

The use of ontologies in wrapper induction

Marek Nekvasil

Department of Information and Knowledge Engineering, University of Economics Prague,
Winston Churchill Sq. 4, 130 67, Prague 3, Czech Republic
nekvasim@vse.cz

Annotation: The purpose of this entry is to bring in an extension of advanced knowledge models, known as ontologies, so that they can be utilized in the process of automated information extraction from the web documents. Major part of it is dedicated to a proposition and derivation of an inference model, based on principles of fuzzy logic, for evaluation of the pattern matches and their combination into templates of typical values of datatype properties of ontology classes. Further is proposed a simple method of wrapper induction which is able to utilize the results of automatic document annotation with the proposed templates.

Keywords: ontology, automatic annotation, information extraction, wrapper

1 Introduction

The vast majority of information resources available on the internet is designed for processing by humans. Formats that are used for visualizing data (e.g. HTML) or handling the users input (e.g. HTML forms) fully agree with this purpose. However, it isn't always appropriate to handle information from available resources by hand. There is an alternative to the manual handling, an automatic handling, by which we mean the use of a computer program instead of human in the interaction with information resources. One of the most simple kinds of such programs are the *wrappers* which are designed to extract specific values from a specific set of documents and in most cases build upon the document structure.

Basically a wrapper is a set of rules that can be used to identify the desired values in the document. These rules can be created by hand or we can use for their creation the methods of machine learning, in which case we are talking about *wrapper induction*. For more information about wrappers and their induction see e.g. [5]. For automatic wrapper induction in principle some examples of the real data to be extracted are needed along with their respective context, which form basically an annotated document. The purpose of this entry is to propose a concept by which it should be possible to annotate the documents automatically with the use of *ontologies*.

From the formal point of view (see [2]) we can consider the ontology to be a certain kind of conceptual data model, which apart from the definition of object classes, their relationships and restrictions contains also information about where and in what circumstances can these object occur and their properties take values. It is clear that if it was possible to specify this additional feature precisely enough it would be possible to utilize a properly formed ontology not just to represent the meaning of data but moreover to identify it. In an ideal case it could be possible, with such an extended ontology, to locate instances of particular class in unstructured and semistructured

documents such as web pages. We will call any ontology that is designed to this use in information extraction an *extraction ontology*.

2 Extending an OWL ontology

An ontology written in a bare OWL language, in any of its versions, has very limited capabilities of describing the possible values of datatype properties (properties which's value is as literal) and therefore does not contain enough information to identify these inside a document.

To remove this insufficiency we introduce an extension which will enable us to append a pattern of typical values to each datatype property. In XML transcription of OWL ontology we will denote the patterns by elements from a special namespace (we use the namespace "op" as a shortening of "ontology pattern").

The patterns will be associated with a datatype property in XML notation in this way:

```
<owl:DatatypeProperty rdf:ID="property_id">
  <rdfs:domain rdf:resource="#resource_id"/>
  <op:SomePattern ... >
  ...
</op:SomePattern>
</owl:DatatypeProperty>
```

3 The patterns

First of all let us introduce what we mean by a pattern with respect to datatype properties' values and what kinds of patterns there will be.

Any general rule for which it is possible to determine on any continuous part of a document (in terms of a string of characters or words) whether it is satisfied or not (or to what extent) we call a *pattern*. An example of a pattern can be a rule that evaluates whether a given part of a document is a number from a given range or whether it is a string from a given list.

From now on we will distinguish between two kinds of patterns. Foremost there will be simple patterns, or rather *atomic patterns* that will be formed by a simple rule, which can be directly evaluated on a part of a document. An example of atomic patterns can be the aforementioned patterns that match a number or a string.

In addition to these atomic patterns there will be *composite patterns*, which we will define as such a combination of rules that can be evaluated on a given continuous part of a document as a whole. A combination of the two mentioned patterns could form a composite pattern. (For example a rule combination that matches a number in the interval <100; 500> in the vicinity of a string "CZK" can make a composite pattern for property "price" of some product.) As such, the composite pattern will be always a combination of other patterns that can be either atomic or composite. Therefore the composite patterns can be hierarchically combined which can be of significant concern in some specific situations.

As it has sense to assign only one pattern per datatype property, more patterns will have to be joined via including them in some composite pattern. Every part of a document that matches a given pattern, i.e. the pattern rule evaluates positively on that part, will we consider a suspected *partial candidate* for the occurrence of the value of datatype property the pattern is assigned to. If that given pattern is the one that is assigned directly to the datatype property, every matching part of the document will be considered to be a suspected *candidate* for the occurrence of the value.

3.1 Atomic Patterns

In the process of evaluating the match of atomic patterns we encounter the problem of deriving the certainty degree of marking the candidate for the value of a certain datatype property from some other inputs. In this point of view we take a pattern as a certain kind of algorithm that can tell for every place in the document to what extent the rule is satisfied depending on its parameters.

Here we have two distinct terms. First of them is the *degree of pattern match* which represents the certainty with which the pattern's algorithm marked the given place in the document. The second term is then the *certainty degree of marking the partial candidate for the value of a certain datatype property* which represents the certainty that the given place in the document really is the occurrence of the value of a datatype property of an object of the extracted class, given sole by the observation of the single pattern and independently of any other patterns. In this manner we will denote the marking the partial candidate for the value of a certain datatype property as the *pattern evidence* and therefore the second term will be equivalent to a *degree of pattern evidence*.

The degree of pattern match and the degree of pattern evidence should intuitively correspond, namely in the way that it should be possible to infer from a high degree of match to a high degree of evidence. If we denote the pattern match as A and the pattern evidence as E we can formally rewrite the rule mentioned above in such a way:

$$A \rightarrow E \tag{1}$$

A rule written in this manner can be considered a form of inference. For our purpose we have chosen a model of fuzzy logic inference (see [3]), however it should be possible to use any other inference model.

While using the fuzzy logic we can define A and E as logical propositions:

- A – “The pattern has marked the given place in the document.”
- E – “The marked place is really a pattern evidence”

The corresponding degrees will then be the truth values of these proposition (i.e. the degree of pattern match $a = val(A)$ and the degree of pattern evidence $e = val(E)$, where $val()$ is a formal function that maps a proposition to its truth value from the $\langle 0, 1 \rangle$ interval).

For better flexibility of our inference model we now introduce two universal parameters that will be assigned to every pattern, namely *precision* and *cover* and we define them in this way:

$$p = \text{val}(A \rightarrow E) - \text{pattern precision} \quad (2)$$

$$c = \text{val}(E \rightarrow A) - \text{pattern cover} \quad (3)$$

While using these parameters we can derive on Łukasiewicz fuzzy logic this form of the inference rule (the detailed derivation is available in [8]):

$$((A \& (A \rightarrow E)) \vee \neg(E \rightarrow A)) \Rightarrow E \quad (4)$$

If we take into account the functions of used connectives in Łukasiewicz fuzzy logic that we used for deriving this relation along with our definitions of the precision and cover parameters we can write down for the degree of pattern evidence this mathematical unequation

$$e \geq \max(\max(0, a + p - 1), 1 - c) \quad (5)$$

To avoid inconsistencies in our inference model we should not overestimate the degree of pattern evidence e , and therefore we will use its minimal value possible from (5). Thus we get

$$e = \max(a + p - 1, 1 - c) \quad (6)$$

If we analyze this simple rule we find out that the value of parameter c determines the minimum degree of pattern evidence that can be assigned to any place in the document. Therefore it is no good to set the values of c too low because in that case the pattern loses its meaning. If we set for example $c = 0$ every place in the document will be marked an evidence with the certainty degree 1 which practically has no sense, however it is a logical consequence of the proposition that the pattern never marks an assumed evidence, which is exactly what $c = 0$ says. For the resulting degrees of evidence to have sense the unequation $c > 1 - p$ has to be satisfied. In case this unequation isn't satisfied any degree of pattern match cannot have influence on the final degree of evidence. That is because from the relation (6) we can directly derive:

$$e \in \langle 1 - c, p \rangle, \text{ for } 1 - c < p, \text{ else } e = 1 - c \quad (7)$$

With the use of parameter p we can set a top limit to the degree of a given pattern evidence. The pattern precision denotes in this context a certainty with what the high degree of pattern match leads to a high degree of pattern evidence.

We can think of other uses of these parameters (some of which we will show in the next part). For example while with parameter c the minimum degree of pattern evidence is set we can ignore any evidences that would have otherwise a lower degree of certainty, thus effectively reducing the absolute number of evidence taken into

account which can have a positive influence on the total computational complexity of this task.

The cover and precision parameters can further be used in the wrapper induction itself and in automatic estimation of the trustworthiness of the induced wrapper which we will show in the last part.

The p and c parameters have been introduced to all of the patterns, however for easing the construction and notation of the patterns in an ontology the namely declaration of these parameters doesn't have to be required and a default value of 1 for both of them can be assumed.

3.2 Composite patterns

In the theory of fuzzy sets the inference rules can be considered a projection into a fuzzy set. From this point of view we can consider the degree of pattern evidence a degree of classification of this part of a document into the fuzzy set of all occurrences of the particular datatype property values.

Would we like to combine the evidences of multiple patterns it will be basically a task in form of a set operation. In principle for this purpose come on force just two set operations, namely union and intersection operations which in combination with the single set complement operation can form any set operation possible. The use of union and intersection on multiple sets correspond with the use logical operations conjunction and disjunction on the individual set elements, i.e. pattern evidences (and the complement operation corresponds with the use of negation).

To express a combination of partial patterns, as mentioned before, we will use the composite patterns. Therefore we will have to introduce the composite patterns of two kinds; according to the appropriate logical operation used will there be conjoint patterns and disjoint patterns (and furthermore possible negating patterns).

The determination of the degree of composite pattern evidence itself is then a trivial matter. The degree of composite pattern match will be determined by simply assembling the degrees of evidences of the partial patterns the composite one is made of with the appropriate logical operation. In case of a conjoint pattern we will determine the pattern match in this way

$$A \Leftrightarrow (E_1 \wedge E_2 \wedge \dots \wedge E_n), \quad (8)$$

where E_i for $i = 1..n$ is the evidence of the i -th partial pattern. For disjoint composite patterns we express the pattern match likewise:

$$A \Leftrightarrow (E_1 \vee E_2 \vee \dots \vee E_n). \quad (9)$$

From the pattern match defined in this way we get the composite pattern evidence by absolutely the same way as we did in case of atomic patterns. In the course of numeric determination of the truth values (and thus the degrees of matches) we will use the function definitions of universal min-conjunction and max-disjunction.

It could be interesting in this place to discuss the meaning of precision and cover parameters of the partial patterns in contrast with the use of the different kinds of

composite patterns. If the parameter p of the partial pattern is lower than 1 , the maximum value the degree of pattern evidence lowers as well. In case of a disjoint composite pattern the high value of p implies that matching the given partial pattern is a sufficient condition for matching the composite pattern. Any partial pattern with a value of c set to low will then in this case supersede any other patterns which is totally meaningless a should be avoided.

On the other hand while the cover parameter determines the lowest possible value of degree of partial pattern evidence it determines also how the partial pattern under consideration can limit the outcome of a conjoint composite pattern. Therefore in case of conjoint composite patterns the high value of c means that matching the partial pattern is a necessary condition for matching the composite pattern. Too low precision parameter of any partial pattern can neglect any other patterns in this case and therefore should also be avoided.

Interesting is the fact that in case of conjoint composite patterns the precision parameters of partial patterns strengthen each other and therefore the effective maximum degree of the composite pattern match is given by the lowest of them. In case of disjoint composite patterns it is vice versa and the effective minimum degree of composite pattern match is given by the lowest of covers.

While designing the patterns it is therefore suggested to carefully take into account the possible effects of these parameters in contrast with the used kind of composite patterns.

4 Designing the patterns

In the extraction ontology we will write the patterns down in the way mentioned in the beginning, that is in the form of XML elements from a special namespace. Type of the pattern will be given by the name of the XML element while its parameters will be given by the elements attributes or nested elements. In case of composite patterns the partial patterns will always be written as the nested elements.

Here we provide a few of the possible patterns, more of them are proposed in [8].

4.1 The template <ot:Template>

One particular pattern that has been mentioned to some extent is this “Template” pattern which we will use to denote any composite pattern. This pattern will be represented by a XML element <ot:Template> and its nested elements would represent the specific partial patterns. There will be an additional attribute *ot:type* which’s value will determine the kind of the composite pattern. Feasible values of this attribute will be “disjoint” and “conjoint” (and “negating” in which case there will be only one nested element allowed), with the default value of „disjoint“.

The composite pattern written down will then look like this:

```
<ot:Template ot:p="0.95" ot:c="0.8" ot:type="disjoint">
  ...
</ot:Template>
```

4.2 String <ot:String>

This string pattern will represent the occurrence of a simple text value. There will be a single parameter of this pattern, namely the text value which's occurrence in the document will cause the pattern to match. This parameter will be given as the content of the element and therefore its notation will take this form:

```
<ot:String ot:p="0.7">Egypt</ot:String>
```

Of course other attributes are thinkable in this case, for example for enabling a restriction on the case of characters.

4.3 Other patterns

In principle we have shown the system of designing the particular patterns on the example of a string pattern. By a similar way it is of course possible to design countless other patterns according to the needs of datatype attributes, for example there could be patterns for evaluation of regular expressions or for comparing values according to statistical distributions (that and more are proposed in [8]). The extent of the patterns can vary from distinguishing time values, named entities to patterns that evaluate the context or format of the document.

While designing a new pattern it is needed to keep in mind the way it evaluates and think carefully the possibilities of its combination of other existing patterns.

Here we provide an example of the extraction ontology designed for extracting the prices and destinations of tours:

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf=. . . >
  <owl:Ontology rdf:about="" />
  <owl:Class rdf:ID="Tour" />
  <owl:DatatypeProperty rdf:ID="price">
    <rdfs:domain rdf:resource="#Tour" />
    <ot:Template ot:type="disjoi nt" ot:c="0.9">
      <ot:Concatenati on>
        <ot:Distri buti on ot:type="gauss" ot:mean="10900"
          ot:vari ance="9200000" />
        <ot:Stringl i st>
          <ot:String ot:case="any">CZK</ot:string>
          <ot:String ot:case="any">kč</ot:string>
          <ot:String ot:case="same">,-</ot:string>
        </ot:Stringl i st>
      </ot:Concatenati on>
    <ot:Context ot:si de="l eft" ot:maxdi stance="2" ot:p="0.6">
      <ot:Templ ate>
        <ot:String ot:case="any">pri ce</ot:string>
        <ot:String ot:case="any">pri ce:</ot:string>
      </ot:Templ ate>
    </ot:Context>
  </ot:DatatypeProperty>
</owl:DatatypeProperty rdf:ID="desti nati on">
```

```

<rdfs:domain rdf:resource="#Tour"/>
<ot:Template ot:c="0.8" ot:p="0.6">
  <ot:StringList ot:source="countries.txt">
</ot:Template>
</owl:DatatypeProperty>
</rdf:RDF>

```

5 A simple wrapper induction method

By applying the rules of available patterns on the content of a document we get a set of evidences along with their certainty degrees for every datatype property in the extraction ontology that has assigned a pattern. For each and every evidence we are able to get its absolute XPath expression with element indexes. It would be of course possible to use some more general approach (as in [1]), however we restrict ourselves to tabular structure of data to be extracted and therefore this rather naïve approach will do.

5.1 Segmentation and purging of evidences

Because we rely on tabular structure of data we can try to separate the evidences in a few segments according to the resemblance of their XPath. The segmentation itself will then be done in a way shown by this pseudo-code:

```

SegmentEvidences (array of evidences E)
  S = array of segments
  for every evidence e1 ∈ E
    for every evidence e2 ∈ E, e2 ≠ e1
      if XPath of e1 and e2 differ by just one index
        if there is no s ∈ S for XPath of e1, e2
          create segment s for XPath of e1, e2
          add e1 and e2 to s
          add s to S
        else add e1 and e2 to s, if they aren't already contained
  return S

```

We can see that the individual segments can overlap. In that case one evidence can be contained in more segments.

The next step in our simple induction algorithm will be the purging of evidences sets. If the pattern did have the precision attribute specified we can afford a small formal inaccuracy and declare that this number specifies the mean ratio of evidences that are marked correctly by this pattern. Therefore we can say that up to $1-p$ of evidences supplied by this pattern can be false. If we sort the segments by their total share we can discard one by one from the smallest as they are probably misleading until the ratio of the discarded reaches $1-p$ of the appropriate pattern. The segment which's discarding would surpass this value is sustained because it could be the last or the only one. The ratio itself can be calculated from the absolute sizes of the segments or

even better from the sum of degrees of evidences. The purging of evidences can be shown by this pseudo-code:

```
PurgeEvidences (array of evidences E, pattern precision p)
  S = SegmentEvidences(E)
  P = sum of degrees of evidence of  $e \in E$ 
  for each segment  $s \in S$ 
    s.P = sum of degrees of evidence of  $e \in s$ 
  for each  $s \in S$ , sorted by s.P, start with the smallest
  until  $(s.P + X) / P \leq p$ 
    X = X + s.P
    discard s and continue with next segment
  return S
```

5.2 Generalizing XPath expressions

The second step in the induction algorithm is the generalization of the XPath expressions, firstly for every datatype property and finally for the instance of the extracted class as a whole.

If the data are stored in the tabular structure the relevant parts of text are generally contained in the same structure of elements that is not changing throughout the document. On the level of XPath expression this will show up as a single changing index in the absolute path and most probably we can rely that individual instances are distinguished just by this index. Within a single segment of evidences we can generalize the XPath expression by temporarily omitting the changing index by which we will get a set of elements that would ideally all contain the value of the respective property.

5.3 Estimation of the generalization error

By generalizing and evaluating the XPath expression we got a set of elements that is greater or equal to the original segment of evidences. The cover parameter of the corresponding pattern was defined as the certainty degree that the pattern will identify every value of the property. Again with some small formal inaccuracy we can say that it is the mean rate of the evidences that the pattern identifies to the total real number of occurrences of the respective property.

While the generalized XPath expression should identify all possible occurrences of the property we can calculate the proportion of evidences of the pattern to this "complete" set and the number should not differ much from the parameter c of the pattern. If this proportion is lower than c , it means that by the generalization we probably included something we did not intend to. The average of these differences over all segments can represent the total error caused by generalizing the paths.

5.4 Forming the instances

Generally we can assume that in this step we have for every property some number of generalized sets of evidences each of every can have a different number of elements. We can assume that while we rely on the tabular structure of the data the instances will be made of n-tuples of values and therefore the number of evidences in the segments of corresponding properties has to be the same (and also the number of segments for each property).

The first criterion for assigning the corresponding segments of different properties will thus be the number of the evidences it contains. In case of more solutions we can use the second criterion of assigning the segments by the resemblance in the beginning of the XPath expression. If there is no solution even with use of both of these criteria we are regrettably not able to assign the corresponding values and therefore cannot extract whole instances.

The result of this assigning should be the set of n-tuples of generalized XPath expressions where n is the number of datatype properties the extracted class has. The particular algorithm for assigning the segments can be expressed in this way:

```
AssignSegments (array of sets of segments by pattern PS = {S1 .. Sn})
  V = array of segment groups
  for each combinations of segments s1 ∈ S1, s2 ∈ S2 .. sn ∈ Sn
    if number of evidences e(s1) = e(s2) = .. = e(sn)
      v = group of segments s1, s2 .. sn
      add v to V
  for each segment groups v1 ∈ V
    for each segment groups v2 ∈ V
      if (number of elements el(v1) = el(v2))
        and (there is an overlapping element in both v1 and v2)
          remove from V the group with greater differences in
          generalized XPath expressions
  return V
```

To sum up this approach is just a simple method and has many limitations, but if the patterns are set well it can work totally automatically and report a suspected error of the extraction (given by the estimation in the generalization step). Besides that this method can extract only properties with cardinality 1 (the tabular structure) it is also limited in its tolerance to the irregularities in the structure of the document, on the other hand to the irregularities in the extracted values it is rather resistant.

6 Conclusion and future work

The proposed method of pattern notation allows hierarchical combining of partial patterns and is open to the possibility of designing additional patterns according to one's need. The patterns are evaluated independently of the structure or format of the document. Similar approach is taken by [6] and [7], however unlike them we do not design proprietary formats of ontologies but try to start from OWL standard. The

annotation method is proposed with respect to allow its use with other existing wrapper induction methods that could be an inspiration of future work.

The limitation of the proposed wrapper induction method is the fact that it relies on the tabular structure of extracted data and on the possibility to construct a DOM tree over the document and thus use the XPath expressions so that the document has to be in HTML or even better XHTML format. The advantage of this approach is that the extraction is completely automatic and with proper setting of the attributes allows the estimation of the error in the course of extraction.

The interesting subject of future work could be to propose a way of automatic learning of the patterns or at least of their parameters or to inspect the possibilities of using the additional ontology properties, such as the cardinality restrictions.

Acknowledgement

The research leading to this paper was supported by the European Commission under contract FP6-027026, Knowledge Space of semantic inference for automatic annotation and retrieval of multimedia content - K-Space.

Reference

1. Anton T.: *XPath-Wrapper Induction by generalizing tree traversal patterns*, in: *Wissensentdeckung und Adaptivität, 2005*, www.ke.informatik.tu-armstadt.de/lehre/diplomandenseminar/fohlen/anton-lwa05-revised-2006-02-15.pdf
2. Antoniou, G., van Harmelen, F.: *A Semantic Web Primer*, Cambridge MA.: MIT Press, 2004, ISBN 0-262-01210-3
3. Hájek P.: *Metamathematics of fuzzy logic*, Dordrecht: Kluwer, 1998, ISBN: 0-792-35238-6
4. Chang, Ch. H., Hsu, Ch. N., Lui, S. Ch.: *Automatic information extraction from semi-structured Web pages by pattern discovery*, *Decision Support Systems*, Amsterdam: Elsevier Science Publishers B. V., 2003, ISSN:0167-9236
5. Kushmerick, N.: *Wrapper induction for information extraction*, PhD thesis, University of Washington, 1997, Department of Computer Science and Engineering
6. Labsky M., Svatek V.: *On the Design and Exploitation of Presentation Ontologies for Information Extraction*, ESWC'06 Workshop on Mastering the Gap: From Information Extraction to Semantic Representation, Budva, Montenegro, 2006
7. Muslea, I., Minton, S., Knoblock, C.: *A Hierarchical Approach to Wrapper Induction*, *3rd Conference on Autonomous Agents*, 1999, <http://www.isi.edu/~muslea/papers.html>
8. Nekvasil M., *Využití ontologií při indukci wrapperů*, graduation thesis, UEP, 2006