Formalization of Graphical User Interfaces Using Ontologies

Michaela Bačíková¹, Štefan Nitkulinec²

¹ Dept. of Computers and Informatics, Technical University of Košice, Letná 9, 042 00 Košice, Slovakia
² Dept. of Cybernetics and Artificial Intelligence, Technical University of Košice, Letná 9, 042 00 Košice, Slovakia

michaela.bacikova@tuke.sk, stefan.nitkulinec@student.tuke.sk

Abstract. The problem of domain model formalization is very complex. In this paper we propose innovative approach to formalization of domain models extracted from graphical user interfaces (GUIs) of existing applications. Domain model extracted from the GUI describes the hierarchy of domain terms and relations between the terms. Output of the formalization process is an ontology where the term hierarchy is represented as the class hierarchy and the relations between terms are implemented using ontology properties. In our research we focus on formalization process that transforms terms into ontology classes with maximal similarity with the transformed term and also identify particular parts of domain model that can be transformed into ontology individuals. Once the domain model is formalized into ontology, new evaluation possibilities are available thanks to the standardized ontological format.

Keywords
formalization, ontology, domain analysis, DEAL, graphical user interface.

1. Introduction

A domain model represents a formalized structure that describes a problem domain, terms from the domain, their properties and relations between the terms. The level of detail of the domain model can vary depending on the goal we aim to achieve. The final form of domain model depends of its purpose.

In our approach, we chose ontologies as means for formalization. At present there is relatively small amount of ontologies and new ones are mostly created by hand or by automated extraction from existing natural language texts. Manual creation of ontologies is time and energy consuming and the results of the automated approaches are oftentimes not optimal, because natural language is ambiguous, therefore further manual work is needed to complete and correct the resulting ontology.

By using our approach it is possible to create new ontologies in an automated manner. Ontologies represent knowledge as a set of concepts within a domain. One of their advantages is that ontologies contain connections (relations) between terms which allows discovering facts without prior knowledge of domain or its terminology.

Compared to other existing approaches to creating ontology, our approach does not try to create ontologies from natural language but from graphical user interfaces of existing applications. We assume that creators of user interfaces named components meaningfully because applications are designed for domain experts. Moreover, components are composed into logical or function groups hierarchically. On this basis we assume that an ontology created by using our approach will be more relevant than such created from natural language texts by automatic processing. This way, we provide an alternative method for creating ontologies that can be combined with the existing approaches to achieve an optimal solution.

Our solution is integrated into an existing project: DEAL¹ [7] - domain extraction and analysis tool, which is able to extract domain models from existing user interfaces. In this paper we will try to prove that it is possible to formalize domain models, extracted by the DEAL method, into ontologies. Thus, at the same time, we will also demonstrate an automatized formalization of existing GUIs into ontologies. The ontological format brings the following benefits:

• Class taxonomy [1] – a structure that can be used to derive relations between terms.

• Semantic check [2] – relations between terms are determined by axioms. Semantic check allows verification and validation of the axioms and looking for any inconsistencies. In case of an inconsistency, reasoners will show where the problem is located and why it arose.

• Ontology comparing [3] – in our approach, the created ontology changes its content according to the changes made in the application which it was extracted from (target application). Tools for comparing ontologies can show differences between

1 https://www.assembla.com/spaces/DEALtool/wiki
two versions of the ontology and allows us to check changes that were made in the application.

- **Ontology querying** [4] – it is possible to query an ontology similarly as a relational database with the difference of the query language. For example it is possible to implement rules verification using queries.

- **Ontology visualization** [5] - there are several tools that can visualize ontology (such as GrOWL, jOWL, Protégé, etc.). Since ontologies created by our approach reflect the target application’s GUI, we can say that we are able to visualize the application itself – specifically the terms represented by its graphical components. So for instance we are able to observe some patterns used in the application user interface without the need to explore the application manually.

The listed evaluation possibilities can be applied on the created ontology, which represents the target graphical user interface. It is important to note that we focus on the terms gained from user interfaces. That means we evaluate ontologies only in context of terms and not in the ergonomic context of applications although it is possible to evaluate both of them.

2. OntoDEAL – ontology formalization method

The OntoDEAL formalization method proposed in this paper is not just a simple transformation of entities from one structure to another – it is a complex solution of how to overcome indifferences between the DEAL domain model and ontology.

![Fig. 1. Context of proposed formalization process](image)

As shown on Fig. 1, OntoDEAL accepts the DEAL domain model as input and formalizes it into ontology. In this section we will describe the rules and method we used to formalize DEAL domain models into ontologies. First we will describe how to formalize term properties from DEAL domain model into ontology properties and then we will show how to identify a term as a class or an individual term.

2.1 Term properties

The domain model is an output of the DEAL extraction process that extracts domain entities from components of the target application’s user interface. An entity of the model contains properties for a particular term and properties of the component, which it is represented by. The formalization process focuses primarily on term properties: name, description and type. This section will describe how these properties are formalized from the DEAL domain model into ontology.

**Name** - is the main identifier of a term. One graphical component in the target application may have the same name as another component in the application. While name duplicity is allowed in the DEAL domain model, ontology format supports only unique names for classes and individuals. To solve this problem we used incremental naming in the cases where the name of an entity already exists. Before an ontology class or individual is created, a name check that looks up into name dictionary if name already exists is performed. If the particular name is not in the dictionary it can be used as the ontology identifier for an entity. In case the name already exists, it will be modified by adding an incremental suffix, which consists of an underline symbol and integer expressing the order of the name. For example if there already is a class named test, the formalization process will handle the situation when another class is about to be created with same name test by renaming the class to test_2 instead. Since the name of a term is the most important information describing the term, it is important not to lose this information. Original name will be kept through the name data property in the ontology. So when referring to a concrete term by name, the name data property will be used instead of a class name.

**Description** – is any additional information about a term. It may represent the purpose of the term or its usage. Since the description is optional and has no predefined form, it is recommended to use regular expressions or wildcards when accessing terms by their description while evaluating ontology.

**Relation Type** – is the type of relation between sibling terms. These types are represented as data properties in ontology. There are 3 relation types:

- **AND** – the term is independent. This type of relation is default and it will not be explicitly stated in ontology.

- **MUTUALLY-EXCLUSIVE** – the term is in disjunction set of terms. Only one term from the disjunction set can be accepted as value to an object property at a time. For example if the data property is gender, there are only two possible terms that can be considered as value: man and woman. However only one of them can be used. In this case the term man as ontology class has the mutuallyExclusiveWith data property with the assigned value woman.

- **MUTUALLY-NOT-EXCLUSIVE** - term is in a set of terms which are not disjunctive. Those terms can be used in one or multiple axioms for the same object property. Such terms are often extracted from graphical components which are part of a selection or of a multiple choice.
Beside term properties there are also properties describing a particular component represented by a term. For this purpose we define several data properties: component class, content/text, tooltip, label, action, icon, type. These properties enrich concrete terms in context of their function or usage.

2.2 Term identification

After initial processing of the DEAL domain model, the term identification phase follows. The most common case is that a term is identified directly by its name. However not every term in the domain model has a name so it must be identified by different way. Indirect identification is done by other term properties or via position of term in a hierarchy.

Term can be identified into two main categories:

- class term
- individual term

Class term is an unchangeable term in a particular domain. In the target application it is such a term, which does not change its form during any user interactions with the application. For example, when an application contains a menu, the terms representing the menu do not change during application usage and therefore such terms can be considered as class terms. A class term is transformed into an ontology class that can be instantiated.

Individual term is an instance of a class. It was created by user interaction with a GUI component represented by the class term. For example, the value of a text field can represent an individual term. Another possibility of term instantiation is dynamic data loading at application runtime affecting components. An individual term is transformed into an ontology individual.

The ratio between class terms and individual terms depends on type of the target application. The number of individual terms cannot exceed the number of class terms because a class term can be instantiated only once. If an application contains many editable components, the number of individual terms increases.

Applications with high number of individual terms are for instance: content management systems, administration tools, applications focused on database manipulation, development environments etc.

On the other hand applications where only small number of individual terms can be identified are: multimedia and graphical software, monitoring tools, games, antivirus applications etc.

3. OntoDEAL prototype

We have implemented the OntoDEAL framework on top of the OWL API [6] to support all stages of formalization process. OWL API supports ontology creation and manipulation. Methods included in our framework simplify the creation of ontology axioms and handle transformation of domain model terms into ontology entities.

Before the input DEAL domain model can be processed, an ontology model is created in memory. Whenever a term from the domain model is processed, corresponding axioms are added into the ontology model. When all the terms are processed, the ontology model is saved into an OWL format [8]. After that the ontology can be evaluated in any software that supports OWL ontologies.

The solution is directly implemented into our DEAL [7] tool. DEAL analyses target application for domain terms and extracts them with their properties, hierarchy and relations. The DEAL method has the following 4 phases:

1. Construction of a component tree from components in the target application’s user interface,
2. Extraction of a domain model without relations between term, with possible domain-irrelevant information,
3. Simplification of domain model i.e. removing empty terms and all unnecessary information from terms,
4. Derivation of relations between terms.

Currently, to be able to use the DEAL tool, the target application has to be written in Java language. However, the proposed formalization method can be applied on any other application made of components.

4. Experiment

We used the results of OntoDEAL to compare two existing applications based on their terminology. For the purpose of this experiment, the following features of the DEAL model are preserved in the generated ontologies:

- term names and descriptions,
- relations between terms,
- class names of the components, which were the sources of terms,
- term hierarchy.

The goal of this experiment is to:

- demonstrate the formalization method on a real application UIs by automatically creating ontologies,
- to utilize one of the benefits of the ontological format – ontology comparing – and thus to answer the question whether it is possible to compare two existing GUIs based on their domain model and to what extent.
Two ontologies were generated from two applications: Freeplane\textsuperscript{2} and FreeMind\textsuperscript{3}, both open-source applications for mind mapping, downloaded from sourceforge.net. By their appearance we assume that both applications are two versions of the same system, while the FreeMind application is older.

The comparison was carried out using the existing ontology tool Protégé\textsuperscript{4}. Protégé has a built-in feature for automatic comparison of two different versions of the same ontology. Protégé can display the number of newly created entities, the number of deleted entities and the number of modified entities and it also provides a list of all changes made in the newer version of the ontology. The result of the comparison of the Freeplane and FreeMind ontologies are the following:

- entities created: 678
- entities deleted: 166
- entities modified: 345

From the results it is obvious that the Freeplane project has changed a lot since the FreeMind version. The results are not 100% accurate, because some components, which have different names but same functionality in both applications, were counted as created/deleted entities not as modified. For example in FreeMind, to create a new mind map, the New term is used for the menu item, and in Freeplane this term was changed to New map, but the functionality stayed unchanged. This is a change that cannot be detected by our method, because the semantics of the functional components is defined by their code and we do not analyze source code of the target application, only the UI.

However, some changes can be identified with certainty, e.g. the change of the menu item class, or the menu item position change that can be identified in the Superclass parameter in Protégé. For example while in FreeMind all the menu items are of the JMenuitem type, in Freeplane the menu items have their own JFreeplaneMenuItem type.

A graphical comparison of icons would certainly improve the results of the comparison, but this is not possible to achieve in Protégé and further research is needed in this area.

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

Related work

Existing approaches to domain analysis of application user interfaces are mostly oriented on reusability. STAR framework [9] focuses on the analysis of web applications to identify design patterns and user interaction paradigms. Its main goal is to formalize the patterns and the paradigms in order to use them in new web applications.

Chen et al. [10] propose the construction of virtual domain ontologies to enrich domain knowledge. Tairas et al. [11] describe the usage of ontologies in domain analysis of domain-specific languages. Unlike our solution, these two methodologies do not create ontologies automatically.

Another approach [12] examines domain models of applications in context of genetic evolution. A more recent publication demonstrates formal processing of domain models as informal meanings [13]. On the other hand OntoDEAL focuses primary on formalization process of domain model.

Conclusion

In this paper, the usefulness of formalization domain models into ontologies was described. The presented innovative approach to formalization brings new evaluation possibilities of application user interfaces in context of their terms. All this is done without having access to source code of target applications.

Every application has its own specific nature regarding user interface. Therefore the proposed OntoDEAL method must be improved in order to deal with particular application specifics. Also formalization of domain models extracted from poorly designed user interfaces that use irrelevant terms can cause that final ontology will be not so useful. Therefore we strongly recommend that only correct applications are used as input to our method.

Since this is only starting point in the area of domain model formalization and evaluation, there still is a need for enhancing the term identification and searching for stereotypes in the existing applications to improve the extracted results.

This research was made for the purpose of automated evaluation of UI domain usability and that is also our plan for future research in this area.

Acknowledgement

This work was supported by VEGA Grant No. 1/0341/13 Principles and methods of automated abstraction of computer languages and software development based on the semantic enrichment caused by communication.
References


