Domain analysis with reverse-engineering for GUI feature models

Michaela BAČÍKOVÁ

Dept. of Computers and Informatics, Technical University in Košice, Letná 9, 042 00 Košice, Slovakia

michaela.bacikova@tuke.sk

Abstract. *A domain analyst must often joint different factors when performing a domain analysis. The domain itself has its specifics therefore it is necessary to study it perfectly. On the other hand the analyst must reflect the knowledge he gained about a particular domain, which are generally non-formalized, into the area of software system design, which is not always an easy task. Existing software systems as a part of the target domain can provide this knowledge in a formalized form. These software systems are however not directly usable for an automatic processing because of high level of implementation details. In this work we review our research in the area of automatic domain analysis with reverse engineering of domain knowledge. Outputs of such analysis are feature models. The main focus of our research is graphical user interface of software systems, because it is the closest to the domain by being designed for domain users. We analyze user interfaces to gain domain knowledge and we create feature models.*

Keywords

domain analysis, graphical user interface, components, feature models, reverse engineering

1. Introduction

Domain analysis (DA) is a process that is currently most often used in the software systems design and analysis. The Fig. 1 describes the most common process of domain analysis. It is necessary to collect information from various sources: a) **domain users and experts**; b) **existing documents**; and c) **existing software systems**. Then the different kinds of information are categorized, analyzed and some forms of a domain models are created.

The domain models have many forms. Commonly used is the feature model in different notations (or example original FODA notation described in [1] or Czarnecki-Eisenecker notation described in [2]). Feature modeling methods analyze product lines and resulting domain model contains varying and consistent features in a domain and also defines the vocabulary used in the domain, which defines concepts, ideas and phenomena within the system. This model (often with a connection with generators and libraries of reusable components or frameworks) is used for creating a new software system or editing an existing system. The domain analysis is often performed by a **domain analyst**.

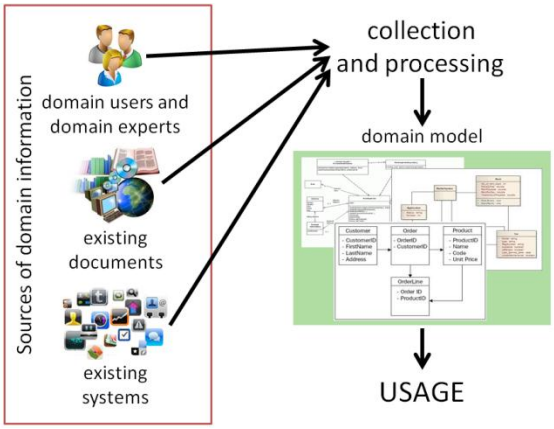

Fig. 1. An illustration of the domain analysis process

2. Problem Statement

Modeling and creating domain models from gained information is now widely supported by numerous methodologies and tools. Gaps remain especially in the areas of **data collection**. Problems of existing approaches in this area can be divided into three groups based on the information source. Each group will be analyzed in the next chapters.

2.1 Problems in the area of data collection from domain users and domain experts

Data collection from domain users and experts is often carried out through reviews, questionnaires and forms. These methods are often *time consuming* and require both *willingness of users and experts* and a certain level of *skill of the domain analyst*. On one hand, a user or a domain expert do not always have the time or mood to fill out some questionnaires or talk with analyst. The domain analyst on the other hand must be able to ask the right questions to gain the information he needs.

2.2 Problems in the area of data collection from non-formalized existing documents

Various automatic techniques are used. The most common are NLP (Natural Language Processing) methods and techniques of AI (Artificial Intelligence). Sometimes also an existing design documentation of software systems is analyzed. The biggest problem of these approaches is the ambiguity of the natural language and therefore there is always a need for additional control and adjustments by a human expert. Finally, it is necessary to gain existing documents to be analyzed and they may not even exist for the particular domain.

2.3 Problems in the area of data collection existing software systems

The approaches for automatic collection of domain information from existing software systems mainly use static analysis of source codes and databases. Databases or source codes are however *not primarily intended for the user*. Hence the author of database or code is **not forced to use domain vocabulary** during development. So the domain terms may not even be included in the database or source code analyzed, they can be written in another language, abbreviations can be used, or there may be some other language barrier.

The analysis of databases is very easy: the names of tables and relations between tables can be exactly determined and represent domain terms and relations. There is, however, *no description of domain processes*.

On the other hand, source codes contain descriptions of domain processes in a form of methods or functions. There is however a *high level of implementation details* getting in the way, preventing the domain analysis to be carried out. *Generalization*, which is currently widely used in the implementation of reusable systems, represents another barrier. The aim of generalization is to use generic terms that can be used to describe objects and thus to ensure reusability of the system also in other domains. This may hinder the domain analysis: a domain-specific model should contain specific terms, not general.

3. Proposed Solution

There are many methods and tools supporting communication with users and domain experts. Also in the area of domain analysis of non-formalized documents there is a significant number or research papers proposing different methodologies and techniques of NLP and AI (see chapter V.). Since these two areas are more or less exhausted, we decided to focus on the last area, the analysis of existing software systems, as this area is only little explored in relation to domain analysis.

Based on the problems and facts identified in the previous chapter we think that more appropriate target for an automatic DA is a **user interface**. A user who comes from the target domain has a *direct access to the UI*. For the user to be able to use the system effectively it must be built with respect to *understandability*, i.e. it must contain *terms from the domain*. It also should describe *domain processes* in a form of event sequences executed on the UI.

Currently applications are often created as component-based because of reusability support. In component-based applications it is possible to separate source code components from user interface components to a certain level. Although there are still very unclear boundaries between application presentation layer and source code layer. Using reflection and aspect oriented programming allows us to separate the implementation details from domain information that we need.

In this paper we will describe the basic graphical components and a method for formalizing them (or their groups) into a domain model illustrated on examples. This will represent a basis for our future research in the area of automated GUI domain analysis.

3.1 Previous research

This paper is based on our previous work in this field, see [3], [4]. In our recent work [5] we analyzed basic stereotypes of creating user interfaces and based on them we defined basic rules for GUI formalization that will be used in implementation of our Automatic GUI Domain Analysis Method (AGDAM). We created an automated tool for extracting information from UI and automated recorder of user actions performed on UI. This tool we want to use in our further research in the area of DA.

4. Motivation

Besides the tasks of creating a new software system, or editing an existing software system, the resulting domain model can be used also in other areas, for which we primarily aim our future work. The domain model can be used for example in model driven engineering (MDD) for *generating various software artifacts or whole software systems*. For example Ristic *et al.* [6] use form specifications to generate forms, however when using their tool the user alone must enter the form specifications into the computer. An automatic domain analysis of existing systems with forms (for example web sites) for such form specifications could help the user with formalizing the requirements and the user would only edit the results of the DA according to his needs.

If there already is an existing software system, based on the domain model of such system and with the help of dynamic analysis of user handling this system and generators, *documentation* (for example a user guide) and different types of *models* (for future development) can be generated.

Last but not least area of use is an automatic evaluation of *domain usability*. Domain usability is a usability related to domain terms contained in the interface.

For the user interface to be usable in a domain the domain user must understand it, we say that it must be *understandable*. With our method, it is possible to extract all terms located in the interface and analyze them for membership in the domain. If there are terms, that do not belong to the domain (for example in an internet banking system, there should not be terms like "student", "lecturer" and so on), then they represent a domain usability issue.

5. State of the Art

The domain analysis was first defined by Neighbors [7] in 1980. By introducing the Draco approach, a code generator system that works by integrating reusable components Neighbors stresses, that domain analysis is the key factor for supporting reusability of analysis and design.

Widely used approach for domain analysis is the FODA (Feature Oriented Domain Analysis) approach [1], that aims for analysis of software product lines by comparing the different and similar features or functionalities of systems. The method explains what the outputs of domain analysis are but remains vague about the process to obtain them. Very similar to the FODA approach and practically based on it, is the DREAM (Domain Requirements Asset Manager) approach by Mikeyong *et al.* [8]. They also perform commonality and variability analysis of product lines, but with the difference of modeling domain requirements, not features or functionalities of systems. Many approaches and tools support the FODA method, e.g. Ami Eddi [9], CaptainFeature [10], RequiLine [11], ASADAL [12], pure::variants [13].

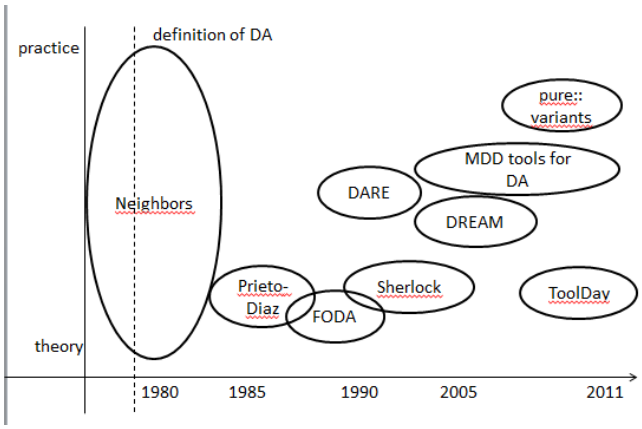

Fig. 2. A time chart of existing approaches and tools for domain analysis (inspired by the time chart by Frakes *et. al* [20])

Not only the process of DA is supported by some approaches, but also reusability feature by providing a library of reusable components, frameworks or libraries. Such approaches are for example the early Prieto-Diaz approach [14] that uses a set of libraries; or the later Sherlock environment [15] that uses a library of frameworks.

The latest efforts are in the area of MDD (Model Driven Development). The aim of MDD is to shield the complexity of the implementation phase by domain modeling and generative processes. The MDD principle support for example the Czarnecki project Feature Plug-in [16] or his newest effort Clafer [17] and an eclipse plug-in FeatureIDE [18] by Thüm and Kästner.

ToolDay (A Tool for Domain Analysis) [19] is a tool with support for every phase of DA with possibilities for validation, generating models and exporting to different formats.

All tools and methodologies listed here support the DA process by analysis of data, supporting summarizing, clustering of data, or modeling features. But the input for domain analysis (i.e. the information about the domain) always come from the users, or it is not specified where they come from. Only the DARE (Domain analysis and reuse environment) tool from Prieto-Díaz [20] primarily aims for *automatic* collection and structuring of information and creating a reusable library. The data are collected not only from human resources, but also automatically from *existing source codes and text documents*. But as mentioned above, the source codes do not have to contain the domain terms and domain processes. The DARE tool *does not analyze the UIs* specifically.

A time chart of the approaches can be seen on Fig. 2.

6. Domain Modeling of Basic Graphical Components

The fundamental building block of each GUI is usually a window or screen, to which all components are put into. It represents the domain of the application and the title of the window represents the name of the domain. The basic components which we will deal with in this paper are: windows, containers, tabpanes, buttons (classic buttons, radio buttons, checkboxes).

Fig. 3. GUI of iAlertU

We use FeatureIDE notation for our examples of domain models because of their easily readable form and very clear representation of hierarchical relations. Hierarchical relations in the model examples can be derived from component structure. These models can be used in feature modeling. The basic domain dictionary can

be created automatically and the user can edit the feature model himself after its automatic generation.

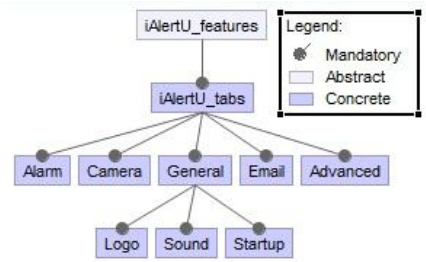

Fig. 4. Basic domain model derived from iAlertU GUI

On Fig. 3 there is a simple GUI of an open source Java application iAlertU, which we choose for this experiment. The GUI is simple, it uses only one window and tabpanes are used for representing its basic structure.

As we determined in our previous works, the hierarchy of the basic terms can be derived from the container structure. In this case it is the structure of tabs and containers in them. On Fig. 4 there is a basic domain model derived from iAlertU GUI based on the structure of tabs. The name of the main feature (*iAlertU*_features) is derived from the title of the window. The names of features are derived from the tab titles. All these properties can be programmatically determined by our approach.

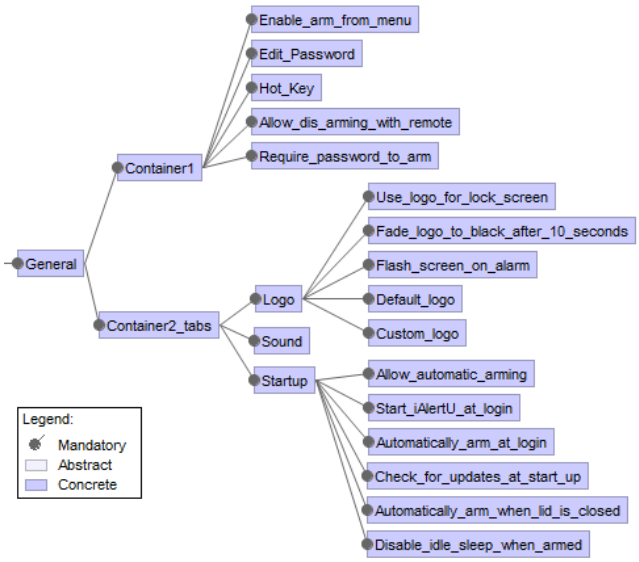

Fig. 5. Extended feature model derived from iAlertU GUI

This however can be taken even further. To address all functionalities of the window, we can add every feature of the GUI to the model. In our approach every component displayed in the UI can be identified in the component tree automatically.

The domain descriptors (for example the domain descriptor of the General tab is the "General" string, i.e. the string which is visible in the UI), used in the model are derived from the component attributes, such as *title*, *labelFor* attribute of a Label component, *toolTipText*, etc.

On Fig. 5 there is an example of extended feature model derived from the iAlertU UI. To simplify it, we displayed only a part of the whole model, the General tab.

We have already implemented an automatic method for analyzing graphical user interfaces of Java applications, which is able to derive the component tree and also to identify components according to their domain descriptions. Our next step is to design and implement an algorithm for automatic derivation of feature models from such component tree.

7. Conclusion

In this paper we described the theory behind domain analysis and we identified the main gaps in the DA of software systems. We determined that the best possible source of information is the presentation layer of software systems, because it is already formalized and it is close to the user (and therefore also close to the target domain). We provided analysis of the actual state of the art and an insight of our previous and recent research. We outlined our aim of designing and implementing the AGDAM method that is a center of our current and future research and we proposed ways of extracting information from various graphical components that will serve for implementation of the AGDAM method, which we illustrated on examples.

Acknowledgment

This work was supported by VEGA Grant No. 1/0305/11 – Co-evolution of the artifacts written in domain-specific languages driven by language evolution.

References

[1] K C Kang, S G Cohen, J A Hess, W E Novak, and A S Peterson, "Feature-Oriented Domain Analysis (FODA) Feasibility Study," Tech. Rep. CMU-SEI-90-TR-21, 1990.

[2] Krzysztof and Eisenecker, Ulrich W. Czarnecki, Generative programming: methods, tools, and applications. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 2000.

[3] Michaela Bačíková, Jaroslav Porubän, and Peter Václavík, "First Step for GUI Domain Analysis: Formalization," Journal of Computer Science and Control Systems, vol. 4, no. 1, pp. 65--69, 2011.

[4] Michaela Bačíková, Jaroslav Porubän, and Dominik Lakatoš, "Introduction to Domain Analysis of Web User Interfaces," in Proceedings of the Eleventh International Conference on Informatics (Informatics'11), Rožňava, 2011, pp. 103-108.

[5] Michaela Bačíková and Jaroslav Porubän, "Analysing Stereotypes of Creating Graphical User Interfaces," Central European Journal of Computer Science, submitted for publication 2012.

[6] S., Aleksic, S., Lukovic, I., Banovic, J. Ristic, "Form-Driven Application Generation: A Case Study," in Proceedings of the Eleventh International Conference on Informatics, INFORMATICS'2011, Rožňava, Slovakia, 2011, pp. 115-120.

[7] James M Neighbors, Software Construction Using Components, 1980, Dizertačná práca, Department of Information and Computer Science, University of California, Irvine.

[8] Mikyeong Moon, Keunhyuk Yeom, and Heung Seok Chae, "An Approach to Developing Domain Requirements as a Core Asset Based on Commonality and Variability Analysis in a Product Line," IEEE Trans. Softw. Eng., vol. 31, no. 7, pp. 551--569, July 2005.

[9] Bendash Thomas, Unger Peter, Eisenecker Ulrich Czarnecki Krzysztof, "Generative programming for Embedded Software: An Industrial Experience," in Generative programming and component engineering, ACM SIGPLAN/SIGSOFT Conference (GCPE 2002) Proceedings, LCNS 2487, vol. 1, Pitsburgh PA, USA, 2002, p. 164.

[10] CaptainFeature, stránka projektu na SourceForge.net, posledná aktualizácia 2005. [Online]. https://sourceforge.net/projects/captainfeature

[11] Stránka projektu RequiLine, posledná aktualizácia 2005. [Online]. https://www-lufgi3-informatik.rwth-aachen.de/TOOLS/requiline/index.php

[12] Prehľad CASE softvérového rámca ASADAL, Postech Software Engineering Laboratory, posledná aktualizácia 2011. [Online]. http://selab.postech.ac.kr/ASADAL-Simple_Overview.pdf

[13] Otto-von-Guericke-Universität Magdeburg and Fraunhofer Instituts Rechnerarchitektur und Softwaretechnik. (2011) pure:variants. http://www.pure-systems.com/pure_variants.49.0.html.

[14] William Frakes, Ruben Prieto-Diaz, and Christopher Fox, "DARE: Domain analysis and reuse environment," Ann. Softw. Eng., vol. 5, pp. 125--141, January 1998.

[15] R. Prieto-Díaz, "Reuse Library Process Model," Electronic Systems Division, Air Force Systems Command, USAF, Hanscom AFB, AM, STARS Reuse Library Program Contract F19628-88-D-0032, Task IS40, 1991.

[16] Andrea Valerio, Giancarlo Succi, and Massimo Fenaroli, "Domain analysis and framework-based software development," SIGAPP Appl. Comput. Rev., vol. 5, no. 2, pp. 4--15, September 1997.

[17] Michal Antkiewicz and Krzysztof Czarnecki, "FeaturePlugin: feature modeling plug-in for Eclipse," in Proceedings of the 2004 OOPSLA workshop on eclipse technology eXchange, eclipse '04, Vancouver, British Columbia, Canada, 2004, pp. 67--72.

[18] Kacper Bąk, Krzysztof Czarnecki, and Andrzej Wasowski, "Feature and meta-models in Clafer: mixed, specialized, and coupled," in Proceedings of the Third international conference on Software language engineering (SLE'10), Eindhoven, The Netherlands, 2011, pp. 102--122.

[19] Thomas Thüm, Christian Kästner, Sebastian Erdweg, and Norbert Siegmund, "Abstract Features in Feature Modeling," in 15th InternationalSoftware Product Line Conference (SPLC), 2011, pp. 191--200.

[20] Liana Lisboa, Vinicius Garcia, Eduardo de Almeida, and Silvio Meira, "ToolDAy: a tool for domain analysis," International Journal on Software Tools for Technology Transfer (STTT), vol. 13, no. 4, pp. 337-353, 2011.