Domain Analysis of Graphical User Interfaces of Software Systems

Michaela Bačíková[*]
Department of Computers and Informatics
Faculty of Electrotechnical Engineering and Informatics
Technical University of Košice
Letná 9, 042 00 Košice, Slovakia
michaela.bacikova@tuke.sk

Abstract

Domain analysis (DA) is the first phase of software systen development. A person who performs DA is called a domain analyst. The task of a domain analyst is to collect information about a domain with the goal of developing a new software system. Usually, this task is manual and time-tedious, because the data have to be manually gathered from different sources such as existing documents, domain experts or existing applications in the domain.

This work is a contribution in the field of domain analysis of software systems. Our goal is to reduce the time and effort of domain analysts by supporting an automated creation of domain models. The goal is achieved through domain-specific language (DSL) development, thus we also contribute to the field of DSL development.

We present a unique method for domain analysis of graphical user interfaces (GUIs) of existing software systems. The output of the method is a formal, platform-independent domain model, which is then used to generate a DSL implementation - classes representing the language model and a language parser. Since the domain model is platform-independent, it can be utilized also in other processes. These processes are described in the work and their utilization is demonstrated in the experimental part.

We developed a prototype to experimentally verify the proposed method. Besides the primary feature of creating DSLs from existing GUIs, the prototype also supports formalization of the domain model into ontologies. The prototype was used in three experiments: to experimentally verify the method on 32 existing applications, to develop 9 DSLs and to compare two systems using ontologies.

Categories and Subject Descriptors

D.2.13 [**Software Engineering**]: Reusable Software—*Domain engineering*; D.3.m [**Programming Languages**]: Miscellaneous; H.5.2 [**Information Interfaces and Presentation (e.g., HCI)**]: User Interfaces—*Evaluation/methodology, Graphical user interfaces (GUI), Natural language, Theory and methods*; D.2.11 [**Software Engineering**]: Software architectures—*Domain-specific architectures, Languages*

Keywords

domain analysis, domain-specific languages, graphical user interfaces, component-based software systems, reverse engineering, ontologies

1. Introduction

When developing a new software system (or a new version of a system) for a particular domain (e.g. banking), the first step of the developer is to speak with the *domain experts* (bank employees), study documents to gain knowledge about the domain and find and analyse existing banking systems (or the old version of the banking system). Then a form of a formal *domain model* is created, describing the knowledge from the banking domain, which is used to develop the new system.

This process is called *domain analysis (DA)*. The person who performs DA is called *domain analyst.* The task is usually manual and often tedious and time consuming. This work aims to reduce the time and effort of the domain analyst by supporting the creation of a formalized domain model in an automated manner.

Why extract domain information from GUIs and not for example from domain experts or domain documents? GUIs are made for domain users, therefore we can assume, they will contain correct domain terms and relations. Otherwise, they would be less usable. Moreover, unlike the knowledge of a domain expert or information stored in domain documents, GUIs are in a formalized form. Taking this facts into account, we can say that GUIs are the best sources of formalized domain knowledge.

The idea goes even further and we propose a method for creating new *domain-specific languages (DSLs)* based on the previously extracted domain models. Thus, we contribute to the DSL development also.

[*]Recommended by thesis supervisor: Prof. Jaroslav Porubän
Defended at Faculty of Electrotechnical Engineering and Informatics, Technical University of Košice on June 30, 2014.

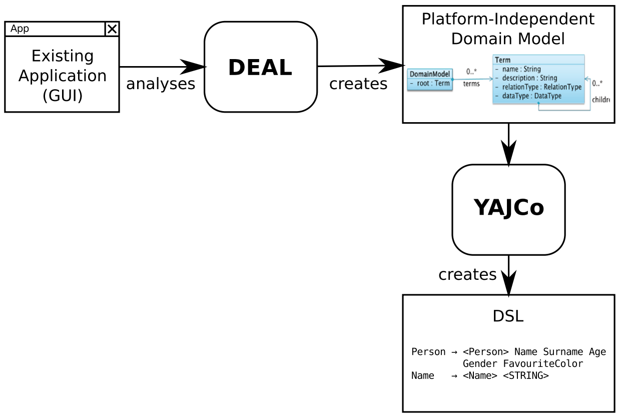
DSLs are small programming languages that provide suitable notations and abstractions for achieving expressive power specialized, and usually restricted, to a domain of problems [1, 2]. By having a DSL, the developer of a new system can benefit from the architecture with a separated domain model. Besides this advantage, DSLs further provide benefits such as easier programming and easier understandability for end-users. Although the new system will use different technology or have a different appearance, because of reusing the DSL from the old system, the terms in the UI will be the same and therefore it will be more user-friendly.

Because the development of tools for supporting the DSL design is often inevitable, the development of a DSL can be of a high cost. Zawoad et al. provide a comprehensible example of creating their own DSL for secure logging [3]. From their work it is evident, that a deep understanding of the DSL creation process is needed to create new DSLs and their supporting tools. Zawoad et al. perform 5 standard phases of creating DSLs: (1) domain analysis, (2) definition of the abstract syntax, (3) definition of the concrete syntax, (4) definition of the translational semantics, (5) implementation. While the last four phases are well supported and automated by many researchers [4], the DA phase is usually performed manually, as it is also in the case of Zawoad et al. We aim to support the DA phase by designing a method for automatized creation of a DSL from existing sources. By providing an automatic generation of a language processor for the generated DSL, we also support the next four phases of the DSL development. Although the development phase is pretty well supported, we integrate existing solutions into our approach and thus provide a comprehensive solution, which supports all phases at once.

We have designed and developed the aforementioned method for domain language extraction. The method is a combination of two steps. In the first step, our DEAL method analyses UIs and creates a platform-independent domain model. The second step is a translation of the domain model to a language specification. We have implemented a prototype of the method – the DEAL tool. As a language specification we use grammar description language supported by annotation-based model-driven parser generator YAJCo [5]. YAJCo is able to generate a language model specification using annotated Java classes. In our case, however, we create the language specification by traversing the domain model and creating language concepts, abstract and concrete syntax in our algorithm. Based on the language specification, YAJCo generates a parser, that can be used as a text-based UI of the application (this could be interesting for testing the non-UI part of the application). Both tools, DEAL and YAJCo were developed at our university and in the following sections we will shortly describe their principles. The concept of the method is displayed in Fig. 1.

2. The DEAL Method

In this section we will describe our DEAL method for domain analysis and DSL development. We will also describe its implementation, the DEAL tool. To explain the process, how the DSLs are created from domain models extracted by the DEAL method we will present the process on a small example of a Person form and we will also shortly describe the YAJCo tool that is used to generate language model classes and parser.



```
Person → <Person> Name Surname Age
         Gender FavouriteColor
Name   → <Name> <STRING>
```

Figure 1: The concept of the DEAL method for domain language extraction

Our method for domain analysis is called *DEAL (Domain Extraction ALgorithm)*. The input of the DEAL method is an existing UI made of components and the output is a platform-independent domain model.

The target UIs to be analysed have to fulfil the following conditions: (i) the UI has to be made of *components*, (ii) the UI programming platform has to enable some form of *introspection*. If the conditions (i) and (ii) are valid, then we can perceive graphical components as domain-specific units that represent the terms of the UI language. The scenes (i.e. windows, dialogs, web pages, etc.) of a UI define the domain or sub-domain and terms represented by all the components located in a particular scene belong to the domain or sub-domain defined by the scene. The components represent domain-specific units of the domain defined by the scene.

The DEAL method uses two kinds of metamodels: application UI metamodel and domain metamodel. The *application UI metamodel* is a model of an application's UI and it contains scenes represented as hierarchies of containers and components and their domain identifiers. It defines the input of the DEAL method, i.e. a UI. The *domain metamodel defines* serves for specifying domain terms, their hierarchy, their properties, constraints and relations between the terms. The domain metamodel defines the output of the DEAL method.

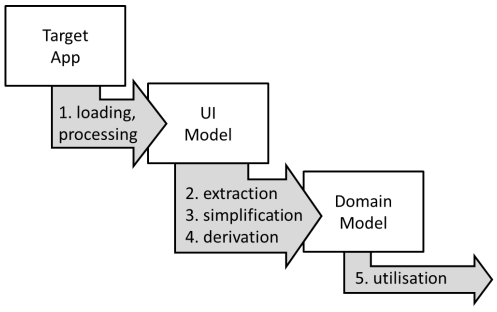


Figure 2: Phases of the DEAL method

The DEAL method has five phases as displayed in Fig. 2:

1. Loading and processing
2. Extraction
3. Simplification

4. Derivation
5. Utilization

In the *loading and processing phase* the target application is loaded into memory and all its scenes and components are traversed and processed. The loading and processing phase is application-specific and the UI programming platform of the target application has to provide means of introspection for this phase to be successful. *Application-specific* means that a loading and processing algorithm has to be created for every new programming language and the programming platform of the target application's UI. The output of the loading and processing phase is an application UI model.

The *extraction phase* is component-specific. Component-specific means that a handler has to be created for each component type and UI programming platform. The output of the extraction phase is an object representation of the domain model, which contains terms, their explicit properties and constraints.

Simplification means filtering information unrelated to the domain. The domain model created in the extraction phase contains data unimportant for the domain model, such as: empty terms (terms without any relevant domain information, created mostly from containers), general terms unrelated to the domain (such as File, Save, Import, Exit, etc.). Filtering involves removing multiple nesting, removing empty terms, shifting single leafs and filtering general non-domain terms. In this context, an empty term is a term created from a component of type container that does not contain any children.

Some empty terms are essential for preserving the hierarchy of terms. However if the target application contains too many containers, it causes excessive deepening of the domain model hierarchy, which is not desirable. Therefore such terms have to be removed. The output of the simplification phase is a simplified domain model.

Derivation is a process where implicit relations are derived. Derivation is defined in component handlers and is based on the identification of different component types. The output is a simplified domain model with implicit relations between terms. DEAL supports three kinds of relations: *and*, *mutually-exclusive* and *mutually-not-exclusive* relation.

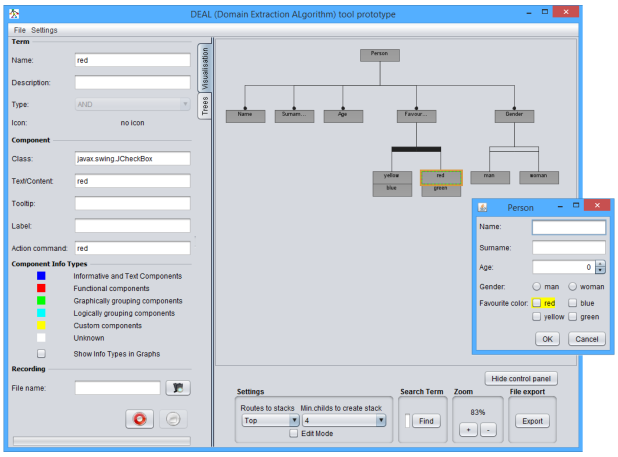
In the *utilization phase* the model created in the previous phase can, for example, be utilized in the following processes:

- generating a DSL (described in [6]),
- generating an ontology (described and experimentally verified in [7]),
- generating new UI (described and experimentally verified in [8],
- evaluating domain usability [9], etc.

Each of the 5 phases are to be performed sequentially in the order they are listed here. Each phase uses the output of the previous phase. Once the domain model is extracted, it can be processed independently. Thus the simplification, derivation and utilisation phases are not necessarily application-specific: they are the same for any target application.

2.1 The DEAL Tool

The DEAL tool[1] is a software solution for extracting domain knowledge from existing UIs and it proves the possibility of using the DEAL method on Java and Windows applications. The prototype is written in Java language and it uses Java reflection, DOM and AspectJ to be able to extract domain information from target applications. The main window of the DEAL tool is displayed in Figure 3. It allows to review domain models extracted from running application. The functionality is interconnected with the target application: when a user clicks on a term in the extracted domain model, then the corresponding component is highlighted in the target application, too. Some of the extracted concepts can be excluded from the model using a "Hide" function.


DEAL (Domain Extraction ALgorithm) tool prototype

Term
Name: red
Description:
Type: AND
Icon: no icon

Component
Class: javax.swing.JCheckBox
Text/Content: red
Tooltip:
Label:
Action command: red

Component Info Types
🟦 Informative and Text Components
🟥 Functional components
🟩 Graphically grouping components
🟦 Logically grouping components
🟨 Custom components
⬜ Unknown
☐ Show Info Types in Graphs

Recording
File name:

Settings
Routes to stacks: Top
Min.childs to create stack: 4
☐ Edit Mode
Zoom: 83%

Person
Name:
Surname:
Age: 0
Gender: ○ man ○ woman
Favourite color: ☐ red ☐ blue ☐ yellow ☐ green

Figure 3: The DEAL tool in the back and a Person form in the front.

The DEAL tool proves that it is possible to:

- traverse a UI of an existing application, given that the application is made of components and enables introspection,
- extract domain information from the UI in a formalized form, and to generate a DSL based on the extracted domain model.

The resulting domain model is in a form of an internal object model.

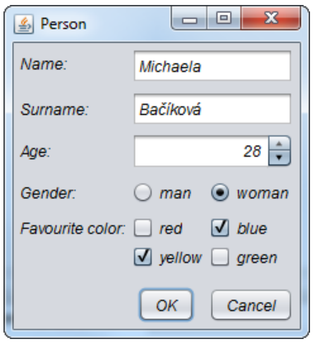
2.2 DSL as Output of the DEAL Method

For better understanding, we will present the result of extraction on a small DSL extracted from a Person form. The form can be seen in Fig. 4.

Domain model of the form was extracted using the DEAL tool. This model was transformed into a language model used by YAJCo tool described in section 2.3.

The concrete syntax of the Person form language can be described using a grammar in EBNF. The elements ⟨STRING⟩ and ⟨NUMBER⟩ represent a terminal string

[1]DEAL project can be found at: https://www.assembla.com/spaces/DEALtool



Name:	Michaela	
Surname:	Bačíková	
Age:	28	
Gender:	man	[✓ woman]
Favourite color:	red	[✓ blue]
	[✓ yellow]	green

Figure 4: The Person form

or numeric values. The concrete syntax of the DSL generated from the Person form is as follows (terminals are noted in quotation marks):

$$\begin{aligned} Person &\rightarrow "Person" Name\ Surname\ Age \\ &\quad Gender\ FavouriteColor \\ Name &\rightarrow "Name"\ \langle \text{STRING} \rangle \\ Surname &\rightarrow "Surname"\ \langle \text{STRING} \rangle \\ Age &\rightarrow "Age"\ \langle \text{NUMBER} \rangle \\ Gender &\rightarrow "Gender"\ ("man"\ |\ "woman") \\ FavouriteColor &\rightarrow "Favourite\ colors" \\ &\quad ("red"?\ "blue"?\ "green"?\ "yellow"?) \end{aligned}$$

In this *domain model to DSL specification* transformation, each term of the domain model is transformed into one concept in the language model. That means, for each term there is one rule in the EBNF. The domain model is represented as a hierarchy of terms. This hierarchy is directly mapped into the hierarchy of language concepts.

Moreover, if a term in the domain model has a *not-mutually-exclusive* relation, then this term is transformed into a rule of 0-1 choices (similar to the *FavouriteColor* rule) and in Java class it is represented as a list of items. On the other hand, if the term has a *mutually-exclusive* relation, then it is transformed into a rule of alternatives (similar to the *Gender* rule) and in Java class it is represented as enumeration.

We also map data types into the language model. Note that since the *Age* field in the graphical form is a numeric spinner in the UI, the data type of the Age property was derived as ⟨NUMBER⟩ type. And since both *Name* and *Surname* came from textfield components, the data type of the Name and Surname properties was derived as ⟨STRING⟩ . For more information about the *domain model to DSL specification* transformation, see our paper [6].

2.3 YAJCo Parser Generator

YAJCo[2] is a parser generator that accepts language specification in a form of Java classes representing language concepts. Relations between the concepts (and therefore grammar structure) are inferred from the relations between classes such as inheritance, composition and aggregation. Concrete syntax of the language and some additional information is specified using Java annotations. Based on such specification YAJCo can generate language grammar and parser. YAJCo also generates the parser using JavaCC or Beaver that can instantiate the model based on a DSL sentence. More about YAJCo can be found in [5].

[2]YAJCo project can be found at: https://code.google.com/p/yajco/

YAJCo can also be used the other way round and generate object oriented model in a form of Java classes from the internal language model. This mode of operation was used to transform domain model extracted by DEAL to both language grammar and classes representing the language model as displayed in Fig. 1. This way the extracted model can be directly used within the new version of the application.

In our example of the Person form language, several classes were generated as shown by class diagram in Figure 5. For each rule in the EBNF grammar there is one class and for each data type in the rule (number, string, date, etc.) in the rule there is one property in the class, e.g. a class *Person* has a *name*, *surname* and *age*. Each of them represents object of classes *Name*, *Surname* and *Age* where the first two have inner values of type String and *Age* has an inner value of type int. *Person* also has a List of *FavouriteColor* instances and a reference to a *Gender* enum.

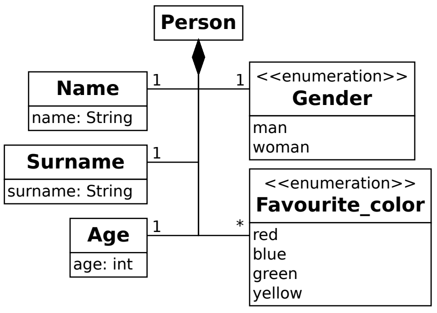


Figure 5: Class diagram of the Person language model.

2.4 Utilization of the DEAL method: Domain Usability

Since the domain model generated by the DEAL method is platform-independent, it can be used also in further processes other than DSL development. *Ontologies* are one of the possible output. Ontologies are formal representations of knowledge as hierarchies of concepts within a domain, using a shared vocabulary to denote the types, properties and interrelationships of those concepts. Ontologies are considered pillars of the Semantic web, where they are used to define semantics of web pages. One of the standard format for ontology notation is OWL.

We developed a method for creating an ontology from the domain model extracted by the DEAL method and we described it in [7]. The output ontology is in the standard OWL format and describes the knowledge extracted from the existing application. Thus, we also support a formalization of existing GUIs into ontologies.

The goal of ontology extraction is not only to aid Semantic web. Because of the OWL standard, we can benefit from ontological methodologies and tools. For example, we can compare two ontologies or query an ontology for facts we need to know. Ontology comparation or querying can be used to evaluate *usability* of the target user interface, by comparing it to another user interface (its ontology) or to an ontological dictionary such as WordNet for correctness. We call this aspect of the system *domain usability.* To define domain usability we have to explain the general usability first.

The common perception of usability is usually in the terms of user experience, satisfaction, application quality and effectiveness or ergonomics. Often it is seen from the *ergonomic* point of view and the domain aspect is neglected or omitted, even if it is included' in the general definition by Nielsen [10], which still serves as a fundamental guide to create usable software systems, new usability guidelines and usability evaluation and testing systems.

Each software is developed for a concrete domain, therefore its UI should contain *terms, relations* and describe *processes* from its specific domain for the users to be able to work with it. If the user does not *understand* the terms in the system's UI, then the whole application is less usable. Based on our experience and research and pursuing the existing current work in the area of usability, we defined *understandability* as the property of a system, that affects usability and relates to the following factors:

- *Domain content:* the UI terms, relations and processes should match the ones from the domain, which the UI is designed for.
- *Adequate level of specificity:* the UI made for a specific domain should not contain terms too general, even if they belong to a parent domain. On the other hand, the terms should not bee too specific, if the system is used by people from a more general domain.
- *Consistency:* words used throughout the whole UI should not differ, if they describe the same functionality, the dictionary shoulxd be consistent.
- *Language barriers:* there should be no foreign words, the translation should be complete. The language of the UI should be the language of the user.
- *Errors:* a UI should not contain stylistic and grammatical errors.

The *domain usability* is defined as the aspect of usability, which is affected by the factor of UI understandability. However, it is not true that understandability = domain usability. Understandability can affect other attributes besides domain usability, e.g. accessibility. The overall usability is defined as a connection of two basic aspects: ergonomic usability and domain usability. These two aspects can be combined together when evaluating usability.

Using our approach, and using the proposed ontology formalization, it is possible to automatically evaluate domain usability as described in the introduction of this section. The feasibility analysis of automated evaluation of domain usability is described in [9].

3. Experimental verification

We performed three experiments. The goal of the first experiment was to verify the DEAL method on existing Java applications. More specifically it verifies the coverage of the DEAL default handlers for components. The second experiment is a verification of the DSL extraction module. The third experiment verifies the ontology extraction module and moreover, it uses the ontologies generated by the module to compare two existing applications. We will describe each experiment in the next sections[3].

Experiment 1: Coverage of the DEAL Default Handlers

We verified the DEAL prototype against 32 open-source Java applications downloaded from sourceforge. The number of 799 components were counted in all 32 applications and 761 of them were successfully extracted. Considering these numbers, the DEAL prototype has a *95% coverage.*

For poorly designed applications, such as *ArcaneAvalon*, where the content was drawn directly in the paint() method of the application frame, the content was not extracted at all.

On the other hand, there were examples of graphical applications such as *FreeMind* and *Freeplane* with very good designs and thus not only standard but also all the application-specific components have been extracted properly from them.

Experiment 2: DSLs of Domain-specific Applications

Using the DEAL method we were able to infer grammars and parsers for 9 domain-specific languages such as: the language of locations and time zones, the language of furniture categories and furniture, the language of egyptian hieroglyphic symbols, the language of star observations and the language of bibliographical references.

The applications, which the DSLs were extracted from, are: PersonForm (a simple testing application), TimeSlotTracker, DaylightChart, JSesh, SweetHome3D, VStar, JabRef (all downloaded from *sourceforge.net*).

All DSLs in our experiment were generated correctly, hence achieving our goal[4].

Experiment 3: Comparison of two Applications

The goal of this experiment was to answer the question whether it is possible to compare two existing GUIs based on their domain model and to what extent. For ontology extraction, we used the DEAL module for generating ontologies and for automatic comparison we used the Protégé ontology tool.

Two ontologies were generated from two applications: *Freeplane* and *FreeMind*, both downloaded from sourceforge. The result of the comparison of ontologies created from the applications are the following: 1. entities created: 678, 2. entities deleted: 166, 3. entities modified: 345. The results are not 100% accurate, because the components with different names, but same functionality in both applications, were counted as created/deleted entities, not as modified. However some changes can be identified with certainty, e.g. the menu item position changes can be identified in the Superclass parameter in Protégé or the

[3]A more detailed description and the results of the experiments can be found online in the DEAL wiki page: https://www.assembla.com/spaces/DEALtool/wiki/Experimental_verification

[4]The DSLs can also be found online in a Dropbox folder at http://goo.gl/PxaFy7

change of the menu item class.

When finding new features, combination of a visual (manual) comparison and automatic ontology comparison can be better than visual-only. Combination of the visual and ontology comparison with highlighting would highly enhance the illustration of the comparison.

4. Related Work

In this section, we will select the most important works, related to our approach.

As proposed by Čeh et al. [11], it is possible to use *ontologies* as *sources* of DSL terminology. They have implemented the Ontology2DSL framework to demonstrate the ontology to DSL transformation. The condition for using their approach is the existence of an ontology designed for the given specific domain. Which is actually an equally difficult problem compared to finding an existing DSL for the given specific domain - the amount of existing software systems is certainly larger than the amount of existing ontologies, therefore we claim that our approach for creating DSLs is more advantageous.

Several approaches exist that deal with automatized DA in the area of *feature modeling.* All mentioned approaches deal with automatized extraction of *feature models* from various sources and with subsequent processing of the extracted information. A feature model is a specific kind of domain model that uses features to describe a domain application. Czarneczki et al. [13] propose mining techniques that extracts feature models from a set of multiple product configurations, however the assumption is that the products are already formally described as sets of features or configurations. However it would be possible to combine the DEAL method with their technique. Davril et al. [14] assume that an organization has no existing product descriptions and must rely on publicly available data from websites, which provide feature lists of products. Weston et al. [15] and Chen et al. [12] propose a method of extracting feature models from informal specifications. The resulting feature model is constrained to the set of features described in the specifications for the existing set of products.

An approach very similar to ours is the diploma thesis of Jiří Sotona [16]. Sotona created a tool called HTML Extract which is able to automatically extract textual information from web GUIs. Similar approach, targeted to mobile device readability, was presented by Buyukkokten et al. [17] who introduced methods for summarizing parts of Web pages in handheld devices. Buyukkokten et al. aimed at enhancing web readability and orientation for mobile web readers. Šváb et al. [18] perform extraction of specific statistical information using the Hidden Markov model technique and the output of the extraction is data in an RDF format. They extract structured semantic information, however their work benefits from targeting the extraction process on specific data such as product catalogues while our approach is aimed at GUIs in general. Bronzi et al. in [19] present their approach of harvesting data exposed by a set of structured and partially overlapping data-intensive web sources. Bronzi et al. use multiple web sites to extract domain-specific information based on the overlapping data. The result is a table of domain data. The approach is quite similar to our approach, however we also create DSLs from the GUIs, i.e. our motivation is to develop new software systems using the DSL. Mazal [20] tries to create an XML with data extracted from existing HTML page. The XML contains textual data from the page, page title and date. Mazal however does not extract any data properties or derives relations between the textual items. Each of the listed approaches deals with data extraction from web pages in the first place, but none of them deals with creating a domain model from the extracted data. Unlike our approach, none of the listed approaches had defined a methodology for extracting domain models along with properties and relations from existing GUIs.

5. Conclusions

The work describes a new unique method and approach for DSL development. Its main contributions include:

- Proposal of an original method of domain analysis of graphical user interfaces.
- Proposal of a unique approach to developing DSLs.
- Proposal of a new definition of domain usability.
- A feasibility analysis of automatic evaluation of domain usability.
- Experimental verification of the proposed method.

The designed approach for domain analysis of GUIs and creating DSLs can be further developed in several fields, including:

- Evaluation of domain usability. We are convinced that the extracted domain models can be used not only for visual evaluation of domain usability by a developer or domain expert, but also automatically by comparing the extracted domain model with an existing ontology or ontological dictionary, that contain pre-defined relations between the terms. This field represents our current and future research.
- Analysis of UI event sequences. In our definition of domain usability we also touch the sequences of events in UIs and we claim that sequences of events should correspond to the real-world processes. Current approaches for analysing UI event sequences focus on analysing procedural errors and bugs, however our view on the event sequences is more as domain processes.
- Generative processes, e.g. a generation of user guides as an example with using the above mentioned UI event sequences recorded on a UI. A tool for recording such event sequences could be used to generate a draft of a user guide along with the screenshots.
- Combination of the DEAL approach with other approaches for domain analysis, such as clustering or for example, the Davril's approach [14] for extracting domain information from product descriptions from public websites of existing software applications.

5.1 Acknowledgments

This work was supported by project VEGA 1/0341/13 “Principles and methods of automated abstraction of computer languages and software development based on the semantic enrichment caused by communication”.

References

[1] M. Mernik, J. Heering, and A. M. Sloane, “When and how to develop domain-specific languages,” *ACM Comput. Surv.*, vol. 37, no. 4, pp. 316–344, Dec. 2005.

[2] M. Fowler, *Domain-Specific Languages*, 1st ed. Addison-Wesley Professional, Oct. 2010.

[3] S. Zawoad, M. Mernik, and R. Hasan, “Fal: A forensics aware language for secure logging,” in *Computer Science and Information Systems (FedCSIS), 2013 Federated Conference on*, Sept 2013, pp. 1579–1586.

[4] T. Kosar, P. E. M. López, P. A. Barrientos, and M. Mernik, “A preliminary study on various implementation approaches of domain-specific language,” *Inf. Softw. Technol.*, vol. 50, no. 5, pp. 390–405, Apr. 2008.

[5] J. Porubän, F. M., M. Sabo, and M. Běhalek, “Annotation based parser generator,” *Computer Science and Information Systems : Special Issue on Advances in Languages, Related Technologies and Applications*, 2010.

[6] M. Bačíková, J. Porubän, and D. Lakatoš, “Defining domain language of graphical user interfaces,” in *SLATE*, 2013, pp. 187–202.

[7] M. Bačíková, Š. Nitkulinec, “Formalization of graphical user interfaces using ontologies,” in *POSTER 2014 : 18th International Student Conference on Electrical Engineering*, vol. 15. Czech Technical University in Prague, May 2014, pp. 1–5. [Online]. Available: http://hornad.fei.tuke.sk/ bacikova/publications/2014–04_poster14_bacikova–nitkulinec.pdf

[8] M. Bačíková and J. Porubän, “Ui creation patterns (using itasks for dsl -> gui transformation,” in *Proceedings of the Twelfth International Conference on Informatics, INFORMATICS’2013*, Spišská Nová Ves, Slovakia, 2013, pp. 145–150.

[9] ——, “Ergonomic vs. domain usability of user interfaces,” in *Human System Interaction (HSI), 2013 The 6th International Conference on*, June 2013, pp. 159–166.

[10] J. Nielsen, *Usability Engineering*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993.

[11] I. Čeh, M. Crepinsek, T. Kosar, and M. Mernik, “Ontology driven development of domain-specific languages,” *Computer Science and Information Systems*, no. 2, pp. 317–342, 2011.

[12] K. Chen, W. Zhang, H. Zhao, H. Mei, “An Approach to Constructing Feature Models Based on Requirements Clustering,” *RE ’05: Proceedings of the 13th IEEE International Conference on Requirements Engineering*, pp. 31–40, IEEE Computer Society, Washington, DC, USA, 2005.

[13] K. Czarnecki, S. She, and A. Wasowski, “Sample spaces and feature models: There and back again,” *Software Product Line Conference, International*, vol. 0, pp. 22–31, 2008.

[14] J.-M. Davril, E. Delfosse, N. Hariri, M. Acher, J. Cleland-Huang, and P. Heymans, “Feature model extraction from large collections of informal product descriptions,” in *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*, ser. ESEC/FSE 2013. New York, NY, USA: ACM, 2013, pp. 290–300.

[15] N. Weston, R. Chitchyan, and A. Rashid, “A framework for constructing semantically composable feature models from natural language requirements,” in *Proceedings of the 13th International Software Product Line Conference*, ser. SPLC ’09, Pittsburgh, PA, USA, 2009, pp. 211–220.

[16] J. SOTONA, “Hypertext data preprocessing for e-learning,” master’s thesis, Masaryk’s university, Faculty of informatics, 2007 [cit. 2014-02-17]. [Online]. Available: http://is.muni.cz/th/60464/fi_m/

[17] O. Buyukkokten, H. Garcia-Molina, and A. Paepcke, “Seeing the whole in parts: Text summarization for web browsing on handheld devices,” in *Proceedings of the 10th International Conference on World Wide Web*, ser. WWW ’01. New York, NY, USA: ACM, 2001, pp. 652–662.

[18] O. Svab, M. Labsky, and V. Svatek, “Rdf-based retrieval of information extracted from web product catalogues,” 2004.

[19] M. Bronzi, V. Crescenzi, P. Merialdo, and P. Papotti, “Extraction and integration of partially overlapping web sources,” *Proc. VLDB Endow.*, vol. 6, no. 10, pp. 805–816, Aug. 2013.

[20] J. Mazal, “Extraction of textual data from web pages,” master’s thesis, Brno University of Technology, 2011 [cit. 2014-02-19]. [Online]. Available: https://www.vutbr.cz/www_base/zav_prace_soubor _verejne.php?file_id=40352

Selected Papers by the Author

M. Bačíková, J. Porubän Evaluating Domain Usability with DEAL: a Case Study. In Journal of Computer Science and Control Systems: JCSCS Vol. 6, no. 1(2013), pages 5–9. ISSN 1844-6043. 2013.

M. Bačíková, J. Porubän Analyzing stereotypes of creating graphical user interfaces In Central European Journal of Computer Science. Vol. 2, no. 3 (2012), pages 300–315. ISSN 1896-1533. 2012. Springer.

M. Kreutzová, J. Porubän, P. Václavík First Step for GUI Domain Analysis : Formalization In Journal of Computer Science and Control Systems. Vol. 4, no. 1 (2011), pages 65-70. ISSN 1844-6043. 2011.

M. Bačíková, J. Porubän Ergonomic vs. Domain Usability of User Interfaces In HSI 2013 : 6th International Conference on Human System Interaction : June 6.–8. 2013, Sopot, Poland, pages 1–8. ISBN 978-1-4673-5636-7. 2013. IEEE. Selected to monograph – waiting for publisher review.

M. Bačíková, J. Porubän, D. Lakatoš Defining Domain Language of Graphical User Interfaces In SLATE 2013: Symposium on Languages, Applications and Technologies : proceedings : June 20-21 2013, Porto, Portugal, pages 187–202. ISBN: 978-3-939897-52-1, ISSN: 2190-6807. 2013.

M. Bačíková, D. Lakatoš, M. Nosál’ Automatized generating of GUIs for domain-specific languages In SLEDS 2012: Doctoral Symposium of the 5th International Conference on Software Language Engineering 2012 : proceedings : Dresden, Germany, September 25, 2012, Dresden, Germany. - Leipzig : Universty of Leipzig, 2012, pages 1–9. ISSN 1613-0073 [Online] http://ceur-ws.org/Vol-935/.

M. Bačíková, J. Porubän, D. Lakatoš Declarative Specification of References in DSLs In Federated Conference on Computer Science and Information Systems, FedCSIS 2013. 8-11.9.2013: Krakow Poland, pages 1527–1534. Los Alamitos, CA, USA : IEEE Computer Society Press, 2013, acceptance rate 40%.

M. Sabo ... [et al.] Computer Language Notation Specification through Program Examples In FedCSIS: Proceedings of the Federated Conference on Computer Science and Information Systems. September 18-21, 2011, Szczecin, Poland. Los Alamitos, pages 895–898. ISBN 978-83-60810-22-4. IEEE Computer Society Press, 2011.

M. Bačíková Formalization of Graphical User Interfaces using Ontologies In POSTER 2014 : 18th International Student Conference on Electrical Engineering : May 15, 2014, Prague, Czech republic. Czech Technical University in Prague, 2014, pages 1–5.

M. Bačíková DEAL – a method for Domain Analysis of Graphical User Interfaces In Poster 2013 : 17th International Student Conference on Electrical Engineering : May 16, 2013, Prague. Czech Technical University in Prague, 2013, pages 1–5. ISBN 978-80-01-05242-6.

M. Bačíková Domain analysis with reverse-engineering for GUI feature models In POSTER 2012 : 16th International Student Conferenece on Electrical Engineering : May 17, 2012, Prague, Czech republic. Czech Technical University in Prague, 2012, pages 1–5. ISBN 978-80-01-05043-9.