Supporting Product Line Adoption by Combining Syntactic and Textual Feature Extraction

András Kicsi[1], László Vidács[1,2], Viktor Csuvik[1], Ferenc Horváth[1], Árpád Beszédes[1] and Ferenc Kocsis[3]

[1] Department of Software Engineering
[2] MTA-SZTE Research Group on Artificial Intelligence
University of Szeged, Szeged, Hungary
{akicsi,lac,csuvikv,hferenc,beszedes}@inf.u-szeged.hu
[3] SZEGED Software Ltd., Szeged, Hungary
kocsis.ferenc@szegedsw.hu

Abstract. Software product line (SPL) architecture facilitates systematic reuse to serve specific feature requests of new customers. Our work deals with the adoption of SPL architecture in an existing legacy system. In this case, the extractive approach of SPL adoption turned out to be the most viable method, where the system is redesigned keeping variants within the same code base. The analysis of the feature structure is a crucial point in this process as it involves both domain experts working at a higher level of abstraction and developers working directly on the program code. In this work, we propose an automatic method to extract feature-to-program connections starting from a very high level set of features provided by domain experts and existing program code. The extraction is performed by combining and further processing call graph information on the code with textual similarity between code and high level features. The context of our work is an industrial SPL adoption project of a large scale logistical information system written in an 4G language, Magic. We demonstrate the benefits of the combined method and its use by different stakeholders in this project.

Keywords: Product lines, SPL, feature extraction, variability mining, Magic, 4GL, information retrieval, call graphs.

1 Introduction

Maintaining parallel versions of a software satisfying various customer needs is challenging. Many times the clone-and-own solution [1] is chosen because of short term time and effort constraints. As the number of product variants increases, a more viable solution is needed through systematic code level reuse. A natural step towards more effective development is the adoption of product line architecture [2]. The extractive approach analyzes existing products to obtain feature models and build the product line architecture [3]. An advantage of the

extractive approach in general is that several reverse engineering methods exist to support feature extraction and analysis [4–6].

Product line adoption is usually approached from three directions: the proactive approach starts with domain analysis and applies variability management from scratch. The reactive approach incrementally replies to the new customer needs when they arise. When there are already a number of systems in production, the extractive approach seems to be the most feasible choice. During the extractive approach the adoption process benefits from systematic reuse of existing design and architectural knowledge.

In this paper, we report on an ongoing product line adoption project in which the extractive approach is being used. Our subject is a legacy high market value, wholesaler logistics system, which was adapted to various domains in the past using clone-and-own method. It is developed using a fourth generation language (4GL) technology, Magic [7], and in this project the product line architecture is to be built based on an existing set of products developed in the Magic XPA language. Although there is reverse engineering support for usual maintenance activities [8, 9], the special structure of Magic programs makes it necessary to experiment with targeted solutions for coping with features. Furthermore, approaches used in mainstream languages like Java or C++ need to be re-considered in the case of systems developed in 4GLs. For instance, in the traditional sense there is no source code, rather the developer sets up user interface and data processing units in a development environment and the flow of the program follows a well-defined structure.

In this work, we concentrate on the feature identification and analysis phase of the project and describe an efficient method for this purpose. This is a well studied topic in the literature for mainstream languages [5], but the same for 4GL is less explored. The method starts from a very high level set of features provided by domain experts, and uses information extracted from the existing program code. The extraction is performed by combining and further processing call graph information on the code with textual similarity between code and high level features, essentially working simultaneously with structural (syntactic) and conceptual (text based) information. Similar approaches have been previously proposed for traditional object oriented systems, e.g. by Al-msie'deen et al. [10].

In previous work [11], we presented our information retrieval approach to feature extraction similar to our textual analysis, but that information alone proved to be very noisy and incomplete. We not only combine conceptual information with structural one, but present an efficient method for filtering the data as well. This results in a set of information that is more suitable for performing the SPL adoption by various stakeholders of the project including domain experts, architects and programmers. In summary, the contributions of this paper are the following:

1. A method for feature extraction by combining sytactic and textual information.
2. Details for applying the approach in an 4GL technology, and the associated experimental results.

3. Details on the use of the approach in an ongoing SPL adoption industrial project from various stakeholder perspectives.

The paper is organized as follows. We present the background of our project with the peculiarities of the underlying technology and the overview of our method in Section 2. The details of our method for combining structural and conceptual feature extraction for Magic systems and the associated experimental results are presented in Section 3. Section 4 deals with the benefits of the approach and how it is used in other phases of the SPL adoption project. Related work is briefly introduced in Section 5, before concluding the paper Section 6.

2 Feature extraction and abstraction of Magic applications

2.1 Product line adoption in a clone-and-own environment

The decision of migrating to a new product line architecture is hard to make. Usually there is a high number of derived specific products and the adoption process poses several risks and may take months [12–14]. The subject system of our analysis is a leading pharmaceutical wholesaler logistics system started more than 30 years ago. Meanwhile more than 20 derived variants of the system were introduced at various complexity and maturity levels with independent life cycles and isolated maintenance. Our industrial partner is the developer of market leading solutions in the region, which are implemented in the Magic XPA 4GL language.

This work is part of an industrial project aiming to create a well-designed product line architecture over the isolated variants. The existing set of products provide appropriate environment for an extractive SPL adoption approach. Characterizing features is usually a manual or semi-automated task, where domain experts, product owners and developers co-operate. Our aim is to help this process by automatic analysis of the relation of higher level features and map program level entities to features.

The 4GL environment used to implement the systems requires different approaches and analysis tools than today's mainstream languages like Java [8, 15]. For example there is no source code in its traditional sense. The developers work in a fully fledged development environment by customizing several properties of programs. Magic program analysis tool support is not comparable to mainstream languages, hence this is a research-intensive project. In our current work we used our previously available tools for static analysis of Magic applications and IR-based feature extraction [11].

The feature extraction process is challenged, since product variants themselves are written in 4 different language versions as shown in Figure 1. In case of the oldest Magic V5 systems, there is a high demand on the migration to a newer version. UniPaaS 1.9 introduced huge changes in the language by using the .NET engine for applications. Most systems are implemented in that version. The newest Magic XPA 3.x line of the language lies close to the uniPaaS v1.9

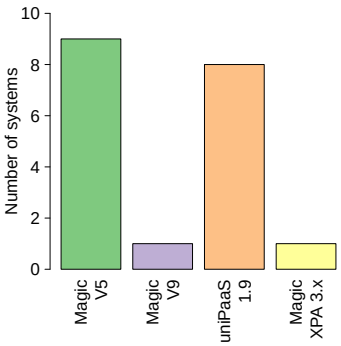

Fig. 1. The Magic language versions of the 19 currently active product variants

Variants	Largest application size Programs	Models	Tables
19	4 251	822	1 065

Table 1. Overview of the common program base of application variants

systems. The overall size of the common codebase of product variants is shown in Table 1. The first column states that there are 19 currently active variants of the application, while the remaining columns contain the main specifications of the largest variant. Magic is a data-intensive language, which clearly reflects on these values as well, containing a large amount of data tables.

2.2 Feature extraction approach

During product line adoption's feature extraction phase various artifacts are obtained to identify features in an application [16]. This phase is also related to feature location. The analysis phase targets common and variable properties of features and prepares the reengineering phase. This last phase migrates the subject system to the product line architecture.

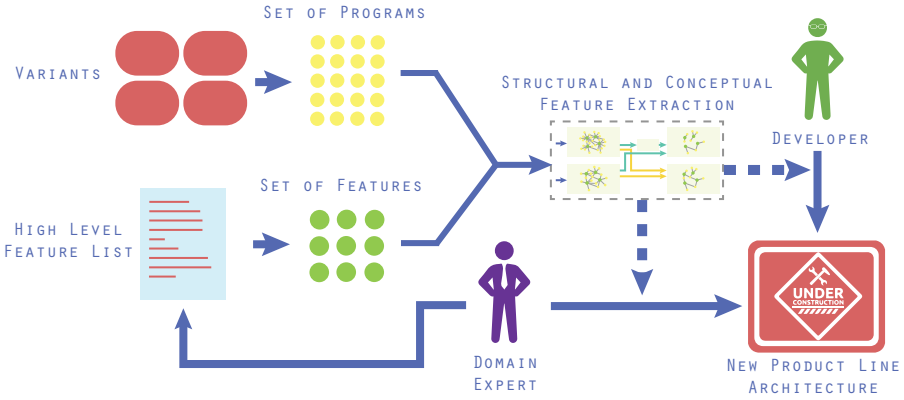

Fig. 2. An illustration of feature extraction as a part of product line adoption

In this phase of the research project we address the feature extraction phase. Our inputs are the high level features of the system and the program code. We apply a semi-automated process as in [4]. High level features are collected by domain experts from the developer company. The actual task is to establish a link between features and main components of the Magic applications. Although there exists a common analysis infrastructure for reverse engineering 4GL languages [17, 8, 9], the actual program models differ.

Figure 2 illustrates our current approach to feature extraction. We assign a number of elements for each high level feature, this information helping the work of developers and domain experts working on the new product line architecture. During the assignment we mainly rely on structural information attained on call dependency by constructing a call graph of the variant in question. This results in high number of located elements, crucial for the development of product line architecture, but the large amount of data can be hard to grasp in its entirety.

We combine this method with information retrieval, which can also make it easier to cope with a 4GL language by utilizing conceptual connections, and is successfully applied in software development tasks, such as in traceability scenarios for object oriented languages [18]. A comprehensive overview of Natural Language Processing (NLP) techniques – including Latent Semantic Indexing, the technique we chose – is provided by Falessi et al. [19]. In our previous work [11] we already presented our LSI-based approach to feature extraction. LSI is already known to be capable of producing quality results combined with structural information [10].

2.3 The Structure of a Magic Application

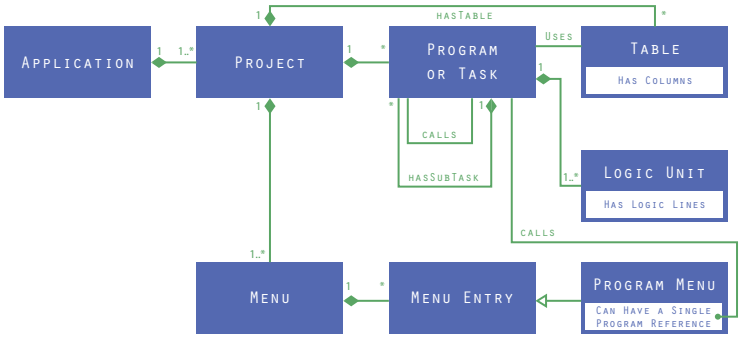

Fig. 3. Illustration on the elements of a Magic application

Being a fourth generation language, Magic does not completely follow the structure of a traditional programming language. Figure 3 represents the most important components of the Magic language from our current point of view.

Each software written in Magic is called an application. These can be built up from one or more projects. In turn, each project can have any number of programs which contain the actual logic of the software. The tasks branching directly from a project are called programs. These can have their own subtasks and be called anywhere in the project like methods in a traditional programming environment, however their subtasks can only be called by the (sub)task containing them. Any task or subtask can access data through tables.

It is also possible for a program to be called through menus, which are controls designed to provide user intervention and usually start a process by calling programs. Having sufficient information on menus, we used these as a base for the call graph in structural feature extraction, deriving calls from menus.

3 Feature extraction experiments

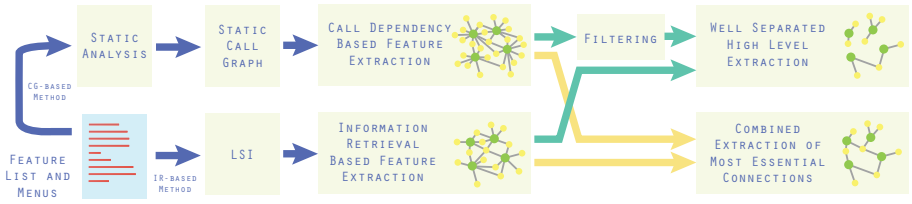

Fig. 4. A more detailed view on the feature extraction process

In this section we present the feature extraction methods used based on call dependency and textual similarity, as well as the combination and possible filtering options. Figure 4 illustrates the processes described in this section. Our static analysis is specific to the Magic language. One of the variants of our subject system was selected by domain experts to be used as a starting point for the product line adoption. It is a specific variant involving 4251 programs, 822 models and 1065 data tables. Our experiments presented in this paper were done on this specific variant. We have been provided with a feature list structured in a tree format consisting of three levels which have 10, 42 and 118 unique elements respectively. From these we chose the upper level to display our results, the features of this level are listed in Figure 5. The numbers shown here are in accordance with the numbers we present on our later graph examples.

3.1 Feature extraction using (task) call dependency

This approach relies mostly on the program structure, but especially on the call dependencies between programs and task. To construct a call graph from these dependencies we use the process that can be seen in Figure 6. For the sake of simplicity, this figure represents only a minimalistic example of a Magic

1 – Manufacturing	6 – Administrator interventions
2 – Interface	7 – Supplier order management
3 – Access management	8 – Invoicing
4 – Quality control	9 – Master file maintenance
5 – Stock control	10 – Customer order reception

Fig. 5. The higher level features of the system

application. We emphasis tasks and programs with squares, while other program elements like projects, logic units, logic lines, etc. are shown as circles.

We have the abstract semantic graph (ASG) as the base, which is provided by our static source code analyzer tool. As the next step we add the call edges to the graph by examining Magic components that operate as calls between tasks and programs. Finally, in the last two step of the process we eliminate some nodes and edges from the graph, keeping only the necessary ones *i.e.,* call edges, tasks and programs. From the CG we obtain the features by running a customized breadth-first search algorithm from specific staring points determined by menu entries. In Figure 7 a graph representation of the CG based results can be seen.

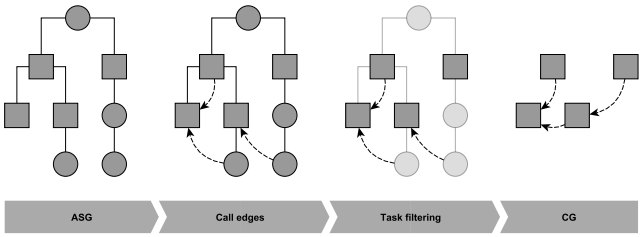

Fig. 6. The process of calculating the call graph

3.2 Textual similarity

Information retrieval (IR) techniques focus on the information content of data, dealing with natural language at a semantic level. This results in a more versatile approach concerning the form of the data, like the language used. In our specific case this can be a huge boon, since the systems processed use different versions of the Magic language, and this would cause serious problems for a technique dealing with the specific syntax of the language. With IR techniques on the other hand, we can process the natural language parts of code, freeing us from the burden of having to solve differences of language versions and syntax of Magic. Latent Semantic Indexing (LSI) [20] is an IR technique capable of measuring semantic similarity between textual data. It is widely used throughout software engineering, mainly in cases involving natural language.

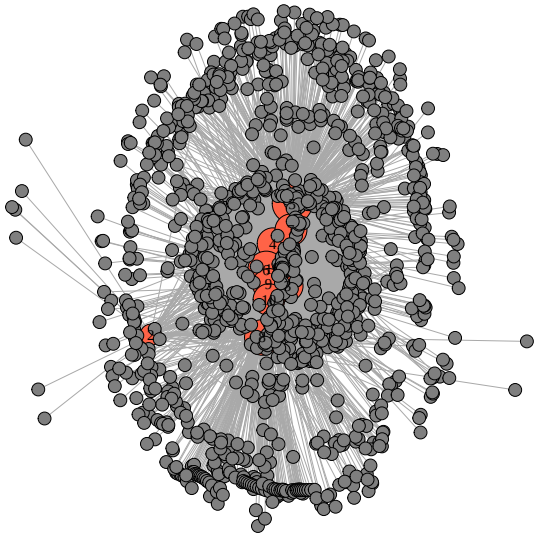

Fig. 7. Graph visualization of the set of results obtained by the call graph technique

A more complete summary of our feature extraction work with LSI is presented in our previous work. Our current experiments differ from these in some respects, the main differences being that these experiments work on a completely different base variant of the system, and consider the top level of features of an updated, more precise feature list obtained from domain experts.

In Figure 8 we can see a graph representation of the results of the top features paired with the programs of a system. Though the structural information obtained from call graph is more thorough, it is more suitable for the developers rather than domain experts. With purely structural information it is hard to separate along the features, having many programs laying the groundwork for any single feature it is hard to grasp the overall aim. Conceptual analysis separates more agreeably along the semantics of the feature, hence it can be more valuable for domain experts.

3.3 Combined technique

As already introduced, the two methods we used for program assignment to features use fundamentally different methods for achieving their results. Consequently, the results themselves also show a significant difference, overlapping only partially. The set of programs for the techniques presented are shown on the left side of Figure 9. Each slice of the diagram represents a top level feature and its colors indicate the number of programs detected by each technique. IR represents the result set of the information retrieval technique, CG represents

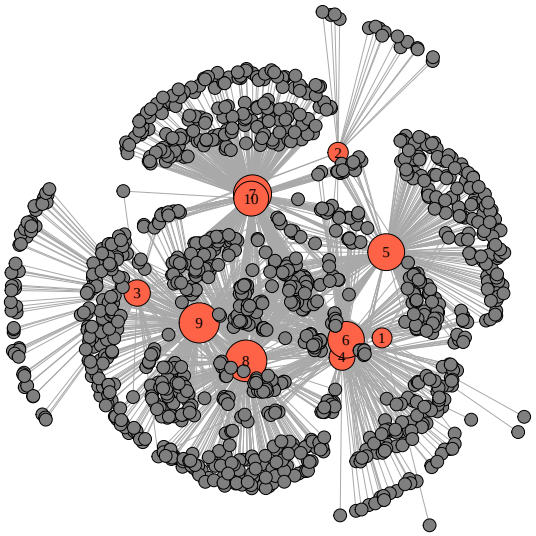

Fig. 8. Graph visualization of the set of results obtained by the information retrieval technique

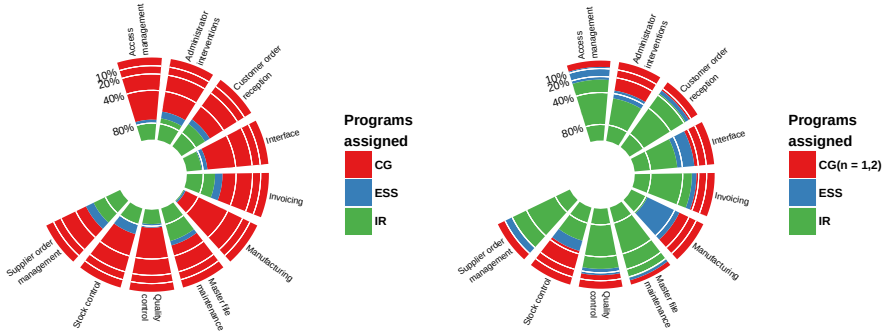

Fig. 9. The size of our result sets for each feature. Results shown on the left, results with filtering on the right

the pairs attained by call graph, while ESS represents the set of programs considered most essential, detected by both techniques. The left side of Figure 11 shows the number of programs assigned in each set, the abbreviations match the ones explained for the previous figure.

The call graph dependency technique produces vast amounts of matches for each feature. These matches build on the real calls inside the code, hence they are considered a reliable source of information. It is important to note that this is only static and not dynamic call information, thus at runtime it is not necessary

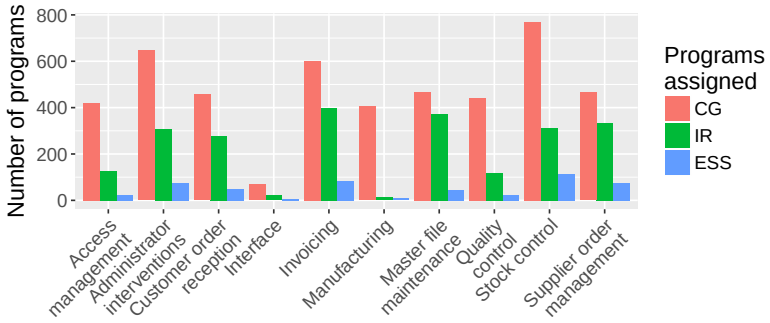

Fig. 10. Number of programs found most essential for each feature

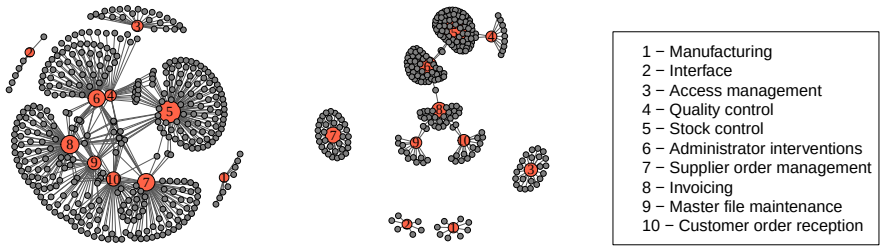

Fig. 11. Graph visualization of the set of programs deemed most essential. Results shown on the left, results with filtering on the right

for every call to occur. The abundance of programs found by this technique can undoubtedly be useful for developers, but it also presents a problem of coping with the large amount of data not distinguishable in any manner.

The conceptual method produces fewer programs for each feature, but further examination of random cases revealed that even considering this, a significant amount of noise presents itself. Textual similarity works with very little information in these cases, hence it is likely for similar wording or more general words like "list" to produce misleading matches, occurring in the text of many features.

Looking at only the intersection of the connections found by these two techniques we find that this set of connections takes into account both the structural and conceptual information, producing only connections which are indeed present on both levels. This results in a clearer, more straightforward set of connections, which contains the most essential findings of the two techniques.

As we could see before, the structural information produces a rather large amount of matches for each feature, and we observed that there is a considerable overlap between features. We decided to attempt to clear these matches too with a filtering technique applied on the structural information output, which filters out less specific programs. The filtering technique works with a number

n, which denotes the maximal number of features a program can connect before it is considered less specific and is filtered out from the program set of features. This removes the programs with less information value and results in even more straightforward groups of programs for each feature. On the right side of Figure 9 and Figure 11 we can see the results of the common structural and conceptual connections of this filtered approach, featuring only programs with maximum two connections. It is apparent from the graph that features are much better separated, providing a suitable high level glance at the background of features without much technical details, ideal for top level understanding.

Examining the graphs we can come to many interesting conclusions. For example feature number 7 is behaving like any other feature considering the purely conceptual or purely structural viewpoint, its common graph provides a clearer picture, apparently connecting through a group of more general features to large number of other features. In the filtered case however, it is nicely separated with a group of unique programs specific to the feature itself.

4 Discussion

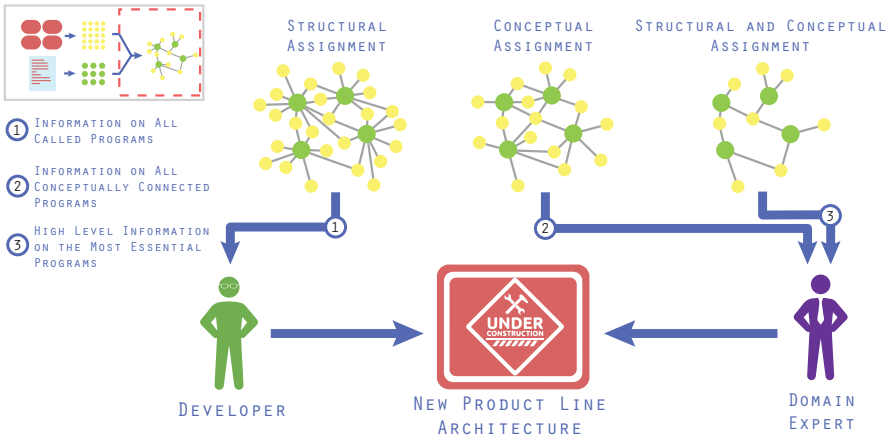

Fig. 12. The possible ways of usage of the results of various feature extraction techniques in helping product line adoption

Figure 12 highlights how various feature extraction techniques can be used to help in building the new product line architecture.

– Structural Extraction - Provides a detailed, widespread analysis. It is good for developers, since they are required to have knowledge of all of the programs called by a feature, too much for domain experts.

- Conceptual Extraction - For domain experts on the other hand, all called programs can be too much. This approach however introduces conceptual dependencies, although with too much noise for smooth work.
- Combination (ESS) - Grasps the essence of features, more fit for domain experts. While constructing the new architecture, the domain experts need to judge properly which parts of the variants should be adopted. In this decision making process the results of this extraction highly decreases complexity, additionally in the future it can facilitate test planning.

Besides these, we would like to mention some other possible ways to use the results. Firstly, connections are not necessarily observable through the calls of the system, programs can for instance connect by accessing to the same data objects. This means that not every connection will present itself on the call graph. These however can be found via conceptual feature extraction, since it is likely that programs using the same data are conceptually connected to the same feature. This is why the programs detected by the conceptual extraction and not discovered via structural information can still be valuable. This data is rather noisy, but still represents a ready source of the semantically connected programs to each feature, hence it can be a reliable starting point with manual evaluation for domain experts looking for connected programs.

Additionally, the structural information produced can be tailored according to our intent, filtering out the more general programs, which provides the possibility to form even better separated sets of programs, making it easier to attain a high level knowledge about the procedures and the basic structure of a system.

We provided our results to our industrial partner, who has already commenced on constructing the new SPL architecture. The work is proceeding well, product line adoption seems to go according the plans and the results of our experiments are utilized in the process.

5 Related Work

The literature of reverse engineering 4GL languages is not extensive. By the time the 4GL paradigm arisen, most papers coped with the role of those languages in software development, including discussions demonstrating their viability. The paradigm is still successful, although only a few works are published about the automatic analysis and modeling 4GL or specifically Magic applications. The maintenance of Magic applications is supported by cost estimation and quality analysis methods [21, 22, 17]. Architectural analysis, reverse engineering and optimization are visible topics in the Magic community [15, 23, 9, 8], and after some years of Magic development migration to object-oriented languages [24] as well.

SPL has a widespread literature, and over the last 8-10 years it has gained even more popularity. All three phases of feature analysis (identification, analysis, and transformation) are tackled by researchers. A recommended mapping study on recent works on feature location can be read in [5].

Software product line extraction is a time-consuming task. To speed up this activity, many semi-automatic approaches has been proposed [25–27]. Reverse

engineering is a popular approach which has recently received an increased attention from the research community. With this technique missing parts can be recovered, feature models can be extracted a set of features, etc. [25, 28]. Applying these approaches companies can migrate their system into a software product line. However, changing to a new development process is risky and may have unnecessary costs. The work of J. Krüger et al. [29] supports cost estimations for the extractive approaches and provides a basis for further research.

Feature models are considered first class artifacts in variability modeling. Haslinger et al. [27] present an algorithm that reverse engineers a FM for a given SPL from feature sets which describe the characteristics each product variant provides. She et al. [30] analyze Linux kernel (which is a standard subject in variability analysis) configurations to obtain feature models. LSI is applied for recovering traceability links between various software artifacts. The work of Marcus and Maletic [18] is an early paper on applying LSI for this purpose. Eyal-Salman et al. [6] use LSI for recovering traceability link between features and source code with about 80% success rate, but experiments are done only for a small set of features of a simple java program. IR-based solution for feature extraction is combined with structural information in the work of Almsie'deen et al. [10]. Further research deals with constraints in a semi-automatic way [31] both for functional and even nonfunctional [32] feature requirements.

A possible future goal could be to make the system dynamically configurable, which is a problem known as Dynamic SPL [33–39]. Today, many application domains demand runtime reconfiguration, for which the two main limitations are handling structural changes dynamically and checking the consistency of the evolved structural variability model during runtime [36]. However, as reported in [40], the current research on runtime variability is still heavily based on the decisions made during design time. Coping with uncertainty of dynamic changes also needs to be addressed [41].

Several existing approaches can be adapted to 4GL environment, although none of the above cited papers cope with 4GL product lines directly.

6 Conclusions

An ongoing industrial project was presented, which undergoes a software product line adoption process. In this work, we concentrated on the feature extraction and analysis aspects of the project, which are fundamental parts of the effort because further architecture redesign and implementation are and will be based on this information. For feature extraction we used two fundamental approaches: one based on computing structural information form the code in for of call-graphs, and the other in which conceptual information was automatically extracted from the textual representation of high level feature models and the code. However, the combination of the two pieces of information had to be performed and processed in such a way that the resulting models are most useful for project participants.

Experimental results show that the final models are significantly more comprehensible, and hence directly usable (though in various forms) by domain ex-

perts, architects, developers and other stakeholders of the project. The extracted information and the associated toolset is currently in use by our industrial partner in this ongoing effort, however in later phases further refinements of the approach are to be expected. For instance, (semi-)automatic classification of the feature sets will probably be needed.

Although the approach was implemented in Magic, a 4GL technology, we believe that the fundamental method could be suitable for other more traditional paradigms as well after the necessary adaptations.

Acknowledgment

Ferenc Kocsis was supported in part by the Hungarian national grant GINOP-2.1.1-15-2015-00370. András Kicsi, László Vidács, Viktor Csuvik, Ferenc Horváth and Árpád Beszédes were supported in part by the European Union, co-financed by the European Social Fund (EFOP-3.6.3-VEKOP-16-2017-00002).

References

1. Fischer, S., Linsbauer, L., Lopez-Herrejon, R.E., Egyed, A.: Enhancing Clone-and-Own with Systematic Reuse for Developing Software Variants. In: 2014 IEEE International Conference on Software Maintenance and Evolution, IEEE (sep 2014) 391–400
2. Clements, P., Northrop, L.: Software Product Lines: Practices and Patterns. Addison-Wesley Professional (2001)
3. Krueger, C. In: Easing the Transition to Software Mass Customization. Springer Berlin Heidelberg, Berlin, Heidelberg (2002) 282–293
4. Kästner, C., Dreiling, A., Ostermann, K.: Variability Mining: Consistent Semi-automatic Detection of Product-Line Features. IEEE Transactions on Software Engineering **40**(1) (2014) 67–82
5. Assunção, W.K.G., Vergilio, S.R.: Feature location for software product line migration. In: Proceedings of the 18th International Software Product Line Conference on Companion Volume for Workshops, Demonstrations and Tools - SPLC '14, New York, New York, USA, ACM Press (2014) 52–59
6. Eyal-Salman, H., Seriai, A.D., Dony, C., Al-msie'deen, R.: Recovering traceability links between feature models and source code of product variants. In: Proceedings of the VARiability for You Workshop on Variability Modeling Made Useful for Everyone - VARY '12, New York, New York, USA, ACM Press (2012) 21–25
7. Magic Software Enterprises Ltd.: Magic Software Enterprises. http://www.magicsoftware.com (last visited May 2017)
8. Nagy, C., Vidács, L., Ferenc, R., Gyimóthy, T., Kocsis, F., Kovács, I.: MAGISTER: Quality Assurance of Magic Applications for Software Developers and End Users. In: 26th IEEE International Conference on Software Maintenance, IEEE Computer Society (September 2010) 1–6
9. Nagy, C., Vidács, L., Ferenc, R., Gyimóthy, T., Kocsis, F., Kovács, I.: Solutions for reverse engineering 4gl applications, recovering the design of a logistical wholesale system. In: Proceedings of CSMR 2011 (15th European Conference on Software Maintenance and Reengineering), IEEE Computer Society (March 2011) 343–346

10. Al-msie'deen, R., Seriai, A.D., Huchard, M., Urtado, C., Vauttier, S.: Mining features from the object-oriented source code of software variants by combining lexical and structural similarity. In: 2013 IEEE 14th International Conference on Information Reuse & Integration (IRI), IEEE (aug 2013) 586–593
11. Kicsi, A., Vidács, L., Beszédes, A., Kocsis, F., Kovács, I.: Information retrieval based feature analysis for product line adoption in 4gl systems. In: Proceedins of the 17th International Conference on Computational Science and Its Applications – ICCSA 2017, IEEE (2017) 1–6
12. Clements, P.C., Jones, L.G., McGregor, J.D., Northrop, L.M.: Getting there from here: a roadmap for software product line adoption. Communications of the ACM **49**(12) (dec 2006) 33
13. Clements, P., Krueger, C.: Eliminating the adoption barrier. IEEE Software **19**(4) (jul 2002) 29–31
14. Catal, C., Cagatay: Barriers to the adoption of software product line engineering. ACM SIGSOFT Software Engineering Notes **34**(6) (dec 2009) 1
15. Harrison, J.V., Lim, W.M.: Automated Reverse Engineering of Legacy 4GL Information System Applications Using the ITOC Workbench. In: 10th International Conference on Advanced Information Systems Engineering, Springer-Verlag (1998) 41–57
16. Ballarin, M., Lapeña, R., Cetina, C.: Leveraging Feature Location to Extract the Clone-and-Own Relationships of a Family of Software Products. In: Proceedings of the 15th International Conference on Software Reuse: Bridging with Social-Awareness - Volume 9679. Springer-Verlag New York, Inc. (2016) 215–230
17. Nagy, C., Vidács, L., Ferenc, R., Gyimóthy, T., Kocsis, F., Kovács, I.: Complexity measures in 4gl environment. In: Computational Science and Its Applications - ICCSA 2011, Lecture Notes in Computer Science. Volume 6786 of Lecture Notes in Computer Science., Springer Berlin / Heidelberg (2011) 293–309
18. Marcus, A., Maletic, J.: Recovering documentation-to-source-code traceability links using latent semantic indexing. In: 25th International Conference on Software Engineering, 2003. Proceedings., IEEE (2003) 125–135
19. Falessi, D., Cantone, G., Canfora, G.: A comprehensive characterization of NLP techniques for identifying equivalent requirements. In: Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement - ESEM '10, New York, New York, USA, ACM Press (2010) 1
20. Deerwester, S.C., Dumais, S.T., Landauer, T.K., Furnas, G.W., Harshman, R.A.: Indexing by Latent Semantic Analysis. Journal of the American Society of Information Science **41**(6) (1990) 391–407
21. Verner, J., Tate, G.: Estimating Size and Effort in Fourth-Generation Development. IEEE Software **5** (1988) 15–22
22. Witting, G., Finnie, G.: Using Artificial Neural Networks and Function Points to Estimate 4GL Software Development Effort. Australasian Journal of Information Systems **1**(2) (1994) 87–94
23. Ocean Software Solutions: Homepage of Magic Optimizer. http://www.magic-optimizer.com (last visited May 2017)
24. M2J Software LLC: Homepage of M2J. http://www.magic2java.com (last visited May 2017)
25. Valente, M.T., Borges, V., Passos, L.: A Semi-Automatic Approach for Extracting Software Product Lines. IEEE Transactions on Software Engineering **38**(4) (jul 2012) 737–754

26. Assunção, W.K.G., Lopez-Herrejon, R.E., Linsbauer, L., Vergilio, S.R., Egyed, A.: Multi-objective reverse engineering of variability-safe feature models based on code dependencies of system variants. Empirical Software Engineering **22**(4) (aug 2017) 1763–1794
27. Haslinger, E.N., Lopez-Herrejon, R.E., Egyed, A.: Reverse Engineering Feature Models from Programs' Feature Sets. In: 18th Working Conference on Reverse Engineering, IEEE (oct 2011) 308–312
28. Lima, C., Chavez, C., de Almeida, E.S.: Investigating the Recovery of Product Line Architectures: An Approach Proposal. Springer, Cham (may 2017) 201–207
29. Krüger, J., Fenske, W., Meinicke, J., Leich, T., Saake, G.: Extracting software product lines: a cost estimation perspective. In: Proceedings of the 20th International Systems and Software Product Line Conference on - SPLC '16, New York, New York, USA, ACM Press (2016) 354–361
30. She, S., Lotufo, R., Berger, T., Wąsowski, A., Czarnecki, K.: Reverse engineering feature models. In: Proceeding of the 33rd international conference on Software engineering - ICSE '11, New York, New York, USA, ACM Press (2011) 461
31. Bagheri, E., Ensan, F., Gasevic, D.: Decision support for the software product line domain engineering lifecycle. Automated Software Engineering **19**(3) (2012) 335–377
32. Siegmund, N., Rosenmüller, M., Kuhlemann, M., Kästner, C., Apel, S., Saake, G.: SPL Conqueror: Toward optimization of non-functional properties in software product lines. Software Quality Journal **20**(3-4) (2012) 487–517
33. Lee, K., Kang, K.C., Lee, J.: Concepts and Guidelines of Feature Modeling for Product Line Software Engineering. In: Software Reuse Methods Techniques and Tools. Volume 2319. Springer, Berlin, Heidelberg (2002) 62–77
34. Baresi, L., Quinton, C.: Dynamically Evolving the Structural Variability of Dynamic Software Product Lines. 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (2015)
35. Bashari, M., Bagheri, E., Du, W.: Dynamic Software Product Line Engineering: A Reference Framework. International Journal of Software Engineering and Knowledge Engineering **27**(02) (2017) 191–234
36. Capilla, R., Bosch, J., Trinidad, P., Ruiz-Cortés, A., Hinchey, M.: An overview of Dynamic Software Product Line architectures and techniques: Observations from research and industry. Journal of Systems and Software **91**(1) (may 2014) 3–23
37. Uchôa, A.G., Bezerra, C.I.M., Machado, I.C., Monteiro, J.M., Andrade, R.M.C.: ReMINDER: An Approach to Modeling Non-Functional Properties in Dynamic Software Product Lines. Springer, Cham (may 2017) 65–73
38. Hinchey, M., Park, S., Schmid, K.: Building Dynamic Software Product Lines. IEEE Computer Society **45**(10) (2012) 22–26
39. Lee, J.: A Feature-Oriented Approach to Developing Dynamically Reconfigurable Products in Product Line Engineering. 10th International Software Product Line Conference (2006) 131–140
40. Bencomo, N., Lee, J., Hallsteinsen, S.: How dynamic is your Dynamic Software Product Line? DiVA project (EU FP7 STREP) (2010) 61–67
41. Classen, a., Hubaux, A., Sanen, F., Truyen, E., Vallejos, J., Costanza, P., De Meuter, W., Heymans, P., Joosen, W.: Modelling Variability in Self-Adaptive Systems: Towards a Research Agenda. Proceedings of International Workshop on Modularization, Composition and Generative Techniques for Product-Line Engineering **1**(2) (2008) 19–26