentence displays the parsing of the entire requirement to which the sentence belongs in the bottom part of this screen. The second requirement in Fig. 2, for example, is parsed into three behavioral vectors. The first vector is classified as an initial state, since it represents a pre-condition (labeled as a temporal modifier). The second vector, representing a login operation, is classified as an external event, since it is performed by an external agent – the librarian. Finally, the third vector is classified as a final state, as it describes an internal operation performed by the system after the librarian logs in.

During the parsing process, the tool further supports interactions with the user, namely, a requirements engineer or a domain analyst. In particular, the user can edit the ontological class, change the order of the parsed behavioral vectors, update the original requirements, and view the semantic role labeling output (the SRL button).

¹ Coreference resolution replaces pronouns (e.g., he, she, and it) with their anaphors (i.e., the nouns to which they refer).

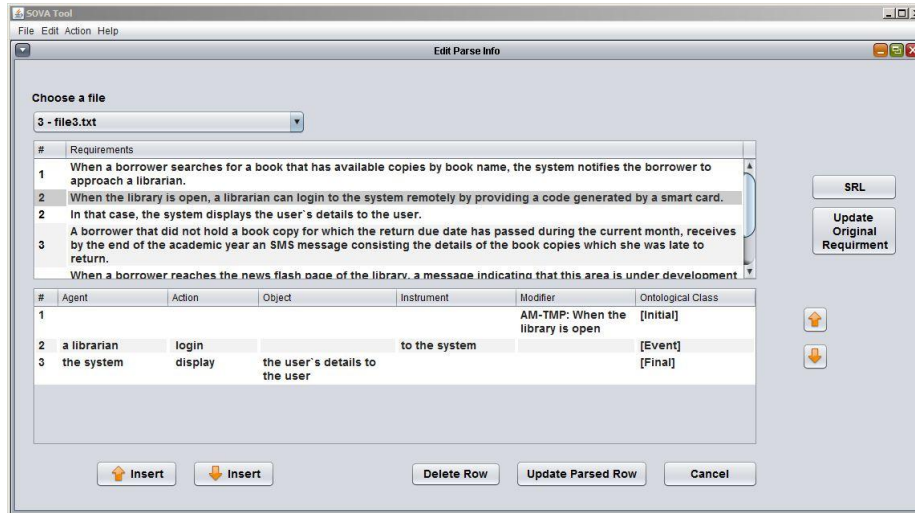
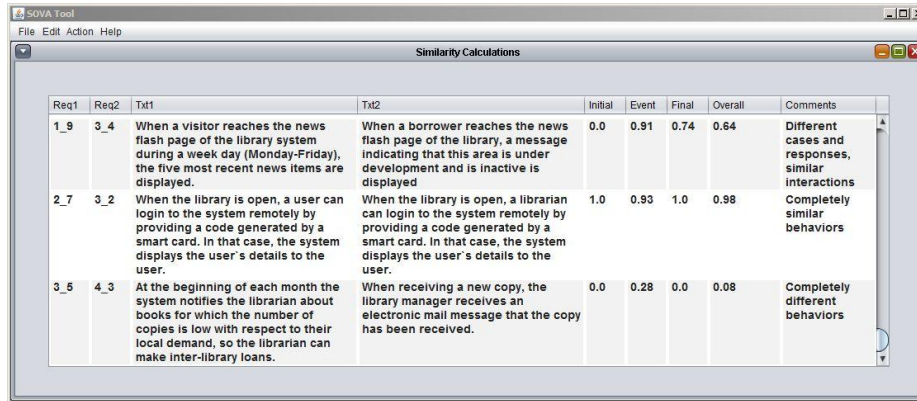


Fig. 2. A screenshot of the requirements parsing outcome

3.2 Behavioral Similarity Calculation

In the second process, the behavioral similarity of each pair of requirements (either from the same document or from different documents) is calculated. The behavioral similarity is the weighted average of the semantic similarities of their behavioral vectors. In other words, the behavioral similarity is the weighted average of the semantic similarities of their initial states, external events, and final states. For calculating the semantic similarities of the behavioral elements different semantic measures can be used. Here we use MCS [12] to measure phrases' similarity and Wu and Palmer [20] to measure words' similarity. The user can further set the weights for agents, actions, objects, and instruments similarities. Perceiving agents and actions as the dominant components in behavioral vectors similarities, Fig. 3 exemplifies the outcome of the behavioral similarity calculation process in SOVA, using 0.3, 0.4, 0.2, and 0.1 for weighting agents, actions, objects, and instruments, respectively. The screen displays (in the right side) the initial state, external event, final state, and overall similarities for each pair of requirements in the source files. The overall similarity is calculated using initial state, external event and final state weights of 0.2, 0.3, and 0.5, respectively, perceiving the final state as the most influencing factor on the overall similarity.

In Fig. 3, for example, the first pair of requirements (the ninth requirement in the first input file and the fourth requirement in the third input file) represents different cases (initial states) and responses (final states), but similar interactions (external events) in which someone (visitor or borrower) reaches the new flash page of the library. The requirements in the second row represent very similar behaviors, which differ only in their agents (users vs. librarians). Finally, the requirements in the third row represent completely different behaviors.



Req1	Req2	Txt1	Txt2	Initial	Event	Final	Overall	Comments
1_9	3_4	When a visitor reaches the news flash page of the library system during a week day (Monday-Friday), the five most recent news items are displayed.	When a borrower reaches the news flash page of the library, a message indicating that this area is under development and is inactive is displayed	0.0	0.91	0.74	0.64	Different cases and responses, similar interactions
2_7	3_2	When the library is open, a user can login to the system remotely by providing a code generated by a smart card. In that case, the system displays the user's details to the user.	When the library is open, a librarian can login to the system remotely by providing a code generated by a smart card. In that case, the system displays the user's details to the user.	1.0	0.93	1.0	0.98	Completely similar behaviors
3_5	4_3	At the beginning of each month the system notifies the librarian about books for which the number of copies is low with respect to their local demand, so the librarian can make inter-library loans.	When receiving a new copy, the librarian receives an electronic mail message that the copy has been received.	0.0	0.28	0.0	0.08	Completely different behaviors

Fig. 3. A screenshot of the behavioral similarity calculation outcome

3.3 Feature Diagram Creation

In the third process, we use the calculated similarity values in order to create a feature diagram that represents the variability found in the input requirements documents. To this end, we utilize a hierarchical agglomerative clustering algorithm. This algorithm starts with putting each requirement in a separate cluster. In each iteration, the algorithm merges the closest clusters, namely, clusters whose average requirements' similarities is the highest. The output of this algorithm is a binary tree of clusters. To better represent the analyzed variability, another pass is performed to flatten sub-trees whose similarities are alike. To demonstrate this pass, consider the schematic tree in the left side of Fig. 4. The leaves of this tree represent requirements (or actually clusters with single requirements), numbered 1 to 5, while the inner nodes represent clusters with several requirements. Each inner node exhibits its identity (e.g., C1:2_4) and the overall similarity of the constituting requirements. Note that the sub-tree whose root is C1:2_4 includes very similar requirements, namely R1, R2, and R4. Therefore, in the flattened tree (in the right side of the figure), the three requirements have the same parent. In contrast, the node C3_1:2:4 holds a requirement, R3, which is quite different from the other related requirements, R1, R2, and R4. Thus, grouping the four requirements together is unjustified. Instead R3 and C1:2_4 become siblings in the flattened tree.

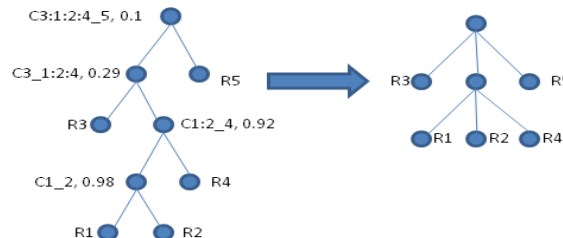


Fig. 4. Illustration of flattening the clustering outcome in the feature diagram creation stage

Optionality as well as OR- and XOR-grouped features are deduced examining the appearance of the different requirements in the input requirements documents. The final output is presented in featureIDE format. FeatureIDE is an eclipse plug-in that supports different phases of the feature-oriented software development [9]. It is user friendly. In particular, the feature diagrams can be presented horizontally or vertically, the requirements can be presented as description of leaf nodes, and the diagrams can be exported to a variety of feature diagram formats.

The SOVA tool enables generating feature diagrams according to different behavioral views, namely, considering only the similarity of the initial states, the external states, the final states, or the overall behaviors. Thus, and as opposed to existing approaches and tools, the variability of the requirements can be analyzed from different perspectives. For example, considering only the similarity of final states may provide an output-driven variability perspective, while considering the external events provides a functional variability perspective.

4 Summary and Future Work

We presented a tool, named SOVA – Semantic and Ontological Variability Analysis – that supports identifying and analyzing behavioral variability of software products based on requirements specifications. The tool combines semantic and ontological considerations through a three stage process that includes parsing the requirements using NLP techniques and Bunge’s ontological model, calculating the behavioral similarity of software requirements using semantic measures, and generating feature diagrams using a hierarchical agglomerative clustering algorithm. All these processes are done automatically and the user is only required to set weights for the different semantic similarities.

In the future, we intend to extend the tool support in several ways. First, we intend to involve the user throughout the process and to allow him/her to provide intermediate feedback which will be taken into consideration in the following stages. Second, we intend to derive state variables from intermediate states and not just from initial and final states. These state variables may further help identify the commonality and variability of software products by refining the external view. Finally, we intend to handle requirements statements that represent “swarms” of behaviors (including branches and loops) and not just single ones. This will enable us to analyze relationships between requirements and not just individual requirements.

References

1. Bunge, M. (1977). *Treatise on Basic Philosophy*, vol. 3, *Ontology I: The Furniture of the World*. Reidel, Boston, Massachusetts.
2. Bunge, M. (1979). *Treatise on Basic Philosophy*, vol. 4, *Ontology II: A World of Systems*. Reidel, Boston, Massachusetts.
3. Clements, P. and Northrop, L. (2001). *Software Product Lines: Practices and Patterns*. Addison-Wesley.

4. Davril, J. M., Delfosse, E., Hariri, N., Acher, M., Cleland-Huang, J., and Heymans, P. (2013). Feature model extraction from large collections of informal product descriptions. *The 9th Joint Meeting on Foundations of Software Engineering*, pp. 290-300.
5. Dumitru, H., Gibiec, M., Hariri, N., Cleland-Huang, J., Mobasher, B., Castro-Herrera, C., and Mirakhorli, M. (2011). On-demand feature recommendations derived from mining public product descriptions. *33rd IEEE International Conference on Software Engineering (ICSE'11)*, pp. 181-190.
6. Gildea, D. and Jurafsky, D. (2002). Automatic Labeling of Semantic Roles. *Computational Linguistics* 28 (3), pp. 245-288.
7. Jaring, M. (2005). Variability engineering as an Integral Part of the Software Product Family Development Process, Ph.D. thesis, The Netherlands.
8. Kang, K. C., Cohen, S. G., Hess, J. A., Novak, W. E., and Peterson, A. S. (1990). Feature-oriented domain analysis (FODA) – feasibility study. Technical report no. CMU/SEI-90-TR-21). Carnegie-Mellon University, Pittsburgh.
9. Kastner, C., Thum, T., Saake, G., Feigenspan, J., Leich, T., Wielgorz, F., and Apel, S. (2009). FeatureIDE: A tool framework for feature-oriented software development. *31st IEEE International Conference on Software Engineering (ICSE'09)*, pp. 611-614.
10. Landauer, T. K., Foltz, P. W., and Laham, D. (1998). Introduction to Latent Semantic Analysis. *Discourse Processes*, 25, pp. 259-284.
11. Mani, I., Verhagen, M., Wellner, B., Lee, C. M., and Pustejovsky, J. (2006). Machine learning of temporal relations. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, pp. 753-760.
12. Mihalcea, R., Corley, C., and Strapparava, C. (2006). Corpus-based and knowledge-based measures of text semantic similarity. *The 21st national conference on Artificial intelligence (AAAI'2006)*, Vol. 1, pp. 775-780.
13. Niu, N. and Easterbrook, S. (2008). Extracting and modeling product line functional requirements. In *the 16th IEEE International Requirements Engineering conference (RE'08)*, pp. 155-164.
14. Pohl, K., Böckle, G., and van der Linden, F. (2005) *Software Product-line Engineering: Foundations, Principles, and Techniques*, Springer.
15. Raghunathan, K., Lee, H., Rangarajan, S., Chambers, N., Surdeanu, M., Jurafsky, D., and Manning, C. (2010). A Multi-Pass Sieve for Coreference Resolution. *The conference on Empirical Methods in Natural Language Processing (EMNLP'10)*, pp. 492-501.
16. Reinhartz-Berger, I., Itzik, N., and Wand, Y. (2014). Analyzing Variability of Software Product Lines Using Semantic and Ontological Considerations *Proceedings of the 26th international conference on Advanced Information Systems Engineering (CAiSE'14)*, LNCS 8484, pp. 150-164.
17. Reinhartz-Berger, I., Sturm, A., and Wand, Y. (2013). Comparing Functionality of Software Systems: An Ontological Approach. *Data & Knowledge Engineering* 87, pp. 320-338.
18. Reinhartz-Berger, I., Sturm, A., and Wand, Y. (2011). External Variability of Software: Classification and Ontological Foundations. *The 30th International Conference on Conceptual Modeling (ER'2011)*, LNCS 6998, pp. 275-289.
19. Weston, N., Chitchyan, R., and Rashid, A. (2009). A framework for constructing semantically composable feature models from natural language requirements. In *Proceedings of the 13th International Software Product Line Conference*, pp. 211-220.
20. Wu, Z. and Palmer, M. (1994). Verbs semantics and lexical selection. *The 32nd annual meeting on Association for Computational Linguistics*, pp. 133-138.