Introduction to Domain Analysis of Web User Interfaces

Michaela Bačíková¹, Jaroslav Porubán², Dominik Lakatoš³
Department of Computers and Informatics
Technical University of Košice
Košice, Slovakia
michaela.bacikova@tuke.sk¹, jaroslav.poruban@tuke.sk², dominik.lakatos@tuke.sk³

Abstract—Graphical user interface (GUI) is the most important part of application, with which users interact directly. It should therefore be implemented in the best way with respect to understandability. Based on this assumption we decided to analyze user interfaces for domain specific information. In our previous works, we strived for a classical desktop GUI analysis and we used a component-based approach. It was done through defining a simple domain-specific language (DSL). Now we would like to continue in this work, but in the field of web user interfaces. The internet provides more promising amount of resources. In this work, we propose taxonomy of web components based on their domain-specific information and we outline our method for extracting this information. In the future, we would like to create a system that is capable of automatically analyzing web UIs. Instead of a DSL we would like to generate ontologies.

Keywords—ontology; graphical user interface; domain-specific language; domain driven component-based analysis; Java; ontology learning; web user interfaces

I. INTRODUCTION

A. Analysis of user interfaces

The question of creating a user interface (UI) has a clear motivation: system that communicates with a user leads to a dialog via UI and therefore it is necessary to construct it – undoubtedly. Sometimes it is hard to separate an analysis and a design phase of software development. An example is when user is a part of development process where the software is incrementally created and in every cycle it is analyzed by developers together with the user.

We however focus on analysis, which is not aimed primarily to create or modify an existing system (our analysis is not the primary feedback for design). In this case it is not required of us to be a part of design and modifications of the system and our conclusions will be useful beyond the existing system. We are conducting analysis for other needs than for design.

This analysis can be used in several fields:

Existing systems are (in most cases) the best existing formal description of a given domain. Based on this assumption, it is possible to initially construct a (semi-)formal description of the domain (a language or an ontology or a tool support for creating sentences) based on UI analysis.

UI usability evaluation – comparing two or more systems (or a new and an old version of a single system) from the perspective of usability, system evaluation by users/automatically.

Generating of user guides

Note: The form of analysis outputs will probably depend on the intended use.

We decided to analyze web user interfaces mainly for the fact, that the Internet is a good source of analysis inputs - one could say: infinite.

B. Analysis of existing systems

Currently, there are three types of analysis of existing systems:

a) on a knowledge base level (database)
b) on a source code level
c) on a presentation level (GUI)

Because user has any access nor to a) or b), these two levels do not need to reflect the domain precisely. The terms used could be written in another language, expressed in shortcuts or there could be any other language barriers. A generic approach to programming prevents expressing the domain (the generic terms are not usable for a specific domain). In a) there is an absence of descriptions of domain processes (apart from the procedures, but we include them into source codes). By contrast, although the source codes represent the domain processes, the source code analysis is very complicated.

C. Analysis of web user interfaces

In our previous work (see III.A) we have tried to automatically formalize interfaces into a form of a very simple domain-specific language (DSL). To design a language, that would provide adequate expressiveness of concepts, their properties, relations with other concepts and hierarchy of interface components, is difficult. Therefore we decided to use an existing solution meeting all these features – ontologies. Existing articles provide many definitions of ontology [10], [12], [16], [17]. We rather use one of them that fully meets our goal than create another one: Ontology is an explicit representation of concepts of some domain of interest, with their characteristics and their relationships.

The benefits of our approach would be a contribution to Semantic Web by generating ontologies automatically. Semantic Web is an extension of the current web in which the data are described in a way that both people and computers
understand. It should facilitate cooperation between people and machines. Central part of the Semantic Web are ontologies. Vision is, software agents will search the web for our requirements and based on them, they offer us solutions to our problems. Manual creation of ontologies is, however, a difficult and tedious work although several tools for creating ontologies exist like Protégé [24]. Therefore there is a need for an automatic creation of ontologies, referred to as an ontology learning.

This article’s goal is to further investigate the extraction process proposed in our previous work (see III.A.) in terms of Web interfaces and ontologies. Contribution of this article would be the proposed taxonomy of GUI components and their preliminary analysis related to domain-specific information. So far, there has not yet been published any such classification with emphasis on defining a GUI metamodel for extracting domain-specific information. This classification will serve our further research in this area and it is a basis for creating our automatic ontology learning system.

D. Tasks and Goals

From our goal, to create a system that would automatically formalize a GUI to generate ontologies, the following basic goals, targets and assumptions have emerged:

Goal 1: Explore the possibility of extracting ontological concepts from different types of interfaces using a component-based analysis.

Assumption 2: The GUI must be made of components.

Note: Assumpt. 1 was defined in our previous work (see III.A.).

Note: Whereas the Web is an endless knowledge base, we will focus primarily on web documents.

Goal 3: Determine whether the analysis of a web interface (or its parts) is possible.

Goal 4: Outline the ontology learning method.

Subgoal 4.1: Identify potential problems related to this approach.

Subgoal 4.2: Suggest other mechanisms (automatic or manual) to improve the outcome.

The paper is further organized as follows: Section II divides GUIs into several groups according to their form and describes possibilities of formalization for each type. We will try to outline our previous work and the state of the art in the field of ontology learning in section III. In section IV we will outline the proposed method for automatic ontology extraction and perform an extensive analysis of existing web components and their meaning as domain specific units which will serve our further research. We also propose new taxonomy for UI components tailored for our approach. Finally section V will provide an insight into our future research and a conclusion.

II. GRAPHICAL USER INTERFACES

We can divide GUIs into five basic groups with respect to their form. We will describe them (and possibilities of their formalization using a component-based approach) in the following subsections.

A. Console UIs

We write “UIs”, not “GUIs” because these are not fully “graphical” UIs. They are obsolete text interfaces that use no graphics. If all text items displayed in the interface program are stored in variables, it is easy to formalize such an interface. But such formalization is rather an analysis on a source code level. Although there are approaches doing this type of analysis like [20] (this work deals with transformation of obsolete UIs into WIMP GUIs). But it is a research from 1997, so we could say it’s obsolete. Console UIs are rarely used today therefore we will not deal with this type of interfaces further.

B. WIMP GUIs

The “WIMP” abbreviation stands for “Windows, Icons, Mouse, Pointers” and represents standard desktop applications. Formalization of such interfaces depends on the type of programming language and the programming style. If governed by guidelines (for example [19]), analysis possibilities grow wider. Analysis of this type of interfaces was a center of our previous work, described in III.A.

C. Web GUIs

In the context of this article we refer to web documents as web applications, for we are dealing with interactive web UIs. Interacting with the interface is either (in HTML documents) by clicking the network links, or (in interactive web applications) similar to WIMP applications – but with content remaining in the web browser (no windows). In this article we will further deal with this type of interfaces.

D. Mobile GUIs

Mobile devices use a completely different and very much simpler type of user interfaces mainly because of their limited operability. There are two types – non-touchscreens and touchscreens. Mobile interfaces (in most cases) consist of Screens that can be pushed to (or popped off) the device screen. Every Screen component has its title and content (in form of graphical components). Domain analysis of such screens should be easy, because they clearly form a hierarchy of concepts. But the process of attaching an analyzer program to a mobile device or mobile device simulator could be complicated. Therefore for now we will not deal with these and we leave them for further research.

E. Other GUIs

This group includes interface of applications that do not fit into the previous four groups, for example Java Web Start applications, applets, flash, touch screen applications (iPad), etc. We will not deal with these types for now.

III. STATE OF THE ART

A. Previous Work

Our previous research has addressed the domain-driven formalization of WIMP GUIs [21], [22], [23]. A formalization program was implemented that produced output in form of a DSL named GUIII (GUI Interaction Language) designed specifically for describing UIs in a simple form. The created
system was capable of extracting terms based on user interaction with GUI. We used Java open source applications for performing experiments. Then after the extraction our system could automatically “replay” the same commands on the application based on the GUIIL file as it would be performed by a user. We used component-based analysis to implement this method.

The idea of the research is that the commands in GUIIL file were domain identifiers of GUI components. For example, if there was a button named “Next”, its domain identifier was “Next”. When in replay mode, the system searched for appropriate component based on its identifier and performed appropriate action on it (for a button, clicking was pre-defined) like it would be performed by a user. The action was not specified in the input file, just the identifier.

Our previous works also contain a detailed description of concepts, features and relations that can be automatically derived and extracted from WIMP GUIs and this was used in the system to derive an action from a component type. We created DSLs to define GUIs, but the types of features in the interfaces rather define ontology than a DSL. We believe that by comparing the application interface metamodel with model of a concrete application it is possible to automatically derive and generate ontologies.

In this article we would like to explore the possibilities of this assumption, to analyze components, their features, relations, and their membership in a hierarchy of concepts derivable by their definitions, structure and location. We will focus on the Web UIs. Concerning Assumption 1 defined earlier, the web GUIs are made of components in a form of HTML tags that can be easily formalized using XML or HTML parsers. Thus it’s possible to use the component-based approach to analyze Web GUIs.

B. Related work

Vision of Semantic Web requires that web documents had semantics defined. The current web however provides documents to read, but not to understand – web agents don’t know anything about their semantics. Ontologies are used to resolve this problem. And the benefit from them is already in a process of utilization. For example in [18] a predefined ontology was used to extract comments from websites to obtain user feedback on some business products, technologies or information systems.

Figure 1. Initial vision of the Web – a net of simple documents/sources

At present therefore many approaches are targeted to ontology learning. Several methodologies for building ontologies exist, such as OTK, METHONTOLOGY or DILIGENT, but they target ontology engineers and not machines [10]. Many methods and different sources of analysis are used to generate ontologies automatically. Among the obsolete and less effective form we would include NLP (Natural Language Processing), used for example in [5]. These approaches are however complicated and their perception Internet as a net of simple documents is obsolete (Fig. 1). The Internet has rather transformed into a web of (dynamic) interactive applications (see Fig. 2).

Figure 2. Transformation of web documents and their perception

A little bit closer to this perception are analyses of concrete web structures (tables) or document fragments or just the basic attributes (title, head) - see methods ix) and x) below. But these are just little parts of a whole web UI.

Results are almost always combined with a manual controlling and completing by a human and as an additional input, there’s almost always some general ontology present (a “core ontology”) serving as a “guideline” for creating new ontologies. Different methods are used to generate ontologies:

i) clustering of terms [7], [13], [15],
ii) pattern matching [5], [13], [14], [15],
iii) heuristic rules [5], [14], [15],
iv) machine learning [1], [12],
v) neural networks, web agents, visualizations [15],
vi) transformations from obsolete schemes [14],
vii) merging or segmentation of existing ontologies [8], [13],
ix) analysis of web table structures [4], [5], [17],
x) analysis of fragments of websites [3].

A condition for creating a good ontology is to use many sources as an input to analysis - structured, non-structured or semi-structured - and to use a combination of many methods [10]. Therefore as an additional mechanism for identifying different types of relationships (e.g. mutual exclusivity, hierarchical relations), a web dictionary WordNet ([4], [9], [11] and [13]) or other web dictionaries or databases available on the Web are used.

A state of the art from 2007 can be found in [10].

IV. PROPOSED METHOD FOR AUTOMATIC ONTOLOGY EXTRACTION

We would like to design such a method which would not require human control at runtime and it would require just a minimal check at the end of the process. We also want to analyze websites in a simple, transparent way to enable extensibility and focus only on part that creates GUIs – because the presentation layer of an application is the first thing that a user sees and it should be programmed in such a manner that a general user would understand its language. This presumption allows us to analyze web documents as UIs. Although we will
use an existing general ontology as an input, this ontology will be created only once by us and will no longer be changed (only when programmer would like to extent the programming). It will define GUI web documents on a metalevel and will serve as a “guideline” for creating new ontologies. In the following sections we will try to describe how we define the characteristics of some Web graphical user interface components. Based on these characteristics we will create the algorithm for GUI formalization.

A. Assumptions and classification

In our previous work we were conducting analysis based on following assumptions:

i. GUI is made of components.
ii. These components are domain-specific units – they represent terms of a GUI language.
iii. For these terms, we can define properties, rules, there are some relations between them and they are creating a hierarchy of terms.
iv. We can identify, extract and process them.

We will continue to use these assumptions when performing analysis of web components. In this work we use a different classification than in our previous research. We believe that this proposed taxonomy will better serve the web interface automatic analysis algorithm:

i. Components giving metainformation about their content

<table>
<thead>
<tr>
<th>HTML Tag</th>
<th>Group</th>
<th>Example of use</th>
<th>Domain-specific information, which the component defines</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;a&gt;</td>
<td>ii, iv</td>
<td><code>&lt;a href=&quot;http://www.google.com&quot;&gt; Click me! &lt;/a&gt;</code></td>
<td>Defines hierarchy or web of documents.</td>
</tr>
<tr>
<td>&lt;abbr&gt;, &lt;acronym&gt;</td>
<td>i</td>
<td>The <code>&lt;abbr title=&quot;World Health Organization&quot;&gt;WHO&lt;/abbr&gt;</code> was founded in 1948.</td>
<td>Defines meaning of a shortcut / acronym – metainformation.</td>
</tr>
</tbody>
</table>
| <address>  | i     | `<address> Written by W3Schools.com<br />
<a href="mailto:us@example.org"> Email us </a> <br />
Address: Box 564, Disneyland </address>`                                                                                               | Define type of their content – the page creator address – metainformation.                                                                     |
| <button>   | iii   | `<button type="button"> Click Me! </button>`                                                                                                                                                          | Customizable push button. Functional component performing some action and changing state of application.                                      |
| <form>     | i, iii, iv, v | `<form>
<legend>Person</legend>
<label for="male"> Male </label>
<input type="radio" name="sex" id="male" />
<br />
<label for="female"> Female </label>
<input type="radio" name="sex" id="female" />
</form>`                                                                 | We can extract many domain specific information of a form:
- If the form has a `<legend> tag (like in this case "Person") or a name attribute specified, it specifies (sub)domain of the form content.
- The input components are of many types
  - submit - submits information to server,
  - radio - terms expressed by radio buttons are mutually exclusive,
  - text - represents textual information, its properties and limitation,
  - password - represents encoded information, password,
  - checkbox - terms expressed by checkboxes are not mutually exclusive.
- For description of the input fields, the `<label>` tag with a for attribute is used. This specifies a meta information about the input component.
Note: a work similar to ours is already in utilization by existing web browsers, for example identifying and remembering user password – the fields with username and password are automatically identified and their content stored in browser’s memory. |

B. The analysis of Web GUI term extraction possibilities

Analysis of web components and their classification to the defined groups describes TABLE I. In the first column there are names of components represented by HTML tags. In the second, there are groups of the taxonomy defined in IV.A (as i, ii, iii, iv, v, vi). In the third column we give an example of use. The code examples were selected from www.w3schools.org. And finally, in the last column we describe each type of component and domain specific information that it defines. This is important for designing the GUI analysis algorithm. For some components (or their parts) it is possible to extract many types of information, therefore we included these components into several groups.

Note: because of limited space, we have chosen only the most commonly used components and such, that can offer the most important domain-specific information defining the various concepts, structures, hierarchy and relations between future ontology concepts.
<table>
<thead>
<tr>
<th>HTML Tag</th>
<th>Group</th>
<th>Example of use</th>
<th>Domain-specific information, which the component defines</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;select&gt;</td>
<td>i, ii</td>
<td>&lt;select name=&quot;cars&quot;&gt; &lt;option value=&quot;volvo&quot;&gt;Volvo&lt;/option&gt; &lt;option value=&quot;mercedes&quot;&gt;Mercedes&lt;/option&gt; &lt;/select&gt;</td>
<td>Selection list item. - the <code>name</code> or <code>id</code> attribute can tell us about the domain of selection's items - the items in the selection list represent mutually exclusive terms Selection list item can be organized also hierarchically, using an <code>&lt;optgroup&gt;</code> item. From such structure we can directly extract the whole hierarchy of terms.</td>
</tr>
<tr>
<td>&lt;dl&gt;</td>
<td>i, iii</td>
<td>&lt;dl id=&quot;menu&quot;&gt; &lt;dt&gt;Milk&lt;/dt&gt; &lt;dd&gt;White cold drink&lt;/dd&gt; &lt;dt&gt;Coffee&lt;/dt&gt; &lt;dd&gt;Black hot drink&lt;/dd&gt; &lt;/dl&gt;</td>
<td>Definition list. The <code>&lt;dl&gt;</code> tag gives items of the definition list. Of them we can tell, that they belong to a same subdomain. The <code>&lt;dt&gt;</code> tag gives description of a definition list item. Very similar tags are <code>&lt;ul&gt;</code> (unordered list) and <code>&lt;ol&gt;</code> (ordered list) and deprecated <code>&lt;dir&gt;</code> (directory list), all with <code>&lt;li&gt;</code> items, but they do not have the description tags (dd). Some HTML creators use these lists to create a menu of a webpage (see the second example with <code>menu</code> attribute). From such structures we can directly extract page menu, e.g. we get the hierarchy of web pages. There is also a deprecated tag named <code>&lt;menu&gt;</code>, that looks exactly like unordered list. This better describes menu content that can be extracted.</td>
</tr>
<tr>
<td>&lt;em&gt;, &lt;strong&gt;, &lt;dfn&gt;, &lt;code&gt;, &lt;samp&gt;, &lt;kbd&gt;, &lt;var&gt;, &lt;cite&gt;</td>
<td>i</td>
<td>&lt;em&gt;Emphasized text&lt;/em&gt; &lt;strong&gt;Strong text&lt;/strong&gt; &lt;dfn&gt;Definition term&lt;/dfn&gt; &lt;code&gt;Computer code&lt;/code&gt; &lt;samp&gt;Sample computer code&lt;/samp&gt; &lt;kbd&gt;Keyboard text&lt;/kbd&gt; &lt;var&gt;Variable&lt;/var&gt; &lt;cite&gt;Citation&lt;/cite&gt;</td>
<td>Formatting tags give metainformation about their content. Commonly when writing some text, the important parts of documents, definitions, or key words, are written using italic or bold format. For tags like <code>&lt;em&gt;</code> and <code>&lt;strong&gt;</code> we therefore can assume, that their content is something important. About the other tags, meaning of their metainformation is obvious when looking at their name or sample codes.</td>
</tr>
<tr>
<td>&lt;table&gt;, &lt;caption&gt;, &lt;th&gt;, &lt;tr&gt;, &lt;td&gt;</td>
<td>i, ii</td>
<td>&lt;table border=&quot;1&quot;&gt; &lt;tr&gt; &lt;th&gt;Month&lt;/th&gt; &lt;th&gt;Savings&lt;/th&gt; &lt;/tr&gt; &lt;tr&gt; &lt;td&gt;January&lt;/td&gt; &lt;td&gt;$100&lt;/td&gt; &lt;/tr&gt; &lt;/table&gt;</td>
<td>Tags defining tabular structures. From <code>&lt;caption&gt;</code> tag we can get table (sub)domain, e.g. metainformation about table content. From <code>&lt;th&gt;</code> we can determine (sub)domain of columns content. This way we can create a concept hierarchy. From cell properties we can derive properties of ontology concepts. Similar work was done in [4], [5], [17], mostly using NLP, regular expressions, type recognizers and clustering. We would rather keep it simple when dealing with tables so the hierarchy of a whole webpage wouldn’t go into too much detail, because then it would be very difficult to maintain such a big ontology.</td>
</tr>
<tr>
<td>&lt;textarea&gt;</td>
<td>v</td>
<td>&lt;textarea rows=&quot;2&quot; cols=&quot;20&quot;&gt; At W3Schools you will find all the Web-building tutorials you need, from basic HTML to advanced XML, SQL, ASP, and PHP. &lt;/textarea&gt;</td>
<td>Text area. From text components we can get domain specific information like: - type of data, - maximal length, - content, - constraints, - purpose from the <code>&lt;label&gt;</code> component, etc.</td>
</tr>
<tr>
<td>&lt;tt&gt;, &lt;i&gt;, &lt;b&gt;, &lt;big&gt;, &lt;small&gt;, &lt;h1&gt; - &lt;h6&gt;</td>
<td>vi</td>
<td>This is some italic text.</td>
<td>Formatters and separators. Besides separating different types of content from each other they do not contain any relevant domain-specific information. Formatting tags serve similar purpose as <code>em</code>, <code>&lt;strong&gt;</code>, etc.</td>
</tr>
<tr>
<td>&lt;head&gt;</td>
<td>i</td>
<td>&lt;head&gt; &lt;title&gt;This is title&lt;/title&gt; &lt;/head&gt;</td>
<td>From web page <code>&lt;head&gt;</code> we can extract page's <code>name</code>, e.g. name of web page's domain. Using a <code>&lt;meta&gt;</code> tag the programmer defines metadata about the website.</td>
</tr>
<tr>
<td>&lt;img&gt;, &lt;object&gt;, &lt;map&gt;</td>
<td>i</td>
<td><code>&lt;img src=&quot;angry.gif&quot; alt=&quot;Angry face&quot; /&gt; &lt;object width=&quot;400&quot; height=&quot;400&quot; param name=&quot;movie&quot; value=&quot;/flash/helloworld.swf&quot; /&gt;</code></td>
<td>Component of type <code>&lt;img&gt;</code> contains a picture. An object can contain image, audio, video, Java applet, ActiveX, PDF or Flash. Tag <code>&lt;map&gt;</code> contains a clickable image map. As a domain-specific information we can consider the object type, e.g. a “media or interactive content” – a metainformation.</td>
</tr>
</tbody>
</table>
V. FURTHER RESEARCH (THE PROPOSED METHOD)

As written in the above sections, we would like to continue our research in the field of domain driven GUI analysis and there’s much work to be done. We will continue to focus on web interfaces, because this area provides promising amount of resources.

Proposed analysis of components must be extended to cover the most of web GUI components and to explain in detail, what domain specific information can be extracted for the algorithm to be defined. A formalization method of each type of component must be defined. Based on this analysis, the general (core) ontology must be defined and created to serve the creation of domain-specific ontologies. The method outlined in this article must be finalized and implemented into a system capable of automatic ontology learning. Subgoal 4.1 and Subgoal 4.2 defined in I.B. must be fulfilled. To better the results WordNet or other dictionaries can be used. Experiments must be done with the implemented system and results evaluated and this information should help fulfilling the subgoals 4.1 and 4.2.

VI. CONCLUSION

This paper serves as an introduction to domain analysis of web GUIs. It is a basis for our further research. In this work we proposed a new taxonomy for web components serving domain driven component-based analysis. We described our method and tried to make an extensive analysis of existing HTML components to serve the design of our algorithm. Based on this analysis we will try to implement a tool which is capable of automatic generation of ontologies and by this it will contribute to the growth of the Semantic Web.

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


