Automated Ontology-Driven Metasearch Generation with Metamorph

Wolfgang Holzinger[1], Bernhard Krüpl[1], and Robert Baumgartner[2]

[1] Vienna University of Technology, Institute of Information Systems, Database and Artificial Intelligence Group (DBAI)
{holzing,kruepl}@dbai.tuwien.ac.at
[2] Lixto Software GmbH, Vienna, Austria
baumgartner@lixto.com

Abstract. We present Metamorph, a system and framework for generating vertical deep Web search engines in a knowledge-based way. The approach enables the separation between the roles of a higher skilled ontology engineer and a less skilled service engineer, which adds new web sources in an intuitive, semi-automatic manner using the proven Lixto suite. One part of the framework is the understanding process for complex web search forms, and the generation of an ontological representation of each form and its intrinsic run-time dependencies. Based on these representations, a unified meta form and matchings from the meta form to the individual search forms and vice versa are created, taking into account different form element types, contents and labels. We discuss several aspects of the Metamorph ontology, which focuses especially on the interaction semantics of web forms, and give a short account of our semi-automatic tagging system.

1 Introduction

Most content on the Web is still in a format that is intended for human beings and is not per se machine readable. The Lixto suite [1] is a commercial product that can be used to extract this content and transform it into a form more suited for automatic processing. Lixto Visual Developer (VD) wrappers are created in an visual and interactive manner by marking relevant content on a web page, and they use Elog and XPath expressions to reference and extract from the relevant pieces.

In this paper, we present the Metamorph system, which extends the hidden web capabilities of Lixto VD for generating vertical deep Web search engines, i.e. engines that search in parallel on a number of web search forms of a particular (vertical) domain. Metamorph provides means to identify, reference, and manipulate web search forms in a knowledge based manner; the knowledge is split into a general web and a domain specific part, is formalized in RDF/OWL plus a set of rules and is capable of covering all kinds of mappings that are necessary to map web forms. Rather than mapping web form elements to those of a central or other remote web forms, Metamorph acknowledges the fact that modern web

© Springer-Verlag Berlin Heidelberg 2009

forms can be regarded as mini applications, because they can contain temporal and other dependencies. Consequently, Metamorph goes beyond traditional schema mapping and models web form interactions, i.e. the whole interaction sequence a user of web forms has to perform in order to reach his goal.

1.1 Related Work and Comparison

Unlike many academic and commercial approaches to web scraping, this work focuses primarily on understanding web forms and deep web navigation. In the area of structured data extraction, a number of well-known methodologies have been proposed, eg. Roadrunner [4] for automatic extraction, Dapper [6] in the area of machine learning; interactive approaches have been proposed by mashup enablers such as Intel Mashmaker [13] or Lixto Visual Developer [1]. Our work is orthogonal in many aspects; it is more focused on understanding the flow logic and the interactions of web applications than on extracting from (more or less) static pages.

In [2][3][11][12] tools and methodologies for web data extraction are compared with each other. These tools offer formidable data extraction and integration solutions. However, most such tools lack the support for meta-search specific problems such as form understanding, a workflow-based extraction from multiple sources, techniques for selecting relevant sources, and sometimes even technical issues such as load balancing. On the other hand, in the area of information integration on the Web several mashup assembly platforms have been created including e.g. Yahoo Pipes [17] and IBM Mashup Center [10]. The DeLa system sends queries to HTML forms after having annotated their elements [16]. Embley et al. describe the usage of tokenization techniques for web data extraction [7].

There are also fully automatic approaches: The University of Illinois and SUNY Binghampton develop interesting approaches and prototypes for discovering web data bases, generating search interfaces for particular domains, and data extraction [14]. The WISE-Integrator [8] uses positive matches and predictive matches to build attribute clusters and applies a majority rule to choose global attribute names. The MetaQuerier system [9] automatically extracts and query interfaces from websites by using just form element labels and matches them. Their application follows a generic approach to match any two search interfaces. The DEQUE system [15] models and wraps deep web search forms and result pages.

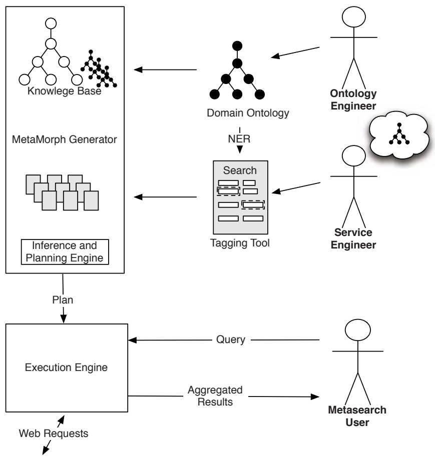
In contrast to the presented literature, Metamorph firmly relies on a domain ontology that has to be created in advance, but enables higher precision form extraction and mapping because attribute values can be considered as well. A distinguishing point from other systems is that Metamorph considers the integration of web forms as more complex than schema matching. Modern web forms often have dependencies between their controls that are expressed in Javascript code (filling out one field changes the options of another field). Metamorph takes this into account by modelling the interactions a user would need to take to formulate a request, and by constructing an interaction sequence for automatic submission. We also believe that the interpretation of query results is essential to any meta-search framework, and that a feedback loop has to be established, which can improve the domain knowledge quality with every request. The Metamorph framework employs Lixto wrappers for the acquisition and transformation of website query results.

2 Overview of the Metamorph System

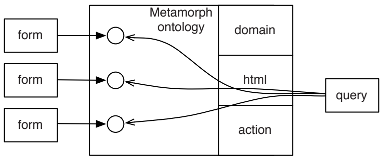
Figure 1a shows the main components of the Metamorph system. The central part is the Metamorph generator (shown on the left side) that holds the knowledge base and the inference and planning engine. For every metasearch domain like air travel, hotel search etc. an ontology engineer has to create a domain ontology that is integrated into the knowlege base.

In each domain, there exists a number of search forms on the Web. These forms are tagged and annotated by a service engineer with the help of a semi-automatic tagging tool: Known domain concepts are automatically tagged and can be reviewed and enhanced by the service engineer on a visual level. After that, all annotations are converted into RDF instances and stored in the generator knowledge base.

Whenever the service engineer adds a new form description to the knowledge base, the Inference and Planning Engine compiles a plan that is ready to run inside the Execution Engine, and that is fully independent of the generator. The plan contains information on how to translate any metasearch query that can be posed by a user back into interaction sequences that are compatible with the original web form. Finally, users can pose queries to the execution engine by filling out a metasearch form. The execution engine executes all the plans on all remote websites and aggregates the results and presents them to the user.



(a) Metamorph system and users



(b) Query mapping via ontology

Fig. 1. Metamorph system overview

In the following sections, we will provide some details about Metamorph's central knowledge base and discuss how web page forms are tagged in a semi-automatic way.

3 The Metamorph Ontology

Our ontology is structured as outlined in figure 1b. Arbitrary web forms are represented in the Metamorph RDF model. A meta-query request posed to the Metamorph system is translated into a query on this model and results in an interaction sequence for all web forms represented in the model. We use RDF/OWL models to formalize the Metamorph ontology, enhanced by rules as provided in the Jena toolkit.

Each search form on a website has its own idiosyncratic way for representing a search request. To capture the specific semantics for such a form, the Metamorph ontology must be able to represent three aspects:

– *html:* The technical details how the local search form is constructed, which HTML input elements were used in which combination and their positions on their source page for further reference.
– *domain:* A domain model rich enough to annotate each semantic reference in the local search form. For our flight–search example, this includes concepts for time and date, geographical locations, passengers and reservations.

Table 1. User interaction primitives

`PICK(x)`	select option x in a choice field
`CLICK(x)`	click on element x
`ENTER(s,x)`	enter string s into textfield x

– *action:* Each HTML form element has an intrinsic action associated with it; a button can be clicked, a string value can be fed into a text field (see table above) Performing the right kind of actions in the correct order will successfully submit a query to the deep web information source.

Interactions are subject to various constraints. Some forms require that the elements in a form are filled in in a certain order; there may be alternatives to fill in certain values (for instance, the user can either use a date picker control or fill in the date in a text field, but not both); the last action to take after filling in all fields is to press the submit button. When constructing the action sequence all these contraints have to be fulfilled.

Example: Suppose there is an select box containing month names. During tagging, the relation from pure text strings to the Month concept has been established. The interaction semantics for each option can be expressed as

```
pick(o1) achieves fillin(monthJanuary)
pick(o2) achieves fillin(monthFebruary)
...
pick(o6) achieves fillin(monthJune)
```

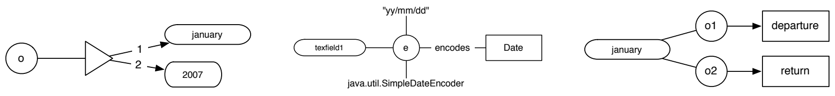
A request posed to the Metamorph system involving the month June will now derive that the corresponding action is to pick the item `o6` (a HTML option element). Other fields in the request will yield similar actions. To fully resolve the request we pick from all those generated actions a sequence that is consistent with all constraints that are in effect.

We can take this translation a step further. With the semantic information gathered in our entangled ontologies, we can actually devise a plan for mapping the query *schema*. This plan can then be translated to a scripting language and executed independently of the resource intensive reasoning framework:

```
if (month == january)  pick(o1)
if (month == february)  pick(o2)
...
if (month == june)  pick(o6)
```

Note that this script does not any more refer to any internal ontology details. It is purely end-to-end from the global search parameters to the interaction language.

Modeling one-to-many mappings. Sometimes it is necessary to map from an input element to a combination of domain concepts, forming an 1:n relation. Straightforward modeling this with multiple mapping relations is undesirable, because it requires higher-order predicates to reason about these situations. Fortunately, in all cases we studied, the arity of the one-to-many mappings was a constant. Therefore we make this explicit by introducing fixed-arity Tuple concepts. In figure 2a we see the common case where a form element matches a combination of month and year. This combination is made explicit by introducing an individual of class `Pair` and the relation `first-element` and `second element`. In the same manner we can define triples, quadruples and higher order combination concepts.



(a) one-to-many mapping (b) encoder (c) roles

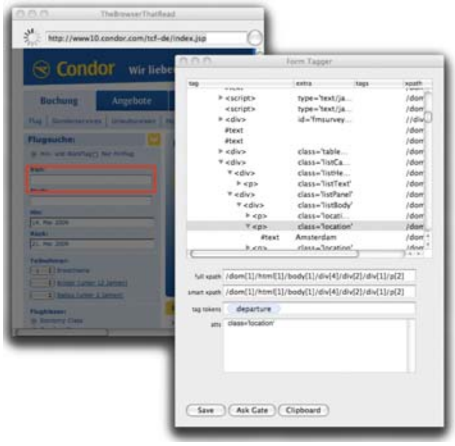
Fig. 2. Metamorph design patterns

Concept-to-text mapping with encoders. Text fields allow no unambiguous translation of domain concepts to form elements, since it is not clear how to compute an appropriate string representation of a concept. We introduce the notion of *encoders*, external functions that will translate a concept to a fitting string representations. A encoder is a special kind of an external functions whose range is always a string. In figure 2b we see an text field matching an encoder *e*, mapping from the concept `Date` to a string, using the encoding `yy/mm/dd`. *e* has a reference to a Java class wrapping up the actual code.

Modeling one-to-many mappings with roles. Sometimes concepts in the request cannot be distinguished by their type alone. To identify them unambiguously, we allow roles to be assign to input concepts. In figure 2c, two form options refer simultaneously to the single individual `january`. The first option, `o1`, is used by the form to describe the desired departure date for a flight, `o2` is used to describe the date of the return flight. By tagging each option with a role, `return` and `departure` respectively, we can capture the different contexts these two individuals stand in. Although both options match an identical concept, they can be distinguished by also taking their roles into account.

4 Tagging Web Forms

The discovery and tagging of relevant forms on a web page takes place in three stages: i) the page is preprocessed with the Mozilla-based Lixto browser component to ensure its consistency and validity; ii) automatic taggers based on named entity recognition, domain grammars and machine learning are deployed on the page to identify forms and their constituents; iii) the tags are revised by a service engineer to ensure the best possible mapping quality. The annotations are currently realized as XPath bound RDF triples; for greater flexibility, we plan to switch to an inline RDFa representation. We adapted the Lixto Visual Developer for the interactive, semi-automatic tagging of web forms. Lixto VD is a Java based wrapper generation tool with a visual user interface that can be used to automatically record a navigation sequence; to parametrize it; and to replay it later by exececuting mouse and key actions and collecting results. The screenshot in Figure 3 shows how a service engineer uses Lixto VD to mark relevant web form elements.



http://www10.condor.com/tcf-de/index.jsp

tag	extra	tags	xpath
▸ `<script>`	type='text/ja...		/dom
`<script>`	type='text/ja...		/dom
▸ `<div>`	id='fmsurvey...		//div
#text			/dom
#text			/dom
▸ `<div>`	class='table...		/dom
▾ `<div>`	class='listCa...		/dom
▾ `<div>`	class='listHe...		/dom
▸ `<p>`	class='listText'		/dom
▾ `<div>`	class='listPanel'		/dom
▾ `<div>`	class='listBody'		/dom
▸ `<p>`	class='locati...		/dom
▾ `<p>`	class='location'		/dom
#text	Amsterdam		/dom

full xpath: /dom[1]/html[1]/body[1]/div[4]/div[2]/div[1]/p[2]
smart xpath: /dom[1]/html[1]/body[1]/div[4]/div[2]/div[1]/p[2]
tag tokens: departure
atts: class='location'

Fig. 3. Screenshot of a tagging session in Metamorph

Automatic tagging. Metamorph follows a knowledge-based approach for automatic tagging. It derives search terms from its ontology and uses gazetteer lists for named entitiy recognition (NER). More complex expressions can be identified by using domain grammars, which are also stored in the central knowledge base. We embedded the GATE natural language processing (NLP) system [5] into Metamorph and make use of its JAPE transducer and gazetteer facilities. Details of the implementation cannot be provided here due to space constraints. Examples for concepts that our GATE application will recognize in the flight search domain are *airport names, iata codes, month names, numbers, person related concepts,* etc. We also allow domain engineers to include a number of domain heuristics in the knowledge base. These are some examples:

– Form element content and tag based heuristics make assumptions on the distribution of content, e.g. that several form elements share some content.
– Form meta data based heuristics recognize that some form elements share the same Javascript validator, or call a function with the name *datepicker()*.
– Cardinality based heuristics exploit the fact that a year has 12 months, and a month 28 to 31 days: If we encounter a select box with 28 to 31 numbers (the actual count may vary in a dynamically created form), it could be a day; a select box with 12 options could be a month selector.

5 Conclusion

Our approach gives the ontology and service engineers a number of possibilities to efficiently create and maintain vertical search scenarios. Since we do not envision a fully automated approach in the spirit of deep Web query probing, it is not meaningful and fair to apply traditional information extraction performance measures. Skilled application designers will fine-tune their applications to reach a robustness and reliability that is far above that of any automatic system. Therefore, we instead point out the main ingredients and "soft measures" why believe our framework to be advantageous to create vertical search scenarios:

– Our main aim is high result precision. The end user only trusts such a service if he trusts each individual result, and believes in the correctness and completeness of the market picture within the scope of the application. Therefore, we provide a visual and interactive system to create and maintain such services.
– The system is easy to learn and it is easy and time-efficient to add new web sources and to maintain existing form matchings, web interactions and data extraction programs.
– The system provides a good robustness to changes. Due to heuristics in the action ontology and a number of alerting conditions and exception handling that can be defined in the web interaction, minor changes pose no problem, and major changes are immediately reported to the service engineer. Further, any changes on the websites regarding data extraction aspects are covered by the robust extraction language used in the Lixto Visual Developer [1].

In the future, we plan to improve our system at the execution side by working on query performance and optimization, to switch to RDFa style annotations to become more tolerant for incorrect HTML code, and to enhance the automatic form detection and tagging component with topological and layout features. Finally, during the course of this project, we realized that many of the employed techniques (in particular the ontology-driven, automatic tagging of relevant concepts) could also be used to provide better accessibility of web pages to blind users. We plan to develop our ideas further into this direction.

References

1. Baumgartner, R., Flesca, S., Gottlob, G.: Visual Web Information Extraction with Lixto. In: Proc. VLDB (2001)
2. Baumgartner, R.: Methoden und Werkzeuge zur Webdatenextraktion. In: Semantic Web: Auf dem Weg zur vernetzten Wissensgesellschaft. Springer, Heidelberg (2006) (in German)
3. Chang, C., Kayed, M., Girgis, M.R., Shaalan, K.F.: A Survey of Web Information Extraction Systems. IEEE Trans. on Knowledge and Data Eng. 18/10 (2006)
4. Crescenzi, V., Mecca, G., Merialdo, P.: Roadrunner: Towards automatic data extraction from large web sites. In: Proc. VLDB (2001)
5. Cunningham, H., Maynard, D., Bontcheva, K., Tablan, V.: GATE: A Framework and Graphical Development Environment for Robust NLP Tools and Applications. In: Proc. ACL (2002)
6. Dapper. Products page (2006), `http://www.dapper.net`
7. Embley, D.W., Campbell, D.M., Smith, R., Liddle, S.W.: Ontology-based Extraction and Structuring of Information from Data-rich Unstructured Documents. In: Proc. CIKM (1998)
8. He, H., Meng, W., Yu, C., Wu, Z.: WISE-Integrator: An Automatic Integrator of Web Search Interfaces for E-Commerce (2003)
9. He, B., Chang, K.: Automatic complex schema matching across Web query interfaces: A correlation mining approach. ACM Trans. Database Syst. 31/1 (2006)
10. IBM Mashup Center, `http://www-306.ibm.com/software/info/mashup-center`
11. Kuhlins, S., Tredwell, R.: Toolkits for generating wrappers. Net.Object Days (2002)
12. Laender, A.H.F., Ribeiro-Neto, B.A., Silva, A.S., Teixara, J.S.: A brief survey of web data extraction tools. SIGMOD Rec. 31/2 (2002)
13. Ennals, R., Garofalakis, M.: MashMaker: Mashups for the Masses. In: SIGMOD (2007)
14. Meng, W., Peng, Q.: Clustering e-commerce search engines (2004)
15. Shestakov, D., Bhowmick, S., Lim, E.P.: DEQUE: Querying the Deep Web. Data and Knowledge Engineering 52 (2005)
16. Wang, J., Lochovsky, F.H.: Data Extraction and Label Assignment for Web Databases. In: Proc. WWW (2003)
17. Yahoo Pipes, `http://pipes.yahoo.com/pipes`