Analyzing Graphical User Interfaces with DEAL

[1]*Michaela Bačíková (3[d] year)*
Supervisor: [1]*Jaroslav Porubän*

[1,2]Dept. of Computers and Informatics, FEI TU of Košice, Slovak Republic

[1]michaela.bacikova@tuke.sk, [2]jaroslav.poruban@tuke.sk

***Abstract*—At present, almost every existing application has a graphical user interface (GUI), which represents the first contact of a user with an application. Therefore the GUI should be created with respect to understandability and domain content. The domain terms, relations and processes should be captured in the GUI so it would be usable. We base our research on this presumption and we created a tool for automated domain analysis, DEAL, which is able to analyze the existing user interfaces and to create a domain model from the gained information. In the current state, we are improving DEAL to be able to derive new relations from the analyzed GUI. In this paper we present the DEAL tool and its features and we also discuss our plans for the future.**

***Keywords*—Domain analysis, Domain extraction, Graphical user interface, DEAL, Component-based software engineering**

I. Introduction

Domain analysis (DA) [1] is the process of analyzing, understanding and documenting a particular domain - the terms, relations and processes in the domain. The result of DA is a *domain model* which is used for creation of new software systems. DA is performed by a *domain analyst* [2]. In Table I we summarized three basic sources of domain information currently used to perform DA. The first column lists the particular information sources. The methods for gaining information are listed in the second column and the last column is a summary of disadvantages of the methods for each of the knowledge source type.

According to our research [3] many methods for dealing with the first two information sources exist. The last area, analysis of domain applications, is the least explored area. The main reason is the high level of implementation details – it is hard to extract domain information from code. Reverse engineering deals with extracting relevant information from existing applications, but with the goal of extracting an exact application model, which can be used for implementation of future systems. And since the model is made for further development, it means that whether in an abstract or concrete form, it contains implementation details which again prevent the domain information from being extracted clearly.

Our research deals with extracting information from existing applications, but not from the source code in general. We think that the more appropriate targets for domain analysis are *graphical user interfaces* (GUIs). We stated three hypotheses. First, *users have direct access to GUIs*. Therefore the programmer is forced to use the domain dictionary in them (domain terms and relations between them). Second, a *GUI describes domain processes in a form of event sequences* that can be performed with it. And third, we confirmed that it is possible to *derive relations between the terms* in the UI semi-automatically. Based on our experiments we assume these three hypotheses to be valid when the target application[1] UI is *made of components*.

TABLE I
Different sources of domain information, existing methods for gaining it and disadvantages of the methods

Information source	Method	Disadvantages
Domain experts	interviews questionnaires forms	• no *formalized* form • *time* consuming • depends on *willingness* of experts • requires a *skilled* domain analyst
Domain documentation	artificial intelligence methods natural language processing modeling	• no *formalized* form • *availability* of data sources • *suitability* of data sources – ambiguity of natural language
Domain applications	automatized analysis of • source codes • or databases	• high level of implementation details • no user access ⇒ programmer is not forced to use domain dictionary

Based on these presumptions we designed a method of automatized domain analysis of user interfaces of software applications which we call *DEAL (Domain Extraction ALgorithm)*. The input of the DEAL method is a GUI of an existing application and the output is a domain model. A DEAL tool **prototype** which implements this method was created and it will be described in this paper. The DEAL tool prototype supports creating a domain model from an existing UI based on its components. First, it creates a so called *component graph* [4], which is a graphical tree-structured graph of all components located in the target UI. Based on the component graph basic information is extracted from target components and basic relations are derived based on target component types. The result is a *term graph* which represents the domain model. A further research is needed to enable deriving more relations. Therefore in this paper we present

[1]In the rest of the paper, an existing application which is an input of the domain analysis, we will call the *target application*. A *target component* is a component located in such an application.

stereotypes of creating GUIs identified in our experiments with the DEAL prototype. They present a template for designing and implementing more deriving procedures in DEAL.

II. State of the art

Here we briefly summarize the approaches which are used for the domain analysis. The domain models (except for DARE) have to be created manually. Then based on them, different outputs are generated (i.e. software product lines, etc.).

The most widely used approach for DA is the **FODA** (Feature Oriented Domain Analysis) approach [5]. FODA aims for analysis of software product lines by comparing the different and similar features. The **DREAM** approach is based on FODA [6]. The approach is similar to FODA, but with the difference of analyzing domain requirements, not features. Many approaches and tools support the FODA method, e.g. Ami Eddi [7], CaptainFeature [8], RequiLine [9] or ASADAL [10]. Domain model created by FODA is used for further generation of a line of software applications (product line).

There are also approaches that do not only support the process of DA, but also the reusability feature by providing a library of reusable components, frameworks or libraries. Such approaches are for example the early **Prieto-Daz** approach [11] or the later **Sherlock** environment [12].

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 provides for example the Czarnecki project Feature Plug-in [13], [14] or his newest effort Clafer [15] and a plug-in FeatureIDE [16], [17].

ToolDay (A Tool for Domain Analysis) [18] is a tool that aims to support all the phases of DA.

All these tools and methodologies support the DA process with different features, but the **input data** for DA (i.e. the information about the domain) always come from the users, or it is not specifed where they come from. Only the **DARE** (Domain analysis and reuse environment) tool from Prieto-Daz [19] primarily aims for automatized collection and structuring of information and creating a reusable library by analysing existing source codes and documentation automatically, but not user interfaces specifically.

Very interesting process is also seen in [20] where authors transform **ontology axioms** into application domain rules which is a reverse process compared to ours.

III. The DEAL tool prototype

The DEAL tool prototype is implemented in Java and at present it enables:

- *Loading* the input (a user interface of an existing application).
- *Processing* the input, the output is a component graph.
- *Generating* the domain model, the output is a term graph.
- *Simplifying* the domain model. The simplification in our case means filtering out unnecessary information unrelated to the domain. The output is a simplified term graph.
- *Registering* recorder. The output is the target application with registered recorder of ui events. This enables the user recording events performed in the UI. In later stages we plan to implement replaying these event sequences, which we experimentally proved to be possible.

The DEAL prototype is published as an open-source project on Assembla on https://www.assembla.com/spaces/DEALtool.

A. Domain model representation

We used a graphical tree-like representation of the domain model inspired by the representation used in the Clafer [15] project which is intended for feature modeling in the FODA notation. In DEAL the domain model is perceived rather as an *ontology*. It uses the same notation as FODA, it has however a different semantics. The relations in DEAL do not represent relations between features (functionalities) of an application, rather they represent relations between the *terms* in the given model as it is in ontologies. We used the FODA notation because it is simple and easily readable. Following relation types are supported by DEAL: Determining the optional and


Model (MODEL)


Grouping (AND)


Mutually exclusive (MUTUALLY_EXCLUSIVE)


Mutually not exclusive (MUTUALLY_NOT_EXCLUSIVE)

mandatory items is not relevant in the case of ontology, we therefore use only one type of node labelling (an empty ring), which represents all types of AND groupings. The MODEL relation is a default relation set for the root of the domain model, it has no special semantics. In DEAL tool, each term is represented by an instance of the *Term* class. It captures information such as name, description, icon (sometimes a term in the interface is represented by a graphical icon, which is self-explanatory *and* domain-specific). It contains a link to the target component which it represents. Each node in the graph has a list of its child Term[2] nodes and a relation type, which applies to the children.

B. The DEAL method

The DEAL method contains the following phases:

(1) **Loading and Processing algorithm** - the input is an existing component-based application with a GUI and the output is a *component graph*. In DEAL the input is a Java application and we use reflection and aspect-oriented programming. The Loading and Processing algorithm is invoked for each activated scene[3]. The Loading and Processing algorithm was completely described in [23] as "the DEAL algorithm".

(2) **Generating of a Domain Model** - the input is the component graph generated in (1), the output is a domain model in a non-simplified form (it contains information not related to domain). For each component in the component graph a new Term is generated. If the component does not contain any relevant information (for example a Container), a Term without any domain-relevant information is generated. This is because of preserving the hierarchy of Terms which is stored in the target application by its programmer.

(3) **Simplification algorithm (SymAlg)** - filtering of information unrelated to the domain, i.e. Terms with

[2]in the rest of this paper, domain terms as real instances (words) will be noted by lower case. Abstract representations of such real terms in the model will be noted by upper case: Term.

[3]We use the term "scene" based on the term by Kösters in [22] A scene represents a window, a dialog or any application frame which contains components.

no domain-relevant information. The input is a domain model from the generator (2) and the output is a simplified domain model. In current state filtering includes removing multiple nesting and removing void containers. A void container is a node with a component of type Container that does not contain any children.

(4) **Algorithms for deriving relations(DerAlg)** - based on the identification of different types of components, relations between Terms in the model are derived. The input is a simplified domain model; the output is a domain model with relations between Terms. In the current state the DEAL tool prototype is able to derive relations from the different types of components (tab panes, check boxes, radio buttons, lists, combo boxes).

These algorithms are called sequentially in the order they are listed here. SymAlg and DerAlg were defined and implemented based on our previous research in [23].

IV. Analysis of stereotypes of creating GUIs using the DEAL tool prototype

In the current state SymAlg and DerAlg algorithms implemented by the DEAL tool prototype contain only the basic procedures. It is necessary to identify stereotypes of creating GUIs and improve SymAlg and DerAlg based on these stereotypes. In the next chapters we will describe some stereotypes of creating GUIs and based on them we will improve the two algorithms if possible.

We already performed an extensive analysis of stereotypes of creating user interfaces in [23]. After developing DEAL we made a series of experiments and identified additional stereotypes. Here we provide the list of the additional stereotypes and based on them we will design and implement new derivation rules in DEAL.

The experiments were performed with these open source Java applications: Java Scientific Calculator, Java NotePad, Jars Browser, jEdit, Guitar Scale Assistant, iAletU. All of them (except iAlertU [4]) are included into the DEAL prototype on Assembla.

The stereotypes were divided into four groups: graphical, functional, logical and custom components.

A. Graphical

The essential feature of a GUI (if we forget the content) is its graphical representation. If the GUI is made in a haphazard fashion, it hardly provides useful information to its users. The GUI elements should be organized in groupings that make sense.

We identified a number of basic stereotypes of positioning components graphically:

a) **Labels describe other components.**

We call this a *label-for* relationship between a label and a target component. It can be derived for example in Java language by reading the *laberFor* attribute of a JLabel component. The identification with this attribute is implemented in the DEAL tool prototype. In HTML language it is possible to read the *for* attribute.

However the programmers often do not specify the label-for attribute therefore a different approach is needed. Deriving the label-for relationship is possible from the graphical layout of elements. Programmers often put labels horizontally aligned with the target component or above it, aligned left. We discussed the problems of deriving the label-for relationship with our approach in [3].

b) **Separators divide groups of items, which are related to each other but which arent related to items that are separated from this group.**

A separator line (in the Java language represented by the Separator class) is used for example in menus or in forms, where it graphically divides related content from its surroundings. Based on this fact we group the terms representing the components in a separated group into an AND group.

c) **Graphical groupings of components.**

The idea is similar to b). If there is a group of buttons close to each other in a calculator app, and another group of buttons divided from the first group by a blank area, then these two groups should be separated also in the domain model.

B. Functional

The functional components are represented by common control items like buttons, menu items, etc. These are often used to provide functionality, which is common for many applications (like OK, Cancel, Reset, Open, Save, Save As..., New, Exit, Close etc.). If these common terms are discarded from the domain model, then only domain-specific terms will remain.

We have implemented a *"hide as common"* functionality into the DEAL tool prototype, which enables hiding a common used term, so it will be invisible in the domain model. These hidden terms will be added to a database that will contain a collection of the most common terms. Based on this database the tool will suggest to hide these terms also in other models. This will ensure the tool will learn.

C. Logical

From some components relations can be logically derived. For example a group of terms representing radio buttons will be mutually exclusive. A group of terms representing check boxes will be not mutually exclusive, but related to each other. The same is true for combo boxes (mutually exclusive) and lists (with a single selection - mutually exclusive, with multiple selection - mutually not exclusive), tab panes (mutually exclusive) etc.

Deriving of these relations is implemented in the DEAL tool prototype.

D. Custom components

Custom components are components programmed by a programmer. These components can serve for gaining domain-specific information.

For example during the experiment with the open-source Launch4j application we encountered the **TitledSeparator** component. It is a normal content separator but it also contains a title. This can be used to describe the group of separated terms. In DEAL new handlers for custom components can be implemented by simply extending the **DomainIdentifiable¡T¿** abstract class, defining the component class as T and by implementing abstract methods for gaining domain information. The handler should also be registered in the list of implemented handlers **domainIdentifiables.properties**. Then the DEAL tool prototype will take also this special type of component into account.

[4] the open source jar for this application does not exist anymore

V. Conclusion

In this paper we introduced our DEAL method for analysing existing software applications for domain terms and for extracting a domain model in a form of a term graph.

We introduced and described our DEAL tool prototype which enables creating domain model from a GUI of an existing Java application. For the future implementation of simplification procedures is planned and implementation of a number of procedures for deriving new relations based on the stereotypes of creating graphical user interfaces which were identified in this paper.

We also plan to implement the functionality for saving the domain model into one of the ontological formats (e.g. OWL) and XML. DEAL supports a recording feature which enables to record the event sequences made by the UI user into a Term sequence. In the final phase we plan to implement a replay feature to replay this sequence on the target application.

Our method and tool can serve as an additional process of domain analysis, where the domain analyst does not start from scratch when creating a domain model, but he can gain a simple domain model from existing application. We plan to apply our research in the field of usability evaluation – specifically domain usability evaluation where it can be utilized in two areas: i) evaluation of domain dictionary of an existing application if it matches the real world; and ii) evaluation of UI event sequences if they are correct and match the domain processes in the real world.

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] J. Neighbors, "Software construction using components," Ph.D. dissertation, University of California, Irvine, 1980.

[2] ——, *The Draco approach to constructing software from reusable components*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1986, pp. 525–535.

[3] M. Bačíková and J. Porubän, "Defining computer languages via user interfaces," Master's thesis, Technical university in Koice, Faculty of Electrotechnical Engineering and Informatics, 2010.

[4] ——, "Automating user actions on gui: Defining a gui domain-specific language," in *CSE 2010: proceedings of International Scientific conference on Computer Science and Engineering*, 2010, pp. 60–67.

[5] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson, "Feature-oriented domain analysis (foda) feasibility study," Carnegie-Mellon University Software Engineering Institute, Tech. Rep., November 1990.

[6] M. Moon, K. Yeom, and H. 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, pp. 551–569, July 2005.

[7] K. Czarnecki, T. Bednasch, P. Unger, and U. W. Eisenecker, "Generative programming for embedded software: An industrial experience report," in *Proceedings of the 1st ACM SIGPLAN/SIGSOFT conference on Generative Programming and Component Engineering*, ser. GPCE '02. London, UK: Springer-Verlag, 2002, pp. 156–172.

[8] "Captainfeature, the webpage of captainfeature sourceforge.net project," https://sourceforge.net/projects/captainfeature, 2005, [Online 2011].

[9] "The webpage of requiline project," https://www-lufgi3-informatik.rwth-aachen.de/TOOLS/requiline/index.php, 2005, [Online 2011].

[10] P. S. E. Laboratory, "A review of asadal case tool."

[11] R. P. Díaz, "Reuse Library Process Model. Final Report," Electronic Systems Division, Air Force Command, USAF, Hanscomb AFB, MA, Technical Report Start Reuse Library Program, 1991.

[12] A. Valerio, G. Succi, and M. Fenaroli, "Domain analysis and framework-based software development," *SIGAPP Appl. Comput. Rev.*, vol. 5, pp. 4–15, September 1997.

[13] K. Czarnecki, M. Antkiewicz, C. Kim, S. Lau, and K. Pietroszek, "fmp and fmp2rsm: eclipse plug-ins for modeling features using model templates," in *Companion to the 20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, ser. OOPSLA '05. New York, NY, USA: ACM, 2005, pp. 200–201.

[14] M. Antkiewicz and K. Czarnecki, "Featureplugin: feature modeling plug-in for eclipse," in *Proceedings of the 2004 OOPSLA workshop on eclipse technology eXchange*, ser. eclipse '04. New York, NY, USA: ACM, 2004, pp. 67–72.

[15] K. Bak, K. Czarnecki, and A. Wasowski, "Feature and meta-models in clafer: mixed, specialized, and coupled," in *Proceedings of the Third international conference on Software language engineering*, ser. SLE'10. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 102–122.

[16] T. Thum, C. Kastner, S. Erdweg, and N. Siegmund, "Abstract Features in Feature Modeling," in *Software Product Line Conference (SPLC), 2011 15th International*. IEEE, Aug. 2011, pp. 191–200. [Online]. Available: http://dx.doi.org/10.1109/SPLC.2011.53

[17] T. Thum, D. Batory, and C. Kastner, "Reasoning about edits to feature models," in *Proceedings of the 31st International Conference on Software Engineering*, ser. ICSE '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 254–264.

[18] L. Lisboa, V. Garcia, E. de Almeida, and S. Meira, "Toolday: a tool for domain analysis," *International Journal on Software Tools for Technology Transfer (STTT)*, vol. 13, pp. 337–353, 2011.

[19] W. Frakes, R. Prieto-Diaz, and C. Fox, "Dare: Domain analysis and reuse environment," *Ann. Softw. Eng.*, vol. 5, pp. 125–141, January 1998.

[20] O. Vasilecas, D. Kalibatiene, and G. Guizzardi, "Towards a formal method for the transformation of ontology axioms to application domain rules," *Information Technology and Control*, vol. 38, no. 4, pp. 271–282, 2009.

[21]

[22] G. Kösters, H.-W. Six, and J. Voss, "Combined analysis of user interface and domain requirements," in *Proceedings of the 2nd International Conference on Requirements Engineering (ICRE '96)*, ser. ICRE '96. Washington, DC, USA: IEEE Computer Society, 1996, pp. 199–. [Online]. Available: http://dl.acm.org/citation.cfm?id=850944.853112

[23] M. Bačíková and J. Porubän, "Analyzing stereotypes of creating graphical user interfaces [unpublished yet]," *Central European Journal of Computer Science*, 2012.