COMPUTER
ENGINEERING
PROGRAM

Middle East Technical
University
Northern Cyprus
Campus

eMINE Technical Report Deliverable 3 (D3),

September 2013

Automatic Discovery of Visual Elements of Web Pages

M. Elgin Akpınar
elgin.akpinar@metu.edu.tr
Middle East Technical University,
Ankara, Turkey

Yeliz Yesilada
yyeliz@metu.edu.tr
Middle East Technical University
Northern Cyprus Campus,
Kalkanlı, Güzelyurt, TRNC,
Mersin 10, Turkey

Web pages are typically designed for visual interaction – they include many visual elements to guide the reader. However, when they are accessed in alternative forms such as in audio, these visual elements are not available and therefore they become inaccessible. To address this problem, we have proposed an approach to identify visual elements in a web page and then characterize the semantic role of these elements. Our system architecture has three major components: 1. automatic identification of visual elements of web pages, 2. automatic generation of heuristics as Jess rules from an ontology and 3. application of these heuristic-based rules to web pages for automatic annotation of visual elements and their roles. The purpose of this technical report is to introduce our probabilistic approach and describe its technical details.

METU
NCC

eMINE

The World Wide Web (web) has moved from the Desktop and now is ubiquitous. It can be accessed by a small device while the user is mobile or it can be accessed in audio if the user cannot see the content, for instance visually disabled users who use screen readers. However, since web pages are mainly designed for visual interaction; it is almost impossible to access them in alternative forms. Our overarching goal is to improve the user experience in such constrained environments by using a novel application of eye tracking technology. In brief, by relating scanpaths to the underlying source code of web pages, we aim to transcode web pages such that they are easier to access in constrained environments.

Acknowledgements

The project is supported by the Scientific and Technological Research Council of Turkey (TÜBİTAK) with the grant number 109E251. As such the authors would like to thank to (TÜBİTAK) for their continued support.

Contents

Middle East Technical University,
Northern Cyprus Campus,
Kalkanlı, Güzelyurt, TRNC,
Mersin 10, TURKEY

Tel: +90 (392) 661 2000
http://www.ncc.metu.edu.tr/

Corresponding author:
M. Elgin Akpınar
elgin.akpinar@metu.edu.tr

1 Introduction

Web pages consist of many visual elements in order to guide the readers and main purpose on design process is typically visual interaction. As web technologies evolve, they come up with more useful and powerful tools and libraries. Using these tools, designers and developers are able to create more visually interactive web pages. This also results in technically more sophisticated layouts and web page structures. Unfortunately, far too little attention has been paid to accessibility issues. Although these technologies improve visual interaction, these visual elements are not available when pages are accessed in alternative forms, such as in audio with assistive technologies, and as a result, web pages become inaccessible. The key to solve this problem is to have a deep understanding of the structure of web pages along with the role of visual elemens, since these identified web elements help to re-process the web page in transcoding applications, which may both improve accessibility and provides better presentation in mobile devices [10, 11, 3]. Furthermore, the application area could be extended to accuracy improvement of information retrieval and data mining [7], or design of better intelligent user interfaces [8]. In order to sum up, role analysis of the web elements in the web pages provides very valuable information about the web page structure, which helps in many web engineering applications.

Visual elements in the web pages have many characteristics, and these characteristics help us to identify the heuristic role of a visual element. However, these characteristics are defined with HTML and CSS by the designers and developers, and both HTML and CSS enable them to create the same visual layout with different coding styles with their flexible syntaxes. This flexibility and lack of standards (or it is more proper to address it as avoidance of standards) in web design and development, make automating the role identification a very challenging task [2]. On the other hand, these characteristics are mainly based on visual representation of the visual elements and visual representation styles change in time, based on the current trends. Therefore, these characteristics must be defined in a dynamic knowledge base, which enables easy modification after some changes in web trends. In order to address these problems, we proposed an ontology based probabilistic approach for automatic identification of web elements.

There are many researches conducted in identification of visual elements. [9] describes these researches in terms of their motivation, purpose, method and evaluation technique in detail. When we look at the literature, we see that recent studies are either outdates by new design trends and new web technologies, or have a very simplistic static definition of heuristic rules. Therefore, we need a more up-to-date method with an extended understanding of visual elements and their roles.

This technical report presents our atomatic discovery method for visual elements in web pages which operates in following way: first of all, visual elements in the page are identified. Then, the characteristics of visual elements are converted to rules in an appropriate syntax for reasoning. Finally, these rules are applied to visual elements and for each visual elements, a likelihood score for each role is calculated. The role with maximum likelihood score is assigned to the corresponding visual element. Our approach also proposes an ontology which aims to define useful roles for visual elements and describe these role based on the general characteristics of their visual representation. In this technical report, each part of our approach and the ontology are explained in detail.

The technical report has been organised in the following way: Section 2.1 gives brief in-

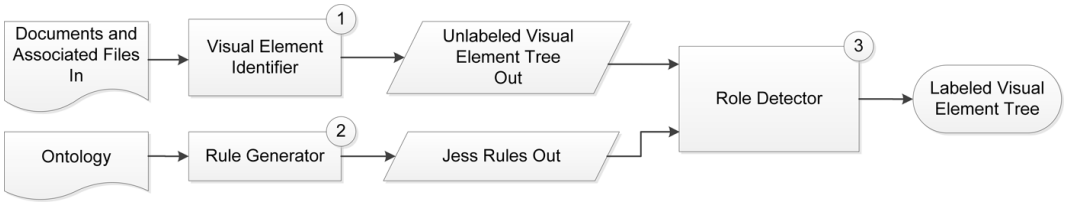

Figure 1: System Architecture

formation about visual element identification process. Section 2.2.1 compares and contrasts existing powerful knowledge bases and explains their weak points. Section 2.2.2 introduces eMine Ontology and its characteristics. Then, Section 2.2.3 describes our iterative role detection implementation process. Finally, Section 3 concludes our technical report.

2 Automatic Discovery of Visual Elements of Web Pages

In this section, we introduce our automatic discovery method for visual elements. We can list our design criteria as follows:

Modifiability: Our method should be modified with respect to different application purposes, since different application areas may focus different characteristics of visual elements.

Maintainability: Proposed method must be able to be adapted to changing design trends and also new web technologies.

Coverage: The role characteristics should cover a wide range of visual element attributes and these characteristics should be properly selected for describing the role.

As was mentioned earlier, web trends and technologies change rapidly. Moreover, application area of heuristic role detection of visual elements is so wide that, it includes many disciplines which may be related, but indeed have some unique requirements. Mobile device adaptation and information retrieval are two examples of such applications. Therefore, our main concern in this part is to provide a knowledge base, which is adaptable to changing trends and technologies as well as modifiable for different application areas. We aim to achieve this by creating an ontology based knowled base.

The overall architecture of our proposed system has three main components which are summarised below and illustrated in Figure 3.

Visual element identifier: takes a web page and by using its visual presentation and source code automatically divides it into visual elements and creates a visual elements tree.

Rule generator: component takes our knowledge base of visual elements implemented in an OWL ontology and generates heuristic rules for visual elements.

Role detector: component takes rules and visual elements tree generated by our first component and returns a labeled visual element tree.

Our system has been implemented on the Accessibility Tools Framework (ACTF) of Eclipse Foundation[1]. We also committed our web element identifier implementation to ACTF and it is available under the terms of the Eclipse Public License v1.0[2], so that, anyone can benefit from our contribution.

2.1 Visual Element Identification

In order to detect the roles of visual elements in a web page, we need to first identify meaningful visual elements in the page by segmenting it in an efficient and convenient way. In order to identify these visual elements in web pages, we proposed an algorithm, which is an extended version of VIPS Algorithm [1, 2]. In brief, our algorithm considers both DOM structure and visual representation of a web to segment it into visual elements. The algorithm produces a tree of visual blocks in a hierarchical structure. This tree differs than the DOM structure of the page, since the algorithm eliminates the invalid nodes and group the adjacent virtual text nodes and inline nodes into one single visual block. Therefore, the tree of visual blocks is simpler when it is compared to the DOM structure of a web page. In our ACTF implementation, the visual element identifier generates the tree of visual elements and creates an XPATH[3] for each visual block with respect to its corresponding node.

2.2 Automatic Role Detection

After identifying the visual elements in a web page, this section explains automatic role detection process for visual elements. The main process can be divided into two: rule generation, in which, a comprehensive knowledge base is constructued and the knowledge is converted to a suitable rule format; and role detection, in which, these generated rules are applied on the visual elements that are identified by segmenting the web page. Finally, we have a tree of visual elements in the web page in an hierarchical structure, and all the visual elements are labeled with an appropriate role.

Most approaches in the literature propose a single rule based role detection methods. These single rules are defined with respect to some specific attributes of visual elements. For example, link count is used to detect the role of the visual element. However, this kind of methods are based on strict assumptions and flexibility in HTML and CSS makes such assumptions invalid. A designer also provide the same functionality by using click events on any other node than link, and do not require to use any link node in the web page. An assumption on a specific attribute as in this case, is likely to fail in most of the web pages.

However, if we increase these assumptions to cover a wide range of visual element attributes and common usage styles, then we increase our chance in detecting the role of a visual element. Therefore, we need to examine distinguishing characteristics of all the roles and this is also a challenging task, since we must define lots of rules. Even if we crate a knowledge base which covers characteristics retrieved from millions of pages, there will still exist many pages which would contain some visual elements which do not satisfy all the requirements of their corresponding roles. Therefore, it is not possible to apply direct reasoning to detect the heuristic role of a visual element, instead we can define likelihood

[1] http://www.eclipse.org/actf/
[2] http://www.eclipse.org/legal/epl-v10.html
[3] http://www.w3.org/TR/xpath/

scores for each role and assign values to these scores based on some criteria which visual element satisfies. At the end the highest likelihood gives us the role of the corresponding visual element.

2.2.1 A Comparison Between WAfA, ARIA and HTML5

There are some studies and standards which aim to provide a classification of visual elements. Among these, the most striking ones are WAfA, ARIA and HTML5. In this section, we compare and contrast these classifications in brief, and discuss how we benefit these knowledge bases.

One study conducted on this field proposes WAfA Ontology, which aims to capture shared understanding of visual elements in web pages [6]. WAfA Ontology provides a classification for atomic web elements and chunks, as well as nodes, which represent web pages, and collections, which represents web sites. When we analyse the coverage of WAfA in its classification, we see that, WAfA provides a very rich knowledge base; however, our initial experiements showed that it still needs further extensions. These extensions includes both some reductions on defined roles and some additional roles. First of all, since we are only interested in identification of visual elements in web pages, node and collection classes should not be included in our knowledge base. Also, some of the concepts are too specific that, it is not possible to generate descriptive rules to distinguish such role from their siblings. For example, *Link* role is divided into five roles, which are *ReferentialLink*, *AssociativeLink*, *StructuralLink*, *SkipLink* and *ToTextOnlyPage*. These specific role definitions have a few or no distinguishing characteristics than each other, so that, it is not possible to define rules to decide on specific role. Consequently, all of these subroles are included as *Link* in our ontology. Finally, when we compare it with other knowledge bases, we see that some further role definitions are required in WAfA Ontology.

Another powerful knowledge base in this field is ARIA Ontology [4]. When we compare the concepts in WAfA to ARIA, we see that, both contain many concepts in common. However, ARIA includes some noteworthy roles which are not available in WAfA. The missing concepts in WAfA are textual classifications and interaction items. Textual classes are important since they describe the main content of the page, which is unique in a web site. Also, there are many form elements for user interaction, and WAfA only includes a narrow definition of such items. HTML5 also provides a role set, which enables developers to specify the role of the visual elements in role attribute. This is also very helpful for our study. However, when we look at the proposed roles, we see that most of the roles are originated from ARIA Ontology, and the remaining roles are already defined in WAfA. Therefore, although HTML5 is also a very powerful resourse, the combination of three ontologies

So far we have analysed existing knowledge resources and their limitations. After analysis, we see that each knowledge resource includes many important roles; however, they also have some missing concepts. Therefore, we decided to create a new ontology, which combines the important concepts of these ontologies. Our new ontology is called eMine Ontology and available at `http://emine.ncc.metu.edu.tr/ontology/emine.owl`. The role set and description of these roles are given in Appendix A.

News	http://news.yahoo.com/, http://edition.cnn.com/, http://huffingtonpost.com, http://bbc.co.uk/news/, http://nytimes.com, http://news.google.com, http://msnbc.msn.com, http://weather.com, http://reddit.com, http://foxnews.com
Shopping	http://amazon.com, http://ebay.com, http://netflix.com, http://walmart.com
Arts	http://imdb.com, http://deviantart.com, http://scribd.com
Design	http://www.awwwards.com/, http://www.cssdesignawards.com/, http://tympanus.net/codrops/, http://www.instapaper.com/, http://www.readability.com/, http://www.html5rocks.com/en/, http://www.google.com/insidesearch/underthehood.html, http://www.aupetitpanisse.fr, http://www.fubizawards.com/, http://www.kevorkkiledjian.com, http://www.colorz.fr/, http://demo.fluent.io/

Table 1: Examined page list

2.2.2 Knowledge Representation and eMine Ontology

In the previous section, we have constructed a set of roles in our ontology. Our main concern is to detect the role of a visual element among these roles according to some criteria which based on the attributes of the visual element. Therefore, our aim here is to describe these roles with their general characteristics and define some properties of them.

In order to find the general characteristics of the roles of visual elements, we have analysed a set of popular web pages. The web pages we analysed were selected from Alexa[4], which provides analytics for web pages. It also classifies web pages into genres, and in order to provide a general understanding for common use of web technologies, we have selected pages from different genres. Also, in order to provide knowledge about current technologies and web trends, we have included some well designed HTML5 web pages. The pages which we analysed are represented in Table 1.

In this analysis, we examined the identical characteristics of each role and also tried to find some unique attributes to distinguish these roles from the other roles in the ontology. As we anticipated, HTML and CSS with their very flexible syntax enables to create the same visual representation by using different coding styles. Therefore, it is not possible to fully describe a role with all of its characteristics, rather, we can define a set of them for our

[4] http://www.alexa.com/

purpose. After our investigations on web pages, we observed that the following properties affects how visual elements are used and represented:

- Underlying tag (HTML/HTML5);
- Children and parent elements in the underlying DOM tree;
- Size of the element;
- Border and background color of the element;
- Position of the element;
- Some attributes including onclick, for, onmouseover, etc.;
- CSS styles (font-size, color, etc.) of the elements;
- Some specific keywords which appear in the textual content of the element;
- Some specific keywords which appear in the id, class, role, src, background-image attributes of the element.

In our web page analysis, we not only defined the attributes which describe a role, but also investigated the values of each attribute for each role. In other words, we described each role with their characteristics in general use. Some of these values were categorised under some classes, such as HTML tags, and some of them included in the ontology as string entities, such as textual keywords. These values were assigned to roles by using object porperties in the ontology. An object property can be thought as a function definition which as both range and domain definitions. The domain specifies the classes which an object property can be applied on, while the range specifies the type of its value. Complete list of object properties in our ontology is given in Appendix B. Based on the ranges of object properties, we classified them under two groups. The first group of object properties takes exactly one value which is either a role, string, tag, attribute or any other class defined in the ontology. These object properties were used to define relations which takes a predefined class or a string literal in its range. The second group of properties, on the other hand, takes an undefined complex description of role as its value, such as, for an object property which defines having a child with DIV tag. In this example, we refer to a class definition, which is not already defined in the ontology, but it needs to be used in a definition. We can refer second group as nested definition of object properties, since in fact it contains two nested object properties.

After these definitions, we have conducted a set of manual experiments with our ontology. After these experiments, we have found that, the accuracy can be further improvement with a couple of additions in the ontology. Therefore, we have followed an iterative approach in development process. First of all, we observed that, some object properties are more important than others in describing a role, since they affect the decision on role detection more than others. For example, id attribute of a visual element gives more accurate information about its role than its size. Therefore, the object property which corresponds to id of the visual elements should have higher weight than the object property which describes the size. In order to achieve this, we defined a factor attribute for object properties, and with manual experiments we assign a factor for each role. By doing so, we defined an

importance value for each object property. Morever, we observed that, some values also affect the accuracy than other values for the same role and the same object property. *Title* role is a good example for such cases. It is possible to create a *Title* visual element by using either a H1, H2 or H3 tag or a P tag with the font weight of the visual element set bolder than the remaining of the page. In both cases, a designer produces the same visual representation and can create a *Title* visual element. Although this is the case, only using a P tag does not produce a *Title* element on its own. In order to specify it in object properties, we defined a multiple level of object properties, in which an object property like must_have_tag has higher factor than an object property, such as may_have_tag. In our manual evaluation, we observed that these enhancements improved the accuracy of role descriptions.

At the end of our development process, an object property has the following structure:

```
<owl:ObjectProperty rdf:about="emine#has_id">
  <rdfs:comment><![CDATA[Specifies the id or class name which an
                atom or a chunk may have.]]></rdfs:comment>
  <rdfs:label>has_id</rdfs:label>
  <rdfs:factor>5</rdfs:factor>
  <rdfs:domain>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="emine#atom"/>
        <owl:Class rdf:about="emine#chunk"/>
      </owl:unionOf>
    </owl:Class>
  </rdfs:domain>
  <rdfs:range>
    <owl:Class rdf:about="http://www.w3.org/2001/XMLSchema#string"/>
  </rdfs:range>
</owl:ObjectProperty>
```

In this example, factor, domain and range definitions are represented. Using the object properties, then we draw relationships between the role definitions and their characteristics. A brief example of a role definition in our ontology can be represented as following:

```
<owl:Class rdf:about="emine#Header">
  <rdfs:comment><![CDATA[Is typically printed at the top of a page
           and includes, for example, a company logo, the title of
           the page, list of links and sometimes a search engine.]]>
           </rdfs:comment>
  <rdfs:label>Header</rdfs:label>
  <rdfs:subClassOf rdf:resource="emine#chunk" />
  <owl:intersectionOf rdf:parseType="Collection">
    <owl:Class rdf:about="emine#chunk" />
      <owl:Restriction>
        <owl:onProperty rdf:resource="emine#must_have_tag" />
        <owl:hasValue rdf:resource="emine#Header" />
      </owl:Restriction>
      <owl:Restriction>
        <owl:onProperty rdf:resource="emine#has_tag" />
        <owl:hasValue rdf:resource="emine#Div" />
      </owl:Restriction>
      <owl:Restriction>
        <owl:onProperty rdf:resource="emine#in_position" />
        <owl:allValuesFrom>
```

```
            <owl:Class>
              <owl:oneOf rdf:parseType="Collection">
                <owl:Thing rdf:about="emine#Top" />
                <owl:Thing rdf:about="emine#Left" />
                <owl:Thing rdf:about="emine#Right" />
              </owl:oneOf>
            </owl:Class>
          </owl:allValuesFrom>
        </owl:Restriction>
      ...
    </owl:intersectionOf>
  </owl:Class>
```

In common use of the ontologies, the object properties fully describe an object. A sample transition of an object and its properties to first order logic is as follows [5]:

$$\begin{aligned}&\exists x \exists y.\mathit{triple}(\mathit{block},\ emine : has\_tag,\ x)\\ &\quad \wedge.\mathit{triple}(\mathit{block},\ emine : has\_style,\ y)\\ &\quad \wedge.\mathit{triple}(\mathit{block},\ emine : has\_id,\ \mathit{"header"})\\ &\quad \wedge.\mathit{triple}(x,\ emine : name,\ \mathit{"Header"})\\ &\quad \wedge.\mathit{triple}(y,\ emine : name,\ \mathit{"border"})\end{aligned}$$

According to this representation, we need to fully describe a visual element role with all of its characteristics. In order to achieve this, we need to be able to determine all of its attributes in the domain of the problem. However, our problem arises from the fact that we cannot define all the attributes of a heuristic role and a visual element does not have to satisfy all the attributes of its role due to the flexible syntax of HTML and CSS. Therefore, we construct a probabilistic approach to the problem. We define a set of attributes which a visual element might satisfy all of the attributes or some of them. The more attribute of a rule a visual element satisfies, the more likely that the visual element has the corresponding rule. Our approach to the ontology and object properties can be represented as follows:

$$\begin{aligned}&\exists x \exists y.\mathit{triple}(\mathit{block},\ emine : has\_tag,\ x)\\ &\quad \vee.\mathit{triple}(\mathit{block},\ emine : has\_style,\ y)\\ &\quad \vee.\mathit{triple}(\mathit{block},\ emine : has\_id,\ \mathit{"header"})\\ &\quad \wedge.\mathit{triple}(x,\ emine : name,\ \mathit{"Header"})\\ &\quad \wedge.\mathit{triple}(y,\ emine : name,\ \mathit{"border"})\end{aligned}$$

2.2.3 Rule Generation

After we created the ontology, we integrated it with our implementation. The first step of integration was ontology parsing with OWL API, which is a Java API and reference implementation for creating, manipulating and serializing ontologies. It is also open source and available under either the LGPL or Apache Licenses. Finally, it includes RDF/XML, OWL/XML parsers which fits with the structure of our ontology.

In order to parse our ontology, we also required a reasoner. A reasoner is used to query basic information from the ontology, such as the subclasses of a given class, or defined object properties and their values for each class. Since our knowledge base is dynamic and open to any modification in both role definitions and object properties, a reasoner is the key component in ontology parsing. By querying the subclasses of two main classes in our

ontology, which are atom and chunk, we can retrieve all roles from the ontology, as well as their object properties. The OWL API includes various interfaces for accessing OWL reasoners. In this study, we have used a reasoner, called HermiT.

HermiT is a reasoner for ontologies written using the OWL and capable of determining whether or not the ontology is consistent and identify subsumption relationships between classes. Using Hermit, we can obtain all subclasses of atom and chunk concepts, and their characteristics defined by object properties. Also, it helps us to check whether the ontology is well formed or contains syntax errors. one reason for using HermiT in our implementation is that it has the required interfaces with OWL API and it is easy to integrate in our system. Another reason is that, HermiT is a buffering kind of reasoner, which is the default kind. In nonbuffering reasoners, the ontology changes are processed by the reasoner immediately so that any queries asked after the changes are answered with respect to the changed ontologies. On the other hand, in buffering reasoners, ontology changes are stored in a buffer and are only taken into consideration when the buffer is flushed with the flush() method. Since ontology changes are not applied in execution life cycle, we used a buffering reasoner.

Using the reasoner, we could retrieve all subclasses of atom and chunk concepts in OWLClass type. Moreover, we can query the properties in OWLObjectProperty type. However, we observed that, ontology parsing is a costful process and increases the response time of the system when it is called in each run. Moreover, although the knowledge base could be modified for some reasons, it is not as frequently that we need to parse the ontology in each time we run our program. Therefore, in order to solve this problem, we came up with a solution, in which ontology is converted into an appropriate format for later role detection process, and parsed again only if it is externally modified. To achieve this, we calculated the MD5 checksum of the ontology file and kept it in a local file. When the program is run and there is an established internet connection, the program calculates the MD5 checksum of the ontology in the server and compares it to the value in the local file. In there is a difference between two values, then it means that ontology was changed and it needs to be parsed again. Parsed information is kept in a local CLP file, which will be explained in detail. With this enhancement, we reduced the time required to download and parse the ontology and convert to the Jess rules. We gained 3-5 seconds in each run.

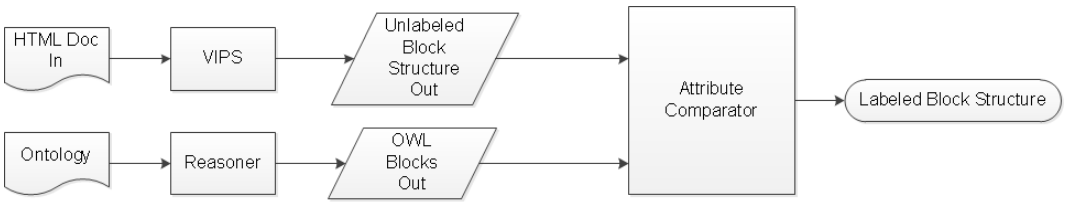

Figure 2: Architecture of the First Approach

2.2.4 Role Detection

In order to detect the roles of visual elements in web pages, we started with a simple approach but faced with some performance issues. In order to solve these issues, we followed an iterative approach in development. In this section, we describe each approach and some general technical terms related with them.

The first approach in heuristic role detection was based on direct string comparison between role characteristics and visual element attributes. The system architecture is represented in Figure 2 and its algorithm is given in Algorithm 1. Although this approach is very simplistic and easy to implement, it has caused some performance issues. Especially for large pages, string comparison is very inefficient and causes long response times.

In order to solve efficiency problem with direct string comparison, we proposed a second approach, in which we use a rule engine to detect roles of visual elements. We used Jess rule engine for this purpose, which is a Java based rule engine and scripting environment and can be integrated with our Java based implementation to ”reason” using role definitions and object properties in the form of declarative rules. One of its advantages is also its short response time. Jess uses an enhanced version of the Rete algorithm to process rules, which is a very efficient mechanism for solving the difficult many-to-many matching problem. This approach is represented in Figure 3 and can be summarised as in Algorithm 2. Jess has a LISP like syntax and each visual element and role definition with its characteristics must be interpreted in this syntax. The syntax of Jess is explained with examples.

In this approach, first we defined a template to describe the visual elements as facts in the working memory. A fact is the representation of a visual element in a web page. The collection these facts is called working memory. Every fact has a template, which defines its structure including its name and the set of its attributes. These attributes are represented

Algorithm 1: String Comparison Based Approach for Role Detection

```
input  : Unlabeled tree of visual elements and ontology
output: Labeled tree of visual elements
foreach visual element e in visual element tree do
    foreach role r in ontology do
        if attributesEqual(e, r) then
            increaseLikelihood(e, r)
        end
    end
end
```

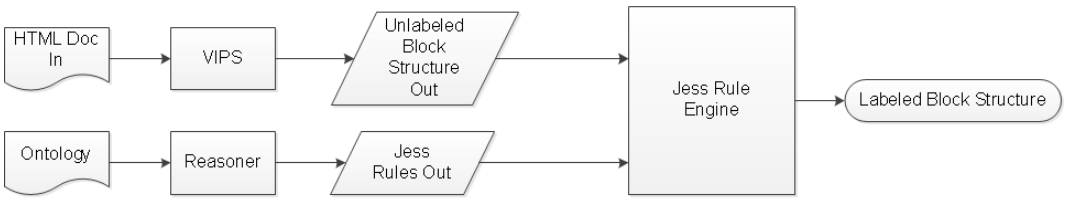

Figure 3: Architecture of Second Approach

as slots. Since each attribute (such as has_tag, has_child, etc.) may have more than one value, we used multislot instead of slot. A sample template definition for visual elements is as follows:

```
(deftemplate block
  (multislot has_id)
  (multislot has_keyword)
  (multislot has_tag)
  (multislot has_attribute)
  (multislot is_style_set)
  ...
)
```

In working memory, the likelihood scores are stored in a variable. Therefore, we need to define a variable for each role to indicate its likelihood score. In order to manipulate a defined variable in heuristic rule functions, its must be defined as global variable. The initial values of these global values were set to 0. Following code snippet illustrates an example global variable definition for *Header* role likelihood score:

```
(defglobal ?*Header* = 0)
```

After global variable creation, then we defined heuristic rules, which were previously defined in the ontology as restrictions describing the characteristics of a specific role. In Jess, rule definitions are very simple and typically in if...then... format. If an attribute of a visual element which is related with the corresponding rule satisfies the requirement in if part, the statement in then part is executed.

Algorithm 2: Rule Engine Based Approach for Role Detection

input : Unlabeled tree of visual elements and ontology
output: Labeled tree of visual elements
Create a working memory using ontology
foreach *visual element* e *in visual element tree* **do**
  `assert(`e`)`
  `getRoleWithMaxLikelihood()`
end

For example, following restriction on *Header* role indicates that, the tag of a *Header* element is more likely to be HEADER:

```
<owl:Restriction>
  <owl:onProperty rdf:resource="emine#must_have_tag" />
  <owl:hasValue rdf:resource="emine#Header" />
</owl:Restriction>
```

The rule corresponding above restriction is defined in Jess syntax as follows:

```
(defrule Header02
  (block (must_have_tag \$? /.*Header.*/ \$?))
    => (bind ?*Header* (+ "8" ?*Header*))
)
```

In this example, Header02 keyword denotes the unique name of the rule. The remaining consists of two parts. The first part, $(block\ (must\_have\_tag\ \$?\ /.*Header.*/\ \$?))$, specifies which types of object this rule is applied on and in which conditions the rule is fired. In this example, the rule is applied on $block$ objects when $must\_have\_tag$ attribute of the visual element contains HEADER tag. The second part, $(bind\ ?*Header*\ (+\ "8"\ ?*Header*))$, is the then part, and defines what happens when the condition in the first part is satisfied. In our example, the global variable which corresponds to the likelihood score of *Header* role is incremented by 8, which is the factor of $must\_have\_tag$ object property.

After defining the template for visual elements, the global variables for likelihood scores and heuristic rules for role characteristics, the next step is to apply these rules on visual elements in the working memory of the Jess. A working memory is the set of facts and a fact is the representation of a visual element in predefined template. In our case, the facts are based on a template, consequently, they are well structured. This type of facts are referred as unordered facts.

Finally, we need to reason on visual elements based on the rules we defined. In order to do so, following script is run in the working memory:

```
(assert
  (block
    (has_id header)
    (has_tag div)
    (is_composite 0)
  )
)
```

In this example, a simple atomic visual element, which has "header" id attribute and DIV tag, is inserted into the working memory. When we run the working memory, all of the rules, which visual element satisfies their first part, fire and their second part is executed. At the end of this set of executions, a set of global variables, each corresponding to a specific role in the ontology, is retrieved. Among these global variables, the one with the highest value is picked and the role corresponding it is assigned to the visual element. This assertion and firing process is repeated for each visual element in the web page. Unlike the string comparison for each attribute of each rule, rule engine respond much faster due to the Rete Algorithm.

In our second approach, template, global variables and heuristic rules were defined in each execution of the system. In order to retrieve the roles and their characteristics,

the ontology was also parsed. However, as explained earlier, we realised that, this is an unnecessary process to handle in each execution, rather we decided to store these definitions in a local file, and reconstruct it only if the ontology changes. Jess provides a file format, which is CLP, and all template, global variable and heuristic rule definitions can be stored in a CLP file. On the other hand, facts which are asserted to the rule engine are based on the visual elements in the segmented web pages and they are constructed in each program execution. After this enhancement, our system architecture had its final state, as represented in Figure 3.

Jess also enables to list the fired rules over an asserted fact. This list can be used in an explanation mechanism to give more detail about why a particular role is assigned for a visual element. We believe that, this can be a useful tool for designers and developers to see the results of their design in terms of heuristic roles and can be used to improve the design quality by validating a given web page.

In conclusion, direct string comparison is an inefficient method for detecting the heuristic role of a visual element, especially when the structural complexity of the web page increases. In order to provide more effective and efficient way, we have used Jess rule engine to reason on visual elements. In this section, we explained our development steps and technical terms of Jess rule engine.

3 Conclusion

This technical report presented our ontology-based heuristic approach to automatically identify visual elements in a web page with their roles. Due to the difficulty of fully describing visual elements with their characteristics, we proposed a probabilistic model. In this model, the role of the visual element is detected with respect to the number of its attributes which satisfy the requirements of a role. The main components of the proposed sistem are a visual element identifier, a knowledge base and a role detector module.

The most significant feature of the proposed approach is the ability to easily modify the knowledge base. Although a wide range of heuristic roles and attributes of visual elements have been analysed and covered, web design trends and patterns may change in the future. Also, new web technologies may arise in the future, and the proposed method is capable of handling such technologies. Finally, different tasks may require a narrowed or extended set of roles and different specifications for role attributes. Our proposed approach is easily modifiable and enables to adapt the changing technologies or web trends and to create task specific ontologies.

In conclusion, the research presented in this technical report contributes an effective method for identification of visual elements in web pages automatically. The findings of this research can be used in different fields including information retrieval, web accessibility, intelligent web user interfaces, web page transcoding or data mining.

A Role Definitions in eMine Ontology

eMine Ontology includes following role definitions. The construction of this role set is explained in Section 2.2.1.

- *Advertisement* is a graphical element which aims to direct users to an external page for commercial purposes.
- *Article* contains a set of paragraphs, especially the main content of the page.
- *Body* represents the root block of the page.
- *BreadcrumbTrail* is a list of links and each link directs to a page in the hierarchical structure of the web site. A highlighted link refers to current page.
- *Caption* is a short text for a table or a figure.
- *Citation* is a quotation or a reference to an external source.
- *ComplementaryContent* is a content element which is not the main content of the element, rather a supplementary content.
- *Container* is a composite visual element which contains smaller sub elements.
- *Copyright* is copyright note, usually located at the bottom of the page.
- *Figure* is a set of special graphics and caption objects.
- *Footer* is a container which is positioned at the bottom of the page.
- *Header* is a container which is positioned at the top of the page.
- *Icon* is a symbol for representing a tool or object in the page.
- *Interaction* consists of form items such as input box, button and combo box.
- *Label* is used with a form element to identify the form element.
- *Link* is used to navigate user to internal or external pages.
- *List* is an array of items, such as links or content.
- *Logo* is used to orient users to page itself, or external sources.
- *Menu* is a list of links for navigation purpose.
- *MenuItem* usually used to navigate users to internal pages.
- *SearchEngine* is used for searching a content. It usually contains a text editor, a button and labels.
- *Sidebar* is a container which is positioned on the left or right side of the main content.
- *Separator* is a separator text or image between two distinct visual element.

- *SpecialGraphic* is an image which has a special meaning in the content.
- *Table* is a tabular array which consists of a set of rows and columns.
- *Title* is an identification name, which is given to a section.
- *TitleBanner* represents the page title or logo.
- *Toolbar* is a list of tool icons.

B Object Properties in eMine Ontology

In order to describe the roles, we have used following object property definitions:

- *has_id* specifies the set of id, class, src attribute values which a role may have.
- *has_keyword* specifies the set of keywords which may appear in textual content of a role.
- *must_have_tag* specifies the tags which a role may have in high probability.
- *has_tag* specifies the tags which a role may have in normal probability.
- *may_have_tag* specifies the tags which a role may have in low probability.
- *has_attribute* describes the set of attributes which a role may have, such as onclick, src, and so on.
- *in_position* specifies the position of a role in the page, such as top or bottom.
- *has_child* specifies which children roles may have a role.
- *has_sibling* specifies which sibling roles may have a role.
- *has_parent* specifies which parent roles may have a role.
- *font_size* defines the font size relative to the body font size.
- *font_weight* defines the font weight relative to the body font weight.
- *border* specifies whether the role has borders.
- *font_color* defines if the font color of a role is the same with the body font color.
- *is_composite* specifies whether the structure of a role is composite.
- *has_list_style* defines if the role's list_style property is set.
- *has_background* specifies whether a role has background color or image.
- *has_order* specifies whether a role is in the given order among its siblings.
- *has_size* describes the exact size of a role.

- *word_count* specifies the word count of a role, such as long or short.
- *is_atomic* specifies whether the structure of a role is atomic. It is the inverse of is_composite property.
- *relative_size* describes the size of the role as wide or high, rather than numerical value of its sizes.
- *doc* is related with VIPS Algorithm and specifies after which block extraction rule the visual element is constructed.

References

[1] Elgin Akpinar and Yeliz Yesilada. Vision based page segmentation: Extended and improved algorithm. Technical report, Middle East Technical University Northern Cyprus Campus, 2012.

[2] D. Cai, S. Yu, J. R. Wen, and W. Y. Ma. Vips: a vision based page segmentation algorithm. 2003. Technical Report MSR-TR-2003-79, Microsoft Research.

[3] Yu Chen, Xing Xie, Wei-Ying Ma, and Hong-Jiang Zhang. Adapting web pages for small-screen devices. *IEEE Internet Computing*, 9(1):50–56, January 2005.

[4] James Craig and Michael Cooper. Accessible rich internet applications (WAI-ARIA) 1.0. http://www.w3.org/TR/2010/WD-wai-aria-20100916/complete, 2010. retrieved on 15.01.2013.

[5] T. Eiter, G. Ianni, A. Polleres, Roman Schindlauer, and Hans Tompits. Reasoning with rules and ontologies. In *In Reasoning Web 2006*, pages 93–127. Springer, 2006.

[6] Simon Harper and Yeliz Yesilada. Web authoring for accessibility (WAfA). *Journal of Web Semantics (JWS)*, 5(3):175–179, 2007.

[7] Milos Kovacevic, Michelangelo Diligenti, Marco Gori, and Veljko Milutinovic. Recognition of common areas in a web page using visual information: a possible application in a page classification. In *Proceedings 2002 IEEE International Conference on Data Mining*, pages 250–257, Washington, DC, USA, 2002. IEEE Computer Society.

[8] Peifeng Xiang and Yuanchun Shi. Recovering semantic relations from web pages based on visual cues. In *Proceedings of the 11th international conference on Intelligent user interfaces*, pages 342–344. ACM, 2006.

[9] Yeliz Yesilada. Web page segmentation: A review. March 2011.

[10] Yeliz Yesilada, Giorgio Brajnik, and Simon Harper. Barriers common to mobile and disabled web users. *Interacting With Computers*, 23(5):525–542, 2011.

[11] Xinyi Yin and Wee Sun Lee. Understanding the function of web elements for mobile content delivery using random walk models. In *The 14th International World Wide Web Conference*, pages 1150–1151. ACM, 2005.