

Modeling Information Extraction Wrappers with Conceptual Graphs

F. Kokkoras, N. Bassiliades and I. Vlahavas

Department of Informatics, Aristotle University,
Thessaloniki, 54124, GREECE
{kokkoras,nbassili,vlahavas}@csd.auth.gr

Abstract. In this paper, we propose the use of the Conceptual Graphs knowledge representation and reasoning formalism to model information extraction wrappers (CG-Wrappers). An information extraction wrapper is a mapping that populates a data repository with implicit objects that exist inside a given web page. Creating a wrapper, usually involves some training by which the wrapper learns to identify the desired information based, mainly, on the surrounding HTML elements. In the paper, we demonstrate how the generalization, specialization and projection operations of the Conceptual Graph theory, naturally support semi-automatic wrapper induction and wrapper evaluation. The proposed modeling approach is flexible enough to support wrapper reuse, enabling us in that way to create more complex wrappers.

1 Introduction

Extracting nuggets of information from Web pages is a tedious task. A web search engine can probably help to locate an information resource, but it is still unable to process that resource and extract certain parts of it. Although RDF and XML are becoming widely recognized as the suggested solution for describing content in the Semantic Web [4] initiative, it will take quite some time until these technologies become common. Enormous amount of semantic data is still being encoded in HTML form, which is primarily for human consumption.

In the recent years, various researchers have proposed methods and developed tools towards the web information extraction task [17]. The term *wrapper* (or extraction rule) is used to describe a mapping that populates a data repository with implicit objects that exist inside a given web page. Creating a wrapper, usually involves some training (wrapper induction - [19]) by which the wrapper learns to identify the desired information. Unlike Natural Language Processing (NLP) techniques that rely on specific domain knowledge and make use of semantic and syntactic constraints, wrapper induction mainly focuses on delimiters. These features are usually the HTML tags that tell a web browser how to render the page.

There are two major research directions in wrapper induction. The first and older one, treats the HTML page as a linear sequence of HTML tags and textual content ([2], [15], [22]). Under this perspective, a wrapper generation is a kind of substring detection problem. Such a wrapper, usually includes delimiters in the form of substrings that prefix and suffix the desired information (Fig. 1). These delimiters can be either spotted to the wrapper generation program by the user (supervised learning) or located automatically (unsupervised learning). The former method usually requires less training examples but should be guided by a user with a good understanding of HTML. The latter approach usually requires more training examples but can be fully automated. Generally, both approaches succeed to their role to a degree (not 100%).

As the Internet technologies emerge, a new breed of wrapper induction techniques appeared ([5], [7], [19]) that utilize the Document Object Model (DOM) [9], a tree representation of an HTML document. Basically, such a tree wrapper uses path expressions to refer to page elements that contain the desired information [23] (Fig. 1).

<p>A linear wrapper extracting a digital camera model name from an HTML snippet. Extraction Rule: <code>skipto(, extractUntil(X,)</code> Source: <code>...<P>New model: Nikon Coolpix 3200</P>...</code></p>
<p>A tree wrapper (as a path expression) combined with the regular expression "<code>€\d</code>", that extracts prices in euros from HTML table cell tags. Extraction Rule: <code>*.table.*td(X, "€\d")</code> Source: <code>....<TABLE><TR><TD>Canon S300</TD><TD>€ 450.00</TD>.....</code></p>

Fig. 1: Typical expressions of linear and tree wrappers and their extracted result (framed text)

Thanks to the advanced tools that are available for web page design, HTML pages are nowadays highly well-formed, but at the same time the content is more decorated by using more HTML tags. As a result, although approximate location of desired information is relatively easy thanks to tree wrappers, extraction of the exact piece of information requires regular expressions or even NLP (Fig. 1). As a result, such hybrid approaches are becoming quite popular. Additionally, as more non-expert users are adapting information extraction technologies for personalized information and information filtering, visual tools that allow the easy creation and maintenance of wrappers ([1], [3], [10], [16], [20]) and declarative languages ([3], [18], [20]) for wrapper encoding have become the current established trend.

On the other hand, Conceptual Graphs (CGs) ([21], [8]) are a proven technology for knowledge representation and have been used in the past for knowledge based management of XML expressed educational metadata [13] and content based video retrieval [14].

In this paper, we propose the usage of the Conceptual Graphs knowledge representation and reasoning formalism to model information extraction wrappers, namely CG-Wrappers. We demonstrate how CGs naturally support semi-automatic wrapper induction as a series of automatic generalizations and user-defined specializations (which are operations of the Conceptual Graph (CG) theory) between training examples corresponding to DOM elements and expressed as CGs. Furthermore, we show how wrapper evaluation corresponds to the projection operation of the CG theory and demonstrate the expressiveness and flexibility of our approach, by combining simple CG-Wrappers into more complex ones.

The rest of the paper is organized as following: Section 2 introduces our novel approach on wrapper modeling and training that is based on the CG formalism. Section 3 describes our proposed execution model for CG-based wrappers, while Section 4 demonstrates the flexibility of our approach by providing examples of wrapper reuse. Section 5 presents related work, and finally, Section 6 concludes the paper and gives insight for future work.

2 CG-Wrappers: Modeling and Training Wrappers with CGs

The ability of CGs to represent entities of arbitrary complexity in a comprehensible way, make them a promising candidate for modeling information extraction wrappers. This perception is strengthened by the highly structured document representation which is defined by the DOM specification. This tree structure allows the easy mapping of web document elements to CG components.

<pre>[Wrapper] ← (targetURL) ← [URL] ← (output) ← [Info] ← (container) ← [HTMLElement]</pre>	<pre>[HTMLElement] ← (parent) ← [HTMLElement] ← (innerText) ← [Text] ← (tag) ← [HTMLTag] ← (siblingCount) ← [Integer] ← (siblingOrder) ← [Integer] ← (attribute) ← [Attribute]</pre>
--	---

Fig. 2: An abstract wrapper (left) and an HTML element (right) in conceptual graph notation

In general, a wrapper accesses a page located at a specific URL, searches inside this page for some specific HTML element which is the container of the desired information and extracts that information from it. This abstract description is encoded as the CG depicted in Fig. 2 (left)¹. Additionally, we can exploit the highly structured and information rich HTML element description provided by modern browsers² to detail the [HTMLElement] concept of this description. Such information includes, among others, the text contained inside the element, its attributes, the parent element under the DOM perspective, its tag name, etc. Besides this information, which is directly accessed, we also exploit calculated information that is derived if someone considers the neighborhood of some element. Such information includes, for example, the sibling order of this element being a child of its parent element and the total number of siblings. With this information in hand, a complex HTML element description can be created in CG form. Such a CG is presented in Fig. 2 (right), where for clarity, a simplification (CG operation) of six simple CGs over the common [HTMLElement] concept is presented. Arbitrary level of detail can be added to this description resulting in much more detailed and complex CGs.

In practice, the wrapper depicted in Fig. 2 (left) is too generic, in the sense that it describes every single element of an HTML page. More specialization is required, particularly in the [HTMLElement] concept.

¹ For space economy, we use textual representation of CGs.

² We have used Internet Explorer's DOM representation.

Under the machine learning perspective, training information extraction wrappers is a combination of generalization and specialization operations that result in a model (pattern) that best describes the training instances and that can be used to detect new, unknown instances. This is similar to the generalization and specialization operations of the CG theory. *Generalization* is an operation that monotonically increases the set of models for which some CG is true [21]. For example, CG₃ in Fig. 3 is the minimum common generalization of CG₁ and CG₂. Only common parts (concepts and relations) of CG₁ and CG₂ are kept in CG₃. In addition, individual concepts like [BGColor: "#FFFFFF"] have become generic by removing the referent field. *Specialization* is the opposite to the generalization operation. It monotonically decreases the set of models for which some CG is true. This is achieved by either adding more parts (concepts and/or relations) to a CG, or by assigning an individual referent to some generic concept [21].

CG ₁ : [HTMLElement: #3] ← (attribute) ← [BGColor: "#FFFFFF"]
CG ₂ : [HTMLElement: #9] ← (attribute) ← [BGColor: "#CCFF12"] ← (parent) ← [HTMLElement: #8]
CG ₃ : [HTMLElement] ← (attribute) ← [BGColor]

Fig. 3: CG₃ is the minimum common generalization of CG₁ and CG₂

Thus, a CG-Wrapper is the result of generalization and specialization operations over two or more training instances (HTML elements) expressed as CGs.

We demonstrate how our generic wrapper can be specialized using the classical problem of extracting information from an electronic flea market. Fig. 4 presents a snippet from a web page of such a site. Information is organized in an HTML table, where the first row contains the headers and the rest of the rows correspond to records describing offered products. For simplicity, let's assume that we want to extract the names of the products offered.

name	user	closes in ▲	offers	price
 MODEM MOTOROLA SM56 E...	WEASEL(140)	2 hours, 50 minutes	1	€ 15.00 (5112 dra.)
 DIAMOND SUPRA v92 inte...	ABBOT(240)	1 day, 10 hours	-	€ 22.00 (7497 dra.)
 USB MICRONET MODEL 300...	hawk21(178)	2 days	-	€ 42.00 (14312 dra.)
 ΣΕΙΡΙΑΚΟ ΜΟΔΕΜ 56K MIC...	zsp(112)	2 days	-	€ 32.00 (10904 dra.)

Fig. 4: A snippet from an on-line flea market

[Wrapper: fleaName] ← (targetURL) ← [URL: "www.fleamarket.gr"]
← (output) ← [Info]
← (container) ← [HTMLElement: #16]
← (parent) ← [HTMLElement:#15]
← (innerText) ← [Text: "MODEM MOTOROLA SM56 E..."]
← (tag) ← [HTMLTag: "TD"]
← (siblingCount) ← [Integer: 5]
← (siblingOrder) ← [Integer: 1]
← (attribute) ← [BGColor: "#FFFFFF"]

Fig. 5: First training instance of a CG wrapper

In a real situation, the wrapper creation program should allow the identification of instances of the desired information, by simply pointing them with the mouse. In addition, visualizing the evaluation of a wrapper can greatly simplify the whole wrapper training task. We have developed and currently refine such a tool by exploiting the Internet Explorer's application programming interface (API).

Let's say the user points, through some visual interface, to the table cell containing the name of the first product. This specializes the generic wrapper description, which takes the form presented in Fig. 5. Unfortunately, this specialized version is not general enough since it is able to extract only the training instance. A second training instance should be used, say the cell containing the name of the second product. This results in the wrapper instance presented in Fig. 6.

```
[Wrapper: fleaName] ← (targetURL) ← [URL: www.fleamarket.gr]
                    ← (output) ← [Info]
                    ← (container) ← [HTMLElement: #26]
                                ← (parent) ← [HTMLElement: #25]
                                ← (innerText) ← [Text: "DIAMOND SUPRA v92 inte..."]
                                ← (tag) ← [HTMLTag: "TD"]
                                ← (siblingCount) ← [Integer: 5]
                                ← (siblingOrder) ← [Integer: 1]
                                ← (attribute) ← [BGCOLOR: "#CCCCCC"]
```

Fig. 6: Second training instance of a CG wrapper

```
[Wrapper: fleaName] ← (targetURL) ← [URL: www.fleamarket.gr]
                    ← (output) ← [Info: X]
                    ← (container) ← [HTMLElement] ← (parent) ← [HTMLElement]
                                                    ← (innerText) ← [Text: ?X]
                                                    ← (tag) ← [HTMLTag: "TD"]
                                                    ← (siblingCount) ← [Integer: 5]
                                                    ← (siblingOrder) ← [Integer: 1]
                                                    ← (attribute) ← [BGCOLOR]
```

Fig. 7: Generalization result of CG wrapper instances of Fig. 5 and Fig. 6

Using the generalization operation of the CG theory for the two CG-Wrapper instances, a generic wrapper describing (extracting) both product names can be created (Fig. 7). Although this wrapper is generic enough to extract all product names of the table in Fig. 4, it also extracts the first header cell. This time, specialization is required to exclude the header cell. This can be established over the HTML element that is the parent of the element containing the extracted information. This element refers to a row of the product table. Excluding this row is as simple as requesting that this element's sibling order is greater than one.

The final CG-Wrapper is presented in Fig. 8. Note that the concept which contains the desired information ([Info]) is fed by the [Text: ?X] concept, since this part of the web page contains the desired information. In addition, parts of the final wrapper description that do not affect the accuracy of the wrapper, such as the [BGCOLOR] are dropped out.

```

[Wrapper: fleaName] ← (targetURL) ← [URL: www.fleamarket.gr]
← (output) ← [Info: X]
← (container) ← [HTMLElement]
    ← (parent) ← [HTMLElement] ← (siblingOrder) ← [Integer:>1]
    ← (innerText) ← [Text: ?X]
    ← (tag) ← [HTMLTag: "TD"]
    ← (siblingCount) ← [Integer: 5]
    ← (siblingOrder) ← [Integer: 1]

```

Fig. 8: The final CG wrapper modeling the product names of the table in Fig. 4

Finally, regular expressions can be used over the initially extracted information to fine-tune the output. For example, extracting the price in euros from the flea market example, requires the replacement of ?X with some proper regular expression that is applied over X.

The way a CG-Wrapper is evaluated over a target web page, is described next.

3 Applying CG-Wrappers

Applying a CG-Wrapper to some web page is equivalent to a projection operation of the wrapper over the web page elements, expressed as CGs. *Projection* is a complex operation that projects a CG v over another CG u which is a specialization of v ($u \leq v$), that is, there is a sub graph u' embedded in u that represents the original v . The result is one or more CGs πv which are similar to v but some of its concepts is possible to have been specialized by either specializing the concept type or assigning a value to some generic referent, or both [21].

```

[HTMLElement: #25] ← (siblingOrder) ← [Integer: 3]

[HTMLElement: #26] ← (parent) ← [HTMLElement: #25]
← (innerText) ← [Text: "DIAMOND SUPRA v92 inte..."]
← (tag) ← [HTMLTag: "TD"]
← (siblingCount) ← [Integer: 5]
← (siblingOrder) ← [Integer: 1]
← (attribute) ← [BGCOLOR: "#CCCCCC"]

```

Fig. 9: Two nodes of an HTML tree, in CG form (partially presented)

```

[Wrapper: fleaName] ← (targetURL) ← [URL: www.fleamarket.gr]
← (output) ← [Info: "DIAMOND SUPRA v92 inte..."]
← (container) ← [HTMLElement]
    ← (parent) ← [HTMLElement] ← (siblingOrder) ← [Integer:>1]
    ← (innerText) ← [Text: "DIAMOND SUPRA v92 inte..."]
    ← (tag) ← [HTMLTag: "TD"]
    ← (siblingCount) ← [Integer: 5]
    ← (siblingOrder) ← [Integer: 1]

```

Fig. 10: The wrapper of Fig. 8 after applying it over the CGs of Fig. 9

For example, consider the two CGs of Fig. 9 which refer to the table of Fig. 4, representing the second product row and the first cell of this row, respectively. Projecting the container part of the CG wrapper of Fig. 8 over the second CG of Fig. 9 results in an instantiated CG-Wrapper where the unbound X referent of the [Text: ?X] concept have been unified with "DIAMOND SUPRA v92 inte...". Note that, the exact projection involves also a replacement of the concept [HTMLElement: #25] of the second CG, with the CG definition of this concept (that is, the first CG in Fig. 9). This inner task corresponds to the expansion operation of the CG theory, where a concept is replaced by its CG definition. The final instantiated wrapper is displayed in Fig. 10.

The way HTML elements are considered can be either *naive* (iterating over all the nodes of the HTML tree, trying to satisfy the constraints imposed by the wrapper components) or *optimized* (iterating over a certain type of element, like table cells). The semantics of both execution models are derived from the CG theory: The evaluation of a CG-Wrapper is the result πv of a projection operation that projects the container part u of the wrapper over a CG expressed HTML node v of some target URL.

4 Reusing CG-Wrappers

The CG-Wrapper model, is expressive enough to handle nested wrapper definitions, that is, wrappers that are defined in terms of other wrappers. Such a very useful case, is the definition of a *looping wrapper* that collects results from chained pages containing search results. Consider for example the typical case in which an on-line store presents the results of some user query in individual pages containing 10 items each. In such cases, at the bottom of all pages but the last one, there is a link to the next result page, usually named "Next Page". A looping wrapper is capable of extracting information from all results pages by automatically following the "Next Page" link.

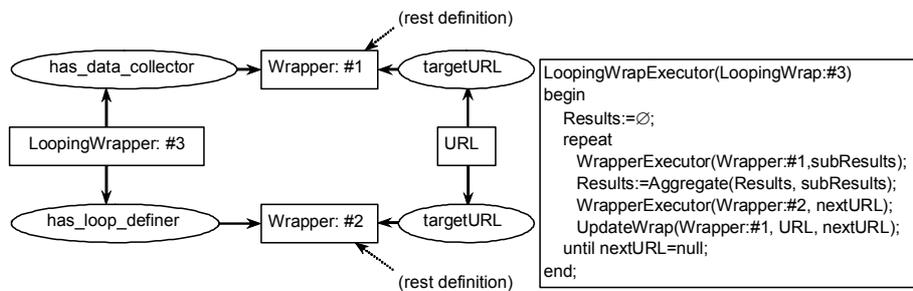


Fig. 11: A looping CG-Wrapper (left) and its abstract execution model (right)

Thus, a *looping CG-Wrapper* (Fig. 11) is a combination of a data collector wrapper (Wrapper:#1) and a loop definer wrapper (Wrapper:#2). A *data collector* is a typical CG-Wrapper that extracts information from a web page. A *loop definer* is a CG-Wrapper that extracts the URL of the next page, in the case of information that is presented in a sequence of pages. These two wrappers have a common target URL.

The evaluation of a looping wrapper is also presented in Fig. 11 (right). First, the data collector is executed and the extracted information is appended to the already extracted results. Then, the execution of the loop definer wrapper follows which extracts, from the same page, the URL of the "Next Page" link. If this second wrapper brings results then the target URL of the data collector is updated. These steps are repeated until the loop definer fails to extract information.

Since an information extraction wrapper is specialised to extract certain pieces of information, another useful case of wrapper reuse is the *dual wrapper* with which we aim at extracting feature-value pairs. Fig. 12 displays a dual wrapper ([DualWrapper: CanonDigitalZoom]) extracting feature-value information (digital zoom of a digital camera model). It is defined in terms of a feature locator wrapper ([Wrapper: #1]) that locates the table cell ([HTMLTag: "TD"]) containing the text "Digital Zoom", and a value extractor wrapper ([Wrapper: #2]) that extract the value of the feature. The second wrapper is modelled to search in the table cell that is right after the cell the first wrapped worked with. This correlation is established over the parameter ?X.

```
[Wrapper: #1] ← (targetURL) ← [URL: www.powershot.com/powershot2/s400/specs.html]
              ← (output) ← [Info: X]
              ← (container) ← [HTMLElement] ← (innerText) ← [Text: "Digital Zoom"]
                                                    ← (tag) ← [HTMLTag: "TD"]
                                                    ← (siblingOrder) ← [Integer: ?X]

[Wrapper: #2] ← (targetURL) ← [URL: www.powershot.com/powershot2/s400/specs.html]
              ← (output) ← [Info: V]
              ← (container) ← [HTMLElement] ← (innerText) ← [Text: ?V]
                                                    ← (tag) ← [HTMLTag: "TD"]
                                                    ← (siblingOrder) ← [Integer: X+1]

[DualWrapper: CanonDigitalZoom] ← (featureWrapper) ← [Wrapper: #1]
                                  ← (valueWrapper) ← [Wrapper: #2] ← (output) ← [Info: ?V]
                                  ← (output) ← [Info: V]
```

Fig. 12: A dual wrapper extracting digital zoom information.

An advantage of this approach is that the second wrapper is becoming capable of performing a "blind" extraction, in case the value of some feature is presented in an unknown way. In a "blind" extraction the wrapper extracts all the text inside some HTML tag, because "it knows" that the information is there. This is obviously better than an exact-or-nothing approach. In addition, we are not depended on absolute positioning to refer to HTML nodes but we follow a "relative to textual information" methodology instead, which is more robust to small page changes. This is very crucial, since many commercial sites tend to make frequent alterations to their sites to prevent wrapping.

5 Related Work

Our approach differs from similar work mainly in the core technology used (CGs). Our choice allow us to exploit both DOM representations of web documents (as in [3], [18] and [20]), as well as user defined structures that describe objects of interest

(as in [1] and [16]). We achieve this by using CG-based generic wrapper descriptions which are detailed by the user in an interactive way utilizing both the DOM representation and the browser itself. CGs naturally support all the major steps in information extraction with wrappers, with its generalization, specialization and projection operations. In addition, CGs are a proper candidate for describing ontological information. They offer a unified and simple representation formalism that covers a wide range of other data and knowledge modelling formalisms and allow matching, transformation, unification and inference operators to process the knowledge that they describe [12]. This can provide a common schema for information integration and improve wrapper's resiliency and adaptivity, in the way [10] does. Beyond that, CGs provide all the operations required to create a functional reasoning system and does not rely on additional technologies like, for example, a possible RDF approach which requires OWL for reasoning. Finally, the CG formalism has, by nature, better visualization potential. This enables our system to provide a more comprehensible wrapper representation to the end-user.

6 Conclusions and Future Work

Wrapper induction technology demonstrates that shallow pattern matching techniques, which are based on document structural information rather than linguistic knowledge, can be very effective. The approach we proposed in this paper, heavily capitalizes on the CG technology. We have shown that CGs naturally support all the major steps in information extraction with wrappers, with their generalization, specialization and projection operations. Furthermore, the expressiveness of CGs allows us to easily model complex wrappers by wrapper reuse.

We are currently implementing a prototype that is based on the proposed model. We aim at providing visual tools for wrapper training by exploiting Internet Explorer's API. For the CG handling we use the CoGITaNT library [6], that allows the handling of CGs using an object oriented approach. In the future, we plan to include ontological knowledge to improve the accuracy and the flexibility of CG-Wrappers. Having already used CGs for modelling and training, CG based ontological knowledge can be easily incorporated and contribute towards knowledge-based wrappers.

References

- [1] Adelberg B.: NoDoSE: A Tool for Semi-Automatically Extracting Structured and Semi-Structured Data from Text Documents. *SIGMOD Record*, 27(2), (1998), 283-294
- [2] Ashish N. and Knoblock C.: Wrapper Generation for Semi-structured Internet Sources. In *Proc. of Workshop on Management of Semi-structured Data*, (1997)
- [3] Baumgartner R., Flesca S. and Gottlob G.: Visual Web Information Extraction with Lixto. In *Proceedings of the 27th International Conference on Very Large Data Bases*, (2001), 119-128
- [4] Berners-L.T., Hendler J., Lassila O.: *Semantic Web*. Scientific American, (2001)

- [5] Buttler D., Liu L. and Pu C.: A Fully Automated Object Extraction System for the World Wide Web. In Proceedings of the 21th International Conference on Distributed Computing Systems, (2001), 361–370
- [6] CoGITaNT library, available under GPL at: <http://cogitant.sourceforge.net>
- [7] Cohen W. W. and Jensen L. S.: A Structured Wrapper Induction System for Extracting Information from Semi-structured Documents. In Proceedings of IJCAI 2001 Workshop on Adaptive Text Extraction and Mining, (2001)
- [8] Conceptual Graph Standard Working Draft. www.jfsowa.com/cg/cgstand.htm
- [9] Document Object Model (DOM): <http://www.w3.org/DOM/>
- [10] Embley D. W., Campbell D. M., Jiang Y. S., Liddle S. W., Ng Y.-K., Quass D. and Smith R. D.: A Conceptual-Modelling Approach to Extracting Data from the Web. In Proc. of Int. Conference on Conceptual Modelling. (1998), 78-91
- [11] Freitag D. and Kushmerick N.: Boosted Wrapper Induction. In Proceedings of the 17th National Conference on Artificial Intelligence, (2000), 577-583
- [12] Gerbe O. and Mineau G. W.: The CG Formalism as an Ontolingua for Web-Oriented Representation Languages. In Proceedings of the ICCS 2002, Springer Verlag, LNAI 2392, (2002), 205-219
- [13] Kokkoras F., Sampson D. and Vlahavas I.: A Knowledge Based Approach on Educational Metadata Use. Post-proc. 8th Panhellenic Conf. in Informatics, Y. Manolopoulos, S. Evripidou and A. Kakas (Eds.), Springer, LNCS 2563, (2003)
- [14] Kokkoras F., Jiang H., Vlahavas I., Elmagarmid A. K., Houstis E. N. and Aref W. G.: Smart VideoText: A Video Data Model based on Conceptual Graphs. ACM-Multimedia Systems Journal, Springer, Vol.8, (2002), 328-338
- [15] Kushmerick N., Weld D. S. and Doorenbos R. B.: Wrapper Induction for Information Extraction. In Proceedings of the 15th International Joint Conference on Artificial Intelligence, (1997), 729–737
- [16] Laender A.H.F., Ribeiro-Neto B.A. and da Silva A.S.: DEByE - Data Extraction by Example. Data and Knowledge Engineering, 40(2), (2001), 121-154
- [17] Laender A., Ribeiro-Neto B., da Silva A. and Teixeira J.: A Brief Survey of Web Data Extraction Tools. SIGMOD Record, 31(2), (2002)
- [18] Liu L., Pu C. and Han W.: XWRAP: An XML-Enabled Wrapper Construction System for Web Information Sources. In Proceedings of the 16th IEEE International Conference on Data Engineering, (2000), 611–621
- [19] Muslea I., Minton S. and Knoblock C.: Wrapper induction for semi structured information sources. Journal of Autonomous Agents and Multi-Agent Systems, 16(12), (1999)
- [20] Sahuguet A. and Azavant F.: Building intelligent web applications using light-weight wrappers. Data and Knowledge Engineering, 36(3), (2001), 283-316
- [21] Sowa J.: Conceptual Structures: Information Processing in Mind and Machine. Addison-Wesley Publishing Company, (1984)
- [22] Yamada Y., Ikeda D. and Hirokawa S.: Automatic Wrapper Generation for Multilingual Web Resources. In Proceedings of the 5th International Conference on Discovery Science, Springer-Verlag, LNCS 2534, (2002), 332–339
- [23] Yamada Y., Ikeda D. and Hirokawa S.: Expressive Power of Tree and String Based Wrappers. In on-line proceedings of IJCAI'03 workshop on Information Integration on the Web (IIWeb-03), <http://www.isi.edu/info-agents/workshops/ijcai03/papers/Herzog-ijcai03-herzog.pdf>, (2003)