Towards a Universal Web Wrapper

Theodore W. Hong and **Keith L. Clark**
Department of Computing
Imperial College London
180 Queen's Gate, London SW7 2BZ, United Kingdom
Tel +44 20 7594-1117 / Fax +44 20 7581-8024
{t.hong,k.l.clark}@doc.ic.ac.uk

Abstract

The wealth of information contained in the world-wide web has created much interest in systems for integrating information from multiple sites. We describe a universal wrapper machine that can learn to extract information from the web given only a set of general rules describing the data domain. It cleanly separates out site-independent and site-specific knowledge from execution implementation. Site-independent knowledge is expressed in user-supplied domain rules, while site-specific knowledge is expressed in automatically-generated context-free grammars that describe site structures. The two are combined by using the domain rules to semantically interpret the parse trees generated by the grammars. The resulting declarative wrapper specifications are easily understandable by humans and can be executed to perform information extraction. Once extracted, tuples can be queried by external agents using a high-level agent communication language.

Introduction

The immense success of the world-wide web has made available a wealth of online information resources of all kinds. Unfortunately, its disorganised and complex nature has made it difficult to exploit the full potential of the data available. This is particularly true for domains in which similar types of data are fragmented across many different sites that are inconsistent and hard to find; for example, real estate listings, flight schedules, weather forecasts, job postings, store catalogs, and many more. In order to run searches or comparisons across the full data space, users are forced to spend a great deal of effort to track down sources, slog through limited-functionality search interfaces, and switch back and forth among multiple sites. It would be much better to have a single metasource that could automatically extract data from multiple sources, mark it up and integrate it semantically, and provide a high-level query interface for user applications.

In the most general case, the extraction task would require full natural language understanding. Fortunately, most web pages provide hints in the form of visual layout structures that show up as syntactic regularities in the corresponding HTML code. Still, since layouts vary widely between sites and mix structural markup with purely presentational elements, extracting data from web pages remains challenging.

The most straightforward approach is to hand-write extraction programs, or *wrappers*, for data sources, using a general-purpose language such as C or Perl, or a dedicated wrapper language such as WebL (Kistler & Marais 1998). Unfortunately, manually coding wrappers is costly and time-consuming because of the large number of sites to be covered and the constant need to keep up-to-date with frequent formatting changes. Another problem is that knowledge about site structures is implicitly embedded inside the wrapper code and mixed with details of the implementation, rather than being specified explicitly.

More recent systems have turned to machine learning to automatically or semi-automatically create wrappers. However, these systems generally rely on either manual labelling of training examples, which still requires substantial user effort, or unprincipled *ad hoc* heuristics. Many also use low-level wrapper representations such as finite-state automata which are difficult to understand or modify, and mix site-independent domain information about attributes with site-specific information about layouts.

As part of the **Grazer** information extraction system (Hong 2003), we have developed a *universal wrapper machine* (UWM) that is able to learn wrappers for sites in a principled way that only requires a set of general domain rules instead of labelled examples. It cleanly separates out site-independent domain knowledge and site-specific structural descriptions from implementation code. Domain knowledge is encoded in declarative rules that are supplied by users to describe domain attributes. Structural descriptions are expressed as context-free grammars that can be automatically generated by an unsupervised learner examining pages on a site. Together, the domain rules and structure grammars form wrapper specifications that can be executed to perform information extraction from arbitrary sources.

The advantage of this approach is that it minimizes human intervention while at the same time avoiding reliance on *ad hoc* heuristics. Wrapper knowledge is made explicit and modular. Domain rules need only be written once and can be reused across all sites providing a particular type of information. Site structures can be easily understood by humans and modified if necessary.

The rest of this paper is organized as follows. We first describe our language for domain rules, then go on to discuss site structures and how they might be generated. Next we present the universal wrapper and describe how it combines domain rules and site structures to perform information extraction. Following that we present the high-level interface that other agents can use to query the extracted data. Finally, we survey related research and offer some conclusions and directions for future work.

Domain Rules

A **Grazer** domain rule consists of an attribute name and datatype, together with a description of the appearance of its corresponding field.[1] It has the general format:

datatype attrname = *regexp*

where *datatype* is one of `string`, `number`, or `boolean`; *attrname* is an alphanumeric identifier specifying the name of the attribute to be extract; and *regexp* is a regular expression specifying a text pattern for the corresponding field.

To execute a rule, *regexp* is compared against a target string. If the expression matches any substring of the target, then the rule succeeds. *regexp* may optionally be annotated by placing curly braces around one of its subexpressions. For `string` or `number` rules, the portion of the match corresponding to that subexpression is captured and returned as the value of the attribute; or if there are no braces, the entire match is returned. For `boolean` rules, `true` or `false` is returned depending on whether or not the rule succeeded.

Multiple rule sets can be loaded; later definitions will override earlier ones if conflicts arise. Typically, the first set loaded will be a generic default for the domain. Special handling for unusual sites can be added by loading supplementary rule sets.

For example, in the real estate domain we might specify a default set of rules such as the following:

```
number price    = £{[0-9,]+}
string postcode = {(N|E|SE|SW|W|NW)[0-9]+}
string tel      = [0-9]+[ -]?[0-9]+[ -]?
                  [0-9]+
boolean garden  = "garden"|"yard"
boolean parking
```

The first rule says that a price looks like a pound sign followed by a string of digits and commas (i.e., a number). That number is returned as the value of the attribute. The second contains a disjunction: a London postal code is a geographic prefix[2] followed by a number. The third rule declares a telephone number as three sets of digits, optionally separated by dashes or spaces, while the fourth defines a boolean attribute which is true if either of the words `garden` or `yard` is present, ignoring case. Finally, the fifth rule defaults to a simple match on the attribute name (in this case, `parking`) in the absence of a specified regexp.

Note that the rules need only be written once for any given domain. They should be fairly easy to write, as they describe general features of the fields themselves, not the peculiarities of site-specific delimiters. They could also be generated by a learning algorithm that induces regular expressions from examples. This would be easier than page labelling because it would only require some examples of field values; they would not need to be marked up in context.

Since the rules are quite general, we cannot simply apply them directly to perform extraction because there would be problems with avoiding spurious matches and distinguishing multiple matches belonging to different records. We need a structure description to provide the necessary context within which to apply the rules and organise the extracted attributes into tuples.

[1]Here, *attribute* refers to an abstract domain element, while *field* refers to a text string encoding one of its possible values.

[2]Some prefixes have been omitted for reasons of space.

Site Structures

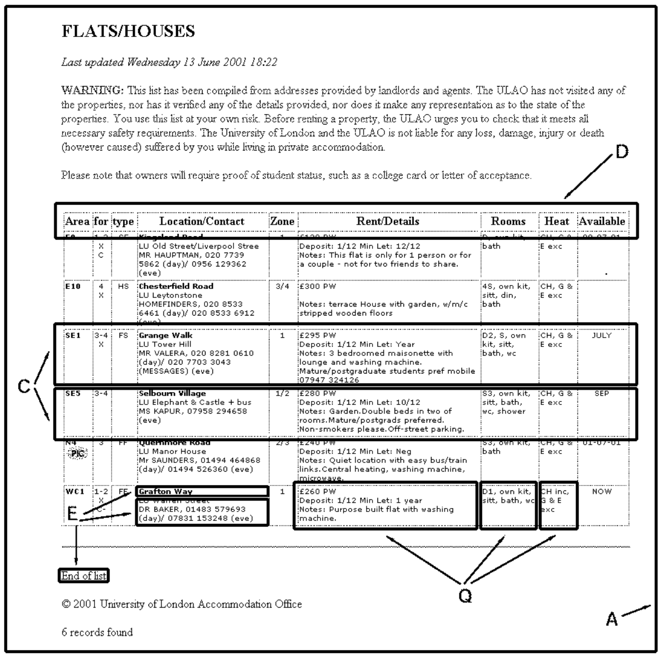
Grazer describes sites using context-free grammars, because they provide a powerful and convenient formalism for representing hierarchical structure. Grammars permit extremely general arrangements of fields and tags to be defined while remaining easy to understand and modify. Previously (Hong & Clark 2001), we described a grammatical inference algorithm based on hill-climbing search and showed how it could learn a structure grammar for a simplified mock page. In this paper we focus in more detail on how to use such structures for information extraction and apply our method to more complex real-world data.

For example, consider the University of London real estate listing page shown in Figure 1. The following structure grammar might be used to describe this page:

```
A  →  html K body_bgcolor(linen) E
      table_valign(top)_border D C
      /table E hr_noshade E /body
      /html
K  →  head title E /title /head
D  →  tr J /tr
J  →  th E /th J  | ϵ
-----------------------------------------
C  →  tr_valign(top) Q /tr C  | ϵ
Q  →  td E /td Q  | td_align(center)
      E /td Q  | ϵ
-----------------------------------------
E  →  text E  | br E  | p
      E  | img_border(0) E
      | font_size(1) E  | a E /a
      E  | b E /b E  | i E /i E
      font_color(red) E  | /font
      hr_noshade E  | script comment
      /script h2 text /h2 E  | ϵ
```

Here, the terminal symbols of the grammar represent generalized HTML tags. For example, `img_border(0)` stands for `<img border="0">` with any additional attributes left unspecified (in particular, `src`). The special symbol `text` represents free text between tags.

We can interpret this grammar as follows. The start symbol A represents a complete page, while the symbol E represents text marked up in various ways. A page consists of an `<html>` tag, a title K, a `<body>` tag, some text, a `<table>` containing one D-type row and several C-type rows, some more text, a horizontal line, yet more text, and closing `</body>` and `</html>` tags. A D-type row is a header row containing a set of column headings J. A C-type row is a data row containing a set of data cells Q. Note



FLATS/HOUSES

Last updated Wednesday 13 June 2001 18:22

WARNING: This list has been compiled from addresses provided by landlords and agents. The ULAO has not visited any of the properties, nor has it verified any of the details provided, nor does it make any representation as to the state of the properties. You use this list at your own risk. Before renting a property, the ULAO urges you to check that it meets all necessary safety requirements. The University of London and the ULAO is not liable for any loss, damage, injury or death (however caused) suffered by you while living in private accommodation.

Please note that owners will require proof of student status, such as a college card or letter of acceptance.

Area	for	type	Location/Contact	Zone	Rent/Details	Rooms	Heat	Available
E8	1-2 X C	SF	**Kingsland Road** LU Old Street/Liverpool Stree MR HAUPTMAN, 020 7739 5862 (day)/ 0956 129362 (eve)	1	[illegible] PW Deposit: 1/12 Min Let: 12/12 Notes: This flat is only for 1 person or for a couple - not for two friends to share.	D, own kit, bath	CH, G & E exc	[illegible]
E10	4 X	HS	**Chesterfield Road** LU Leytonstone HOMEFINDERS, 020 8533 6461 (day)/ 020 8533 6912 (eve)	3/4	£300 PW Notes: terrace House with garden, w/m/c stripped wooden floors	4S, own kit, sitt, din, bath	CH, G & E exc	
SE1	3-4 X	FS	**Grange Walk** LU Tower Hill MR VALERA, 020 8281 0610 (day)/ 020 7703 3043 (MESSAGES) (eve)	1	£295 PW Deposit: 1/12 Min Let: Year Notes: 3 bedroomed maisonette with lounge and washing machine. Mature/postgraduate students pref mobile 07947 324126	D2, S, own kit, sitt, bath, wc	CH, G & E exc	JULY
SE5	3-4		**Selbourn Village** LU Elephant & Castle + bus MS KAPUR, 07958 294658 (eve)	1/2	£280 PW Deposit: 1/12 Min Let: 10/12 Notes: Garden.Double beds in two of rooms.Mature/postgrads preferred. Non-smokers please.Off-street parking.	S3, own kit, sitt, bath, wc, shower	CH, G & E exc	SEP
N4 PIC	3	FF	**Quernmore Road** LU Manor House Mr SAUNDERS, 01494 464868 (day)/ 01494 526360 (eve)	2/3	£240 PW Deposit: 1/12 Min Let: Neg Notes: Quiet location with easy bus/train links.Central heating, washing machine, microwave.	S3, own kit, bath	CH, G & E exc	01-07-01
WC1	1-2 X C	FF	**Grafton Way** LU Warren Street DR BAKER, 01483 579693 (day)/ 07831 153248 (eve)	1	£260 PW Deposit: 1/12 Min Let: 1 year Notes: Purpose built flat with washing machine.	D1, own kit, sitt, bath, wc	CH inc, G & E exc	NOW

End of list

© 2001 University of London Accommodation Office

6 records found

Figure 1: A sample real estate listing page, divided into grammatical units.

that the C and Q productions correspond to the records and fields that we would like to extract.

This approach is not just restricted to representing pages with tables, however. Almost any type of page structure can be represented by an appropriate grammar, although learning the right grammar is not always easy.

Page Parsing

Given a structure grammar, the UWM can process an input page from a new site into a parse tree of text fragments. The result for the real estate page is shown in Figure 2. Leaves shown in boldface correspond to `text` productions. Notice that the column headings, such as "Area," have ended up in a different branch from the column data, such as "E8" (a specific area postcode). This separation would not have occurred if the page were parsed according to the usual HTML grammar, but reflects additional knowledge contained in the structure grammar.

The subtrees rooted at the various nonterminals A, D, J, C, and so on can be regarded as dividing the input page into various overlapping units, as shown by the boxes in Figure 1. In order to determine which of these subtrees might correspond to record units, the wrapper machine builds a table of their text expansions.

For each subtree, an entry is created that pairs its root nonterminal with a list of all the `text` productions appearing in it. If a subtree contains recursive subtrees, as with the J-, Q-, and C-subtrees, the recursive productions involving the same nonterminal as its root are not placed in its entry.

Some typical entries are shown in Table 1. In this example, the nonterminal symbol A has one entry containing all of the text on the page. D has one entry containing a list of

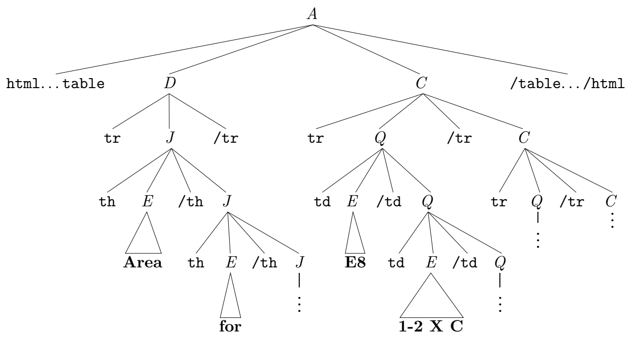


Figure 2: Part of the parse tree for Figure 1.

all of the column headers, while J has six entries, one for each header (not all are shown). Similarly, C has six entries each containing the text of an entire data row, while Q has 52 entries each containing one non-empty data cell. E has a large number of small entries, one for each individual piece of text on the page.

Attribute Extraction

We now need to extract attributes from these text strings and group them appropriately. This is done by using the domain rules to score the entries, as follows. Each rule in turn is applied to every string in the expansion list for a given entry. One point is given for every rule that matches once and only once in an expansion list, meaning that the corresponding attribute can be extracted successfully and unambiguously.

Symbol	Text
A	"FLATS/HOUSES", "Last updated 13 June", ...
D	"Area", "for", "type", "Location/Contact", ...
J	"Area"
J	"for"
J	"type"
C	"SE1", "Grange Walk", "LU Tower Hill", ...
C	"SE5", "Selbourn Village", "LU Elephant", ...
Q	"SE1"
Q	"Grange Walk", "LU Tower Hill", ...
Q	"£295", ...
E	"FLATS/HOUSES"
E	"Area"
E	"SE1"
E	"Grange Walk"
E	"LU Tower Hill"
E	"£295"
E	"End of list"
E	"6 records found"

Table 1: Some typical entries in the text expansion table.

For example, the first C entry contains exactly one string ("SE1") that matches the postcode rule, and thus gains a point for that rule. The A entry, on the other hand, contains the entire text of the document. It does not score a point for the postcode rule, since its entry matches the rule six times (there are six postcodes present on the page in total).

After scoring the entries, the UWM assigns an overall score to each nonterminal by averaging over all of its entries in the table. In this case, we obtain the following set of overall scores (out of a maximum of five, which would mean every attribute extracted perfectly every time):

Symbol	Average score
A	1.0
K	0.0
D	0.0
J	0.0
C	**2.7**
Q	0.33
E	0.18

A did not score well because most rules matched multiple times (apart from the parking rule, which coincidentally matched only once because only one property had parking), indicating that it probably contains more than one record. By contrast, Q and E had several unique matches, but those matches were sparsely scattered across many entries, lowering the average scores. This pattern indicates that they are probably smaller than a record. K, D, and J contained no matching attributes at all and scored nothing. The highest-scoring nonterminal was C, as its entries successfully matched the most rules without duplicates. Therefore, the UWM considers it the most likely candidate to be a record. A look at Figure 1 intuitively confirms these results.

Wrapper Execution

Having marked the C nonterminal as corresponding to a record, we are ready to perform information extraction. For subsequent pages from this site, the UWM can skip directly to this point without redoing the scoring. Each subtree rooted at an occurrence of the marked nonterminal is scanned using the domain rules and the attributes found in that subtree are extracted as a tuple. (If a rule matches more than once, the first value is used.)

In the running example, this procedure yields the following tuples:

Postcode	Price	Telephone	Garden	Parking
E8	120	020 7739 5862	false	false
E10	300	020 8533 6461	true	false
SE1	295	020 8281 0610	false	false
SE5	280	07958 294658	true	true
N4	240	01494 464868	false	false
WC1	260	01483 579693	false	false

which are collected by the UWM.

Query Interface

The UWM provides a standard agent communication interface that permits external agents to query collected tuples using the agent communication language FIPA ACL (FIPA 2001). Incoming `query` messages specifying high-level queries are responded to with sets of `inform` messages carrying tuples as facts. Agents can also `subscribe` with the UWM to be informed of new tuples on an ongoing basis as they are extracted.

A typical FIPA ACL query might be:

```
query-ref:
  sender:
    agent-identifier:
      name: User1
      addresses:
        sequence:
          url: 10.0.0.2
  receiver:
    set:
      agent-identifier:
        name: UWM
        addresses:
          sequence:
            url: 10.0.0.1
  content: (kappa (?price ?beds ?pcode)
                (and (rentals
                       ?price ?beds ?pcode
                       ?garden ?parking)
                     (< ?price 400)
                     (or (= ?pcode 'sw6)
                         (= ?pcode 'sw7))))
  language: FIPA-KIF
  ontology: real-estate-ontology
```

This message specifies the `query-ref` performative, meaning that the sender (User1) wishes to ask the receiver (the UWM) for some referenced object. In this case, it is a `kappa` object that evaluates to the set of `(price,beds, pcode)` tuples satisfying the conditions that they are drawn from the 5-ary `rentals` relation, the price is less than 400, and the postcode is either SW6 or SW7. The content language used to express this query is FIPA KIF (Knowledge Interchange Format), under the real estate ontology.

The UWM might respond to this query with the following message (abbreviated for space):

```
inform:
  sender:
    agent-identifier:
      name: UWM
  receiver:
    set:
      agent-identifier:
        name: User1
  content: (= (kappa (?price ?beds ?pcode)
                     (and (rentals etc...)))
            ((250 1 SW7) (390 3 SW6)
             (340 2 SW6)))
```

This message specifies the `inform` performative, meaning that the sender (the UWM) wishes to inform the receiver (User1) of some fact. The content of the fact is that the previous query for rentals under £400 in SW6 or SW7 evaluates to the following set of tuples: a 1-bedroom property in SW7 at £250, a 3-bedroom property in SW6 at £390, and a 2-bedroom in SW6 at £340.

The querying agent can also subscribe to be informed of new results as the UWM's knowledge changes:

```
  subscribe:
    sender:
      agent-identifier:
        name: User1
    receiver:
      set:
        agent-identifier:
          name: UWM
    content: (kappa (?price ?beds ?pcode)
                    (and (rentals
                          ?price ?beds ?pcode
                          ?garden ?parking)
                        (> ?beds 5)))
```

This `subscribe` message sets up a persistent `query-ref` for properties having more than five bedrooms, which causes the receiver to `inform` the sender of the query results and to continue to send further `inform`s if the results change (for example, after extracting additional listings).

Related Work

BargainFinder (Krulwich 1996), one of the first web wrapper systems, used hand-coded wrappers to compare music prices at different online vendors. Subsequently, others developed semi-automatic toolkits to help write wrappers, for example in Ariadne (Ashish & Knoblock 1997) and XWRAP (Liu, Pu, & Han 2001).

A number of supervised learners have been built to generate wrappers automatically. ShopBot (Doorenbos, Etzioni, & Weld 1997) uses heuristics to learn to submit queries and parse their results, but is quite specific to online stores. Some domain knowledge is separated out into rules, but these describe field labels (e.g., "price"), whereas ours describe field contents (e.g., a dollar sign followed by a number). (Kushmerick 2000) defined various wrapper classes learnable from labelled examples. For example, an HLRT wrapper consists of a set of delimiters that mark the head of the page, the left and right boundaries of its fields, and the tail of the page. Given a set of training examples, wrappers are induced by searching all possible delimiters until a consistent set is found. Stalker (Muslea, Minton, & Knoblock 2001) expresses wrappers as trees where internal nodes represent record lists and leaves represent fields. The system extracts data by descending the tree to progressively refine the document segment to be extracted. At each node, extraction boundaries are defined by sequences of tokens, classes, and wildcards, learned by an incremental covering algorithm. In SoftMealy (Hsu & Dung 1998), wrappers are specified by so-called contextual rules associated with transitions in some finite-state transducer. A contextual rule is a disjunction of token sequences that marks the inside or outside of a field boundary. States in the transducer correspond to fields. (Chidlovskii 2001) describes another transducer-based approach where fields are attached to transitions instead of states. Fields are recognized by computing the minimal disjunction of prefixes and suffixes required to disambiguate them in the training examples.

(Cohen & Fan 1999) describes a site-independent method that learns general heuristics for extracting certain extremely simple types of sites. Rather than being trained on a single site, the learner is trained on many different sites to create a general-purpose wrapper. Web pages are regarded as HTML parse trees, and fields are found by classifying interior nodes as either positive or negative and extracting the text of subtrees rooted at positive nodes.

Another group of wrapper learners comes from the natural-language tradition. These systems are designed mainly for genres like email announcements and news articles that contain longer passages of free text and few markup tags. These include WHISK (Soderland 1999), SRV (Craven *et al.* 2000), and RAPIER (Califf & Mooney 1999).

Finally, some other unsupervised systems have been developed that build wrappers by examining unlabelled pages. (Embley *et al.* 1998) performs extraction using ontologies annotated with regular expressions that are similar to our domain rules. Rather than using document structure to constrain matches, they use heuristics such as keyword proximity and ontological knowledge about whether relationships are one-to-one or one-to-many to disambiguate each match individually. RoadRunner (Crescenzi, Mecca, & Merialdo 2001) takes a different approach by comparing two different documents from the same site that are assumed to have the same structure, and computing the minimal union-free regular expression that covers both. This expression corresponds directly to a wrapper structure.

Other unsupervised systems exist that are very restricted in the types of sites they can wrap. AutoWrapper (Gao & Sterling 1999) uses a similarity algorithm to induce wrappers for simple table structures. In this approach, pages are assumed to contain logical lines delimited by boundary tags. Adjacent lines are aligned and compared for similarity in terms of edit distance, then generalised to form an extraction pattern. MORPHEUS (Yang, Seo, & Choi 2001) is an extraction agent specialized for online shopping which assumes that input pages can be broken down into a header, a list of logical lines containing product descriptions, and a trailer. Each line is heuristically assigned a category such as product name or price. MORPHEUS simply looks for the most common sequence that begins with a name and ends with a price, and uses that as the extraction pattern.

Conclusion

In conclusion, we have described a universal wrapper machine that is able to take a set of general domain rules and learn to wrap sites without relying on manually-labelled training examples or *ad hoc* heuristics. It cleanly separates out site-independent and site-specific knowledge from execution implementation in a high-level way. Site-independent knowledge about domains is expressed using simple sets of rules that can be reused across a domain. Site-specific knowledge about the structure of pages on a particular site is expressed using context-free grammars. The universal wrapper machine brings these two components together by using the domain rules to semantically interpret the parse trees generated by the grammars. The resulting declarative wrapper specifications are easily understandable by humans and can be executed to perform information extraction. Once extracted, tuples can be queried by external agents using a high-level agent communication language, either as a one-time query or a persistent subscription.

Various directions for future work exist. It would be helpful to develop a monitoring capability that would enable the UWM to detect when a data source changes formats, so that it can reinvoke the grammar learner to repair the wrapper. Another enhancement would be to incorporate ontological knowledge in the form of XML schemas. Instead of extracting web documents to database form, the UWM could output tuples as XML documents conforming to some ontological schema, creating a migration bridge from plain HTML to the semantic web.

References

Ashish, N., and Knoblock, C. 1997. Semi-automatic wrapper generation for internet information sources. In *2nd IFCIS Int'l Conference on Cooperative Information Systems*, 160–169. Los Alamitos, CA: IEEE Computer Society.

Califf, M. E., and Mooney, R. J. 1999. Relational learning of pattern-match rules for information extraction. In *16th National Conference on Artificial Intelligence*, 328–334. Menlo Park, CA: AAAI Press.

Chidlovskii, B. 2001. Wrapping web information providers by transducer induction. In *12th European Conference on Machine Learning*, 61–72. Berlin: LNAI 2167, Springer-Verlag.

Cohen, W. W., and Fan, W. 1999. Learning page-independent heuristics for extracting data from web pages. In *8th International World Wide Web Conference*. Amsterdam: Elsevier Science.

Craven, M.; DiPasquo, D.; Freitag, D.; McCallum, A.; Mitchell, T.; Nigam, K.; and Slattery, S. 2000. Learning to construct knowledge bases from the world wide web. *Artificial Intelligence* 118:69–113.

Crescenzi, V.; Mecca, G.; and Merialdo, P. 2001. RoadRunner: Towards automatic data extraction from large web sites. In *27th International Conference on Very Large Data Bases*, 109–118.

Doorenbos, R.; Etzioni, O.; and Weld, D. 1997. A scalable comparison-shopping agent for the world-wide web. In *1st International Conference on Autonomous Agents*, 39–48. New York: ACM Press.

Embley, D. W.; Campbell, D. M.; Smith, R. D.; and Liddle, S. W. 1998. Ontology-based extraction and structuring of information from data-rich unstructured documents. In *1998 ACM CIKM International Conference on Information and Knowledge Management*, 52–59.

FIPA. 2001. FIPA ACL message structure specification. Specification XC00061, Foundation for Intelligent Physical Agents, Concord, CA.

Gao, X., and Sterling, L. 1999. AutoWrapper: Automatic wrapper generation for multiple online services. In *Asia Pacific Web Conference (APWeb '99)*, 61–70. Las Vegas, NV: CSREA Press.

Hong, T. W., and Clark, K. L. 2001. Using grammatical inference to automate information extraction from the web. In *5th European Conference, PKDD 2001*, 216–227. Berlin: LNAI 2168, Springer-Verlag.

Hong, T. 2003. *Grammatical Inference for Information Extraction and Visualisation on the Web*. Ph.D. Dissertation, Imperial College, London, United Kingdom.

Hsu, C.-N., and Dung, M.-T. 1998. Generating finite-state transducers for semi-structured data extraction from the web. *Information Systems* 23(8):521–538.

Kistler, T., and Marais, H. 1998. WebL—a programming language for the web. *Computer Networks and ISDN Systems* 30:259–270.

Krulwich, B. 1996. Information integration agents: BargainFinder and NewsFinder. In *Internet-Based Information Systems: Papers from the 1996 AAAI Workshop*, 79. Menlo Park, CA: AAAI Press.

Kushmerick, N. 2000. Wrapper induction: Efficiency and expressiveness. *Artificial Intelligence* 118:15–68.

Liu, L.; Pu, C.; and Han, W. 2001. An XML-enabled data extraction toolkit for web sources. *Information Systems* 26:563–583.

Muslea, I.; Minton, S.; and Knoblock, C. A. 2001. Hierarchical wrapper induction for semistructured information sources. *Journal of Autonomous Agents and Multi-Agent Systems* 4:93–114.

Soderland, S. 1999. Learning information extraction rules for semi-structured and free text. *Machine Learning* 34(1–3):233–272.

Yang, J.; Seo, H.; and Choi, J. 2001. MORPHEUS: A more scalable comparison-shopping agent. In *Fifth International Conference on Autonomous Agents*, 63–64. New York: ACM Press.