DEAL – a method for Domain Analysis of Graphical User Interfaces

M. BAČÍKOVÁ

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

Abstract. Why are we creating graphical user interfaces (GUIs, UIs)? It is the first point for the user to meet the application. Users remember daily actions that they perform with applications through the UI. But not only through the visual form, but also through the content-procedural form. They remember the sequences of words naming the components they need to use to carry out their task. If the terms in the UI are incorrect, the UI is less understandable and less usable. Under this presumption we designed a method of automatized GUI domain analysis. We verified the method by creating a prototype and we used it to perform experiments on different open-source Java application. In this paper, we present the results of our research.

Keywords
DEAL, graphical user interfaces, domain analysis, usability, domain usability

1. 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 1 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) because they are the layer of first contact with the users. 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 verified the method by creating a prototype and we used it to perform experiments on different open-source Java application. In this paper, we present the results of our research.

<table>
<thead>
<tr>
<th>Information source</th>
<th>Method</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Domain experts</td>
<td>interviews, questionnaires,</td>
<td>- no formalized form</td>
</tr>
<tr>
<td></td>
<td>forms</td>
<td>- time consuming</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- depends on willingness of experts</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- requires a skilled domain analyst</td>
</tr>
<tr>
<td>Domain documentation</td>
<td>artificial intelligence</td>
<td>- no formalized form</td>
</tr>
<tr>
<td></td>
<td>methods, natural language</td>
<td>- availability of data sources</td>
</tr>
<tr>
<td></td>
<td>processing, modeling</td>
<td>- suitability of data sources (ambiguity of natural language)</td>
</tr>
<tr>
<td>Domain applications</td>
<td>automatized analysis of</td>
<td>- high level of implementation details</td>
</tr>
<tr>
<td></td>
<td>- source codes, or databases</td>
<td>- no user access</td>
</tr>
<tr>
<td></td>
<td></td>
<td>=&gt; programmer is not forced to use domain dictionary</td>
</tr>
</tbody>
</table>

Tab. 1. Different sources of domain information, existing methods for gaining it and disadvantages of the methods.
assumed these three hypotheses to be valid when the target application’s UI is made of components.

Based on these presumptions we designed a method of automatized domain analysis of user interfaces of software applications which we call DEAL (Domain Extraction ALogrithm). 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, 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. In this paper we present the DEAL method and prototype and the experiments and results performed on several open-source Java applications.

Our method and tool can serve as an additional process of domain analysis, where the domain analyst would not start from scratch when creating a domain model, but he or she can gain a simple domain model from an existing obsolete application.

2. The DEAL method

The input of DEAL is an existing graphical user interface. As already mentioned, the input GUI has to be made of components. The output of DEAL is a domain model created from the GUI. DEAL has 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. The Loading and Processing algorithm was completely described in [4] 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 no domain-relevant information. The input is a domain model from the generator (2) and the output is a simplified domain model. Filtering involves 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 simplified domain model with relations between Terms. The relations are derived from the different types of components (tab panes, check boxes, radio buttons, lists, combo boxes, separators, etc.).

These algorithms are to be called sequentially in the order they are listed here. SymAlg and DerAlg were defined and implemented based on our previous research in [4].

2.1. Domain model representation

We used a graphical graph representation of the domain model inspired by the representation used in the Clafer [6] 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:

- Model (MODEL)
- Grouping (AND)
- Mutually exclusive (MUTUALLY_EXCLUSIVE)
- Mutually not exclusive (MUTUALLY_NOT_EXCLUSIVE)

Determining the optional and 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 nodes and a relation type, which applies to the children.

The generated domain model can serve for different tasks. The first is generating a domain dictionary or an ontology which can be used in further processes. It can also be used for evaluating domain usability. Least but not least, the generative processes could benefit from a domain model created from an obsolete application to be able to generate a new UI or a completely new application using the specification in a form of a domain model. Another area is an automatic generation of user manuals similar to the Epipance XYDOCS software.

3. 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 Assembla project on https://www.assembla.com/spaces/DEALtool.

4. The User Interfaces are Created with a Purpose

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 [4]. 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 (all these are included in the DEAL prototype example directory on Assembla), jarsBrowser, GloboNote, SweetHome3D, TimeSlotTracker, Torrent Episode Downloader, PdfSam, jSignPdf, Finanx.

We divided the stereotypes into four groups: graphical, functional, logical and custom components.

4.1. 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 labelFor 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 aren’t 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.
4.2. 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.

4.3. 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.

4.4. 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 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. Then the DEAL tool prototype will take also this special type of component into account. For now, we created 4 handlers for custom components and we are preparing more of them.

5. Further Reading

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 [8]. FODA aims for analysis of software product lines by comparing the different and similar features. The DREAM approach is based on FODA [9]. 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 [10], CaptainFeature [11], RequiLine [12] or ASADAL [13]. 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 [14] or the later Sherlock environment [15].

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 [16] [17] or his newest effort Clafer [6] and a plug-in FeatureIDE [18] [19].

ToolDay (A Tool for Domain Analysis) [20] 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 specified where they come from. Only the DARE (Domain analysis and reuse environment) tool from Prieto-Daz [21] 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 [22] where authors transform ontology axioms into application domain rules which is a reverse process compared to ours.

A business solution similar to ours, only on a higher level of a specific process of generating user manuals, is the Epiance XYDOCS [7] software. XYDOCS is a software for automatic generation of user manuals from existing user interfaces related to a newly created enterprise application. It supports different platforms such as Windows, .NET, Green Screen, Java and HTML.
6. Conclusion

In this paper, we presented the DEAL method for analysing graphical user interfaces. The result of DEAL is a domain model created from the content of the user interface given as the input. The DEAL method can only be used on component-based applications. We created a prototype in Java language using reflection and aspect-oriented programming. However there are still some issues with custom components and errors in applications that prevent the domain analysis to be performed. Based on these results we will improve our method and the prototype. In the future, we plan to expand the prototype to be able to analyse web applications.

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.

Acknowledgements

Research described in the paper was supervised by Associate Prof. J. Porubán, FEI TUKE in Košice and supported by VEGA Grant No. 1/0305/11 ?. Co-evolution of the artifacts written in domain-specific languages driven by language evolution.

References


