Adaptive information extraction: Core technologies for information agents

Nicholas Kushmerick[1] Bernd Thomas[2]

[1] Computer Science Department, University College Dublin; email: nick@ucd.ie
[2] Institut für Informatik, Universität Koblenz-Landau; email: bthomas@uni-koblenz.de

1 Introduction

For the purposes of this chapter, an information agent can be described as a distributed system that receives a goal through its user interface, gathers information relevant to this goal from a variety of sources, processes this content as appropriate, and delivers the results to the users. We focus on the second stage in this generic architecture. We survey a variety of information extraction techniques that enable information agents to automatically gather information from heterogeneous sources.

For example, consider an agent that mediates package-delivery requests. To satisfy such requests, the agent might need to retrieve address information from geographic services, ask an advertising service for freight forwarders that serve the destination, request quotes from the relevant freight forwarders, retrieve duties and legal constraints from government sites, get weather information to estimate transportation delays, etc.

Information extraction (IE) is a form of shallow document processing that involves populating a database with values automatically extracted from documents. Over the past decade, researchers have developed a rich family of generic IE techniques that are suitable for a wide variety of sources, from rigidly formatted documents such as HTML generated automatically from a template, to natural-language documents such as newspaper articles or email messages.

In this chapter, we view information extraction as a core enabling technology for a variety of information agents. We therefore focus specifically on information extraction, rather than tangential (albeit important) issues, such as how agents can discover relevant sources or verify the authenticity of the retrieved content, or caching policies that minimize communication while ensuring freshness.

Before proceeding, we observe that neither XML nor the Semantic Web initiative will eliminate the need for automatic information extraction. First, there are terabytes of content available from numerous legacy services that will probably never export their data in XML. Second, it is impossible to determine "the" correct annotation scheme, and applications will have their own idiosyncratic needs ("should the unit of currency be included when extracting prices?", "should people's names be split into first and surname?", "should dates such as 'Sun. May 14, 67' be canonicalized to 14/05/1967?"). For these reasons we expect that automatic information extraction will continue to be essential for many years.

Scalability is the key challenge to automatic information extraction. There are two relevant dimensions. The first dimension is the ability to rapidly process large document

collections. IE systems generally scale well in this regard because they rely on simple shallow extraction rules, rather than sophisticated (and therefore slow) natural language processing.

The second and more problematic dimension is the number of distinct sources. For example, a package-delivery agent might need to request quotes from a thousand different freight forwarders, weather information from dozens of forecast services, etc. IE is challenging in this scenario because each source might format its content differently, and therefore each source could require a customized set of extraction rules.

Machine learning is the only domain-independent approach to scaling along this second dimension. This chapter focuses on the use of machine learning to enable *adaptive information extraction* systems that automatically learn extraction rules from training data in order to scale with the number of sources.

The general idea behind adaptive information extraction is that a human expert annotates a small corpus of training documents with the fragments that should be extracted, and then the learning system generalizes from these examples to produce some form of knowledge or rules that reliably extract "similar" content from other documents. While human-annotated training data can be expensive, the assumption underlying adaptive IE is that it is easier to annotate documents than to write extraction rules, since the latter requires some degree of programming expertise. Furthermore, we will describe techniques aimed at minimizing the amount of training data required for generalization, or even eliminating the need for manual annotation entirely.

The adaptive information extraction research community has developed a wide variety of techniques and approaches, each tailored to particular extraction tasks and document types. We organize our survey of this research in terms of two distinct approaches. First, we describe *finite-state* approaches that learn extraction knowledge that is equivalent to (possibly stochastic) finite-state automata (Section 2). Second, we describe *relational* approaches that learn extraction knowledge that is essentially in the form of Prolog-like logic programs (Section 3). For the sake of brevity we can not describe these techniques in detail, but see [26] for more information about some of these ideas.

2 Finite-state techniques

Many approaches to Web information extraction can be categorized as finite-state approaches, in that the learned extraction knowledge structures are formally equivalent to (possibly stochastic) regular grammars or automata. In this section we survey several prominent examples, as well as some additional research that relates to the entire wrapper "life-cycle" beyond the core learning task.

2.1 Wrapper induction

Kushmerick first formalized adaptive Web information extraction with his work on *wrapper induction* [21, 17, 19]. Kushmerick identified a family of six wrapper classes, and demonstrated that the wrappers were both relatively expressive (they can learn wrappers for numerous real-world Web sites), and also relatively efficient (only a handful of training examples, and a few CPU seconds per example, are needed for learning).

To illustrate Kushmerick's wrapper induction work, consider the example Web page shown in Figure 1(a), its HTML encoding (b), and the content to be extracted (c). This example is clearly extremely simple, but it exhibits all the features that are salient for our discussion.

Kushmerick's wrappers consist of a sequence of delimiters strings for finding the desired content. In the simplest case (shown in Figure 1(d–e)), the content is arranged in a tabular format with K columns, and the wrapper scans for a pair of delimiters for each column, for a total of $2K$ delimiters. The notation "ℓ_k" indicates the left-hand delimiter for the k'th column, and "r_k" is the k'th column's right-hand delimiter. In this case of the country-code wrapper $\mathsf{ccwrap}_{\mathrm{LR}}$, we have $K = 2$.

To execute the wrapper, procedure $\mathsf{ccwrap}_{\mathrm{LR}}$ (Figure 1(d)) scans for the string $\ell_1 =$<B> from the beginning of the document, and then scans ahead until the next occurrence of $r_1 =$</B>. The procedure then extracts the text between these positions as the value of the first column of the first row. The procedure then scans for $\ell_2 =$<I> and then for $r_2 =$</I>, and extracts the text between these positions as the value of the second column of the first row. This process then starts over again with ℓ_1; extraction terminates when ℓ_1 is missing (indicating the end of the document).

Figure 1(e) formalizes these ideas as the *Left-Right* (LR) wrapper class. An LR wrapper w_{LR} consists of a set $\{\langle \ell_1, r_1 \rangle, \ldots, \langle \ell_K, r_K \rangle\}$ of $2K$ delimiters, one pair for each column to be extracted, and the "operational semantics" of LR are provided by the $\mathsf{exec}_{\mathrm{LR}}$ procedure (Figure 1(e)). This procedure scans for ℓ_1 from the beginning of the document, and then scans ahead until the next occurrence of r_1. The procedure then extracts the text between these positions as the value of the first column of the first row. $\mathsf{ccwrap}_{\mathrm{LR}}$ then scans for ℓ_2 and then for r_2, and extracts the text between these positions as the value of the second column of the first row. This process is repeated for all K columns. After searching for r_K, the procedure starts over again with ℓ_1; extraction terminates when ℓ_1 is missing (indicating the end of the document).

Given this definition, the LR machine learning task is to automatically construct an LR wrapper, given a set of training documents. LR learning is relatively efficient, because the $2K$ delimiters can all be learned independently. The key insight is that whether a particular candidate is valid for some delimiter has no impact on the other delimiters. Based on this observation, Kushmerick describes a quadratic-time algorithm for learning LR wrappers. The algorithm simply enumerates over potential values for each delimiter, selecting the first that satisfies a constraint that guarantees that the wrapper will work correctly on the training data. Kushmerick demonstrates (both empirically, and theoretically under the PAC model) that this algorithm requires a modest training sample to converge to the correct wrapper.

Of course, just because an efficient learning algorithm exists does not mean that the wrappers are useful! Below, we discuss the limitations of the LR class and show that it can not handle documents with more complicated formatting. However, even the very simple LR class was able to successfully wrap 53% of Web sites, according to a survey. While LR is by no means a definitive solution to Web information extraction, it clearly demonstrates that simple techniques can be remarkably effective.

LR is effective for simple pages, but even minor complications to the formatting can render LR ineffective. For example, consider ℓ_1. The LR class requires a value

(a)

(b)

```
<HTML><TITLE>Some Country Codes</TITLE><BODY>
<B>Congo</B> <I>242</I><BR>
<B>Egypt</B> <I>20</I><BR>
<B>Belize</B> <I>501</I><BR>
<B>Spain</B> <I>34</I><BR>
</BODY></HTML>
```

(c)

$$\left\{ \begin{array}{c} \langle \text{`Congo'}, \text{`242'} \rangle, \\ \langle \text{`Egypt'}, \text{`20'} \rangle, \\ \langle \text{`Belize'}, \text{`501'} \rangle, \\ \langle \text{`Spain'}, \text{`34'} \rangle \end{array} \right\}$$

(d)

procedure $\mathsf{ccwrap}_{\mathrm{LR}}$(page P)
  while there are more occurrences in P of '<B>'
    for each $\langle \ell_k, r_k \rangle \in \{\langle$'<B>', '</B>'$\rangle, \langle$'<I>', '</I>'$\rangle\}$
      scan in P to next occurrence of ℓ_k; save position as start of k'th attribute
      scan in P to next occurrence of r_k; save position as end of k'th attribute
  return extracted $\{\ldots, \langle \text{country}, \text{code} \rangle, \ldots\}$ pairs

(e)

procedure $\mathsf{exec}_{\mathrm{LR}}$(wrapper $w_{\mathrm{LR}} = \{\langle \ell_1, r_1 \rangle, \ldots, \langle \ell_K, r_K \rangle\}$, page P)
  $m \leftarrow 0$
  while there are more occurrences in P of ℓ_1
    $m \leftarrow m+1$
    for each $\langle \ell_k, r_k \rangle \in \{\langle \ell_1, r_1 \rangle, \ldots, \langle \ell_K, r_K \rangle\}$
      scan in P to the next occurrence of ℓ_k; save position as $b_{m,k}$
      scan in P to the next occurrence of r_k; save position as $e_{m,k}$
  return label $\{\ldots, \langle \langle b_{m,1}, e_{m,1} \rangle, \ldots, \langle b_{m,K}, e_{m,K} \rangle \rangle, \ldots\}$

Fig. 1. A fictitious Internet site providing information about countries and their telephone country codes: (a) an example Web page; (b) the HTML document corresponding to (a); (c) the content to be extracted; (d) the $\mathsf{ccwrap}_{\mathrm{LR}}$ procedure, which generates (c) from (b); and (e) the $\mathsf{exec}_{\mathrm{LR}}$ procedure, a generalization of $\mathsf{ccwrap}_{\mathrm{LR}}$.

for ℓ_1 that reliably indicates the beginning of the first attribute. However, there may be no such delimiter. For example, suppose that Figure 1(b) was modified to include a heading `<B>Country code list</B>` at the top of the document. In this case the delimiter $\ell_1 =$`<B>` used by $\mathsf{ccwrap}_{\mathrm{LR}}$ would not work correctly. Indeed, it is possible to show that there is no legal value for ℓ_1 and hence no LR wrapper for documents modified in this manner.

Kushmerick tackled these issues by extending LR to a family of five additional wrapper classes. First, the *Head-Left-Right-Tail* (HLRT) class uses two additional delimiters to skip over potentially-confusing text in either the head (top) or tail (bottom) of the page. In the example above, a head delimiter h (such as $h =$`list`) could be used to skip over the initial `<B>` at the top of the document, enabling $\ell_1 =$`<B>` to work correctly. Alternatively, the *Open-Close-Left-Right* (OCLR) class uses two additional delimiters to identify an entire tuple in the document, and then uses the regular LR strategy within this mini-document to extract each attribute in turn. These two ideas can be combined in fourth wrapper class, the *Head-Open-Close-Left-Right-Tail* (HOCLRT) class.

Finally, Kushmerick explored two simple wrappers for data that is not formatted in a simple tabular fashion. The *Nested-Left-Right* (NLR) class can be used to extract hierarchically-organized data, such as a book's table of contents. NLR operates like LR except that, after processing r_k, there are $k+1$ possibilities (start at level $k+1$, continue level k, return to level $k-1, \ldots$, return to level 1) instead of just one (proceed to attribute $k+1$). The *Nested-Head-Left-Right-Tail* (NHLRT) class combines NLR and HLRT.

Kushmerick developed specialized learning algorithms for each of these five classes. He demonstrated, both empirically and using complexity theory, that there is a trade-off between the expressive power of the wrapper classes and the extent to which they can be efficiently learned. For example, even though the six classes can successfully wrap 70% of surveyed sites, the algorithms for learning NLR and NHLRT wrappers take time that grows exponentially in the number of attributes, and a PAC analysis reveals that HOCLRT requires substantially more training examples to converge compared to the other classes.

2.2 More expressive wrapper classes

Following Kushmerick's initial investigation of the LR family of wrappers, there has been substantial research effort at elaborating various alternative wrapper classes, and deriving more efficient learning algorithms. Even when Kushmerick's various extended wrapper classes are taken into consideration, there are numerous limitations. Muslea et al [27], Hsu and Dung [14], and others have developed various wrapper-learning algorithms that address the following shortcomings:

Missing attributes. Complicated pages may involve missing or null attribute values. If the corresponding delimiters are missing, then a simple wrapper will not process the remainder of the page correctly. For example, a French e-commerce site might only specify the country in addresses outside France.

Multi-valued attributes. The simple wrapper classes discussed so far assume a simple relational model in which each attribute has a single value, but non-relational structures such as multi-valued attributes are natural in many scenarios. For example, a

hotel guide might explicitly list the cities served by a particular chain, rather than use a wasteful binary encoding of all possible cities.

Multiple attribute orderings. The wrappers described so far assume that the attributes (and therefore the delimiters) will occur in one fixed ordering, but variant orderings abound in complicated documents. For example, a movie site might list the release date before the title for movies prior to 1999, but after the title for recent movies.

Disjunctive delimiters. The wrappers discussed above assume a single delimiter for each attribute, but complicated sites might use multiple delimiters. For example, an e-commerce site might list prices with a bold face, except that sale prices are rendered in red.

Nonexistent delimiters. The wrappers described earlier assume that some irrelevant background tokens separate the content to be extracted, but this assumption may be violated in some cases. For example, how can the department code be separated from the course number in strings such as "COMP4016" or "GEOL2001". This problem is also relevant for many Asian languages in which words are not tokenized by spaces.

Typographical errors and exceptions. Real-world documents may contain errors, and if these errors occur in the formatting that drives extraction, then a simplistic wrapper may fail on the entire page even if just a small portion is badly formatted.

Sequential delimiters. So far, the wrapper classes above assumed a single delimiter per attribute, but the simplest way to develop an accurate wrapper might be to scan for several delimiters in sequence. For example, to extract the name of a restaurant from a review it might simpler to scan for `<B>`, then to scan for `<BIG>` from that position, and finally to scan for `<FONT>`, rather than to force the wrapper to scan the document for a single delimiter that reliably indicates the extracted content.

Hierarchically organized data. Kushmerick's nested classes are a first step at handling non-tabular data, but his results are largely negative. In complicated scenarios there is a need extraction according to a nested or embedded structure.

Hsu and Dung [14] addresses the problem of learning wrappers that correspond to an expressive class of deterministic finite-state transducers. This formalism handles all but the last two requirements just mentioned. The transducer processes the document to extract a single tuple; after extraction control returns to the start state and the second tuple is extracted, etc. Each extracted attribute is represented as a pair of states: one state to identify the start of the attribute value and the second to identify the end.

Since a general automaton model is used, states can be connected in an arbitrary manner, permitting missing attributes (skipped states), multi-valued attributes (cycles) and multiple attribute orderings (multiple paths from the start to end state). Furthermore, state-transitions are governed by an expressive rule language that allows disjunctive delimiters. A limited form of exception-processing is permitted, allowing the system to recover from formatting errors and exceptions. Crucially, Hsu and Dung describe an algorithm for efficiently learning their wrapper transducers from training data. Empirically, the report that their wrapper classes handles the 30% sites that could not be wrapped by Kushmerick's wrapper classes.

Muslea et al [27] identify a class of wrappers that, unlike Hsu and Dung, tackle the last two issues mentioned above. The main distinguishing feature of Muslea et al's

wrappers is the use of multiple delimiters that they call landmarks. Rather than insisting that there exist a single delimiter that exactly identifies the relevant position deep inside some document, landmark-based wrappers use a sequence of delimiters to jump to the appropriate position in a series of simple steps. These simple steps are usually easier to learn, and enable more robust extraction. A second major feature of Muslea et al's work is that their "embedded catalog" formalization of nested data is more expressive than the simple hierarchical approach used by Kushmerick.

2.3 Extraction from natural text

The techniques described so far are aimed at highly regular documents, such as machine-generated HTML emitted by CGI programs. However, most research on information extraction has focused on natural free-text documents, such as email messages, newspaper articles, resumes, etc. Are the "wrapper" results relevant to these less structured domains. Several recent investigations have shown promising results.

Freitag and Kushmerick [11] explore "boosted wrapper induction". They define a class of extraction patterns that is essentially the LR class, for the case when there is exactly $K = 1$ attributes. They then enrich this class by permitting delimiters to contain wild-cards over token types (eg, `<Num>` rather than specific instances such as `23`).

For example, for a corpus of email seminar announcements, the algorithm learns the following rule for extracting the starting time: {([`time :`],[`<Num>`]), ([], [`- <Num> : <*> <Alpha>`])}, which matches a document such as "...`Time: 2:00 - 3:00 pm` ...", where the fragment to be extracted has been underlined. This rule basically says "to find the start of the time, look for 'time:' followed by any number; then find the end of the time by looking for a dash, another number, a colon, any token at all, and finally an alphanumeric token".

This simple rule language is by itself not very useful for extraction from free text. Freitag and Kushmerick improve the performance by using boosting (a general technique for improving the accuracy of a weak learning algorithm) to learn many such rules. Each individual rule has high precision, but low recall; when combined, the rule set has both high precision and high recall. The result is an accurate extraction algorithm that is competitive with other state-of-the-art approaches in a variety of free-text domains, and superior in many. For example, boosted wrapper induction performs essentially perfectly at the task of extracting seminar announcement times, and better than most competitors at other attributes such as the speaker name and seminar location.

Soderland [32] describes a related approach to using finite-state techniques for information extraction from free text. Soderland's extraction rules correspond to a restricted class of regular expressions. These regular expressions serve two purposes: they can be both contextual pattern for determining whether a particular fragment should be extracted, or delimiters for determining the precise boundaries of the target fragment. Soderland's language is important because it is designed to work for documents that span the spectrum from unstructured natural text through to highly structured Web pages. Depending on the degree of structure in the training documents, the learning algorithm automatically creates appropriate patterns. For example, if simple delimiter-based extraction is sufficiently accurate then the learning algorithm will not bother to add additional contextual constraints.

For example, consider extracting the price and number of bedrooms from apartment listing documents such as "`Capitol Hill- 1 br twnhme. D/W W/D. Pkg incl $675. 3BR upper flr no gar. $995. (206) 999-9999`". Soderland's system learns rules such as "`* (<Digit>) 'BR' * '$' (<Numb>)`", where the parenthesized portions of the regular expression indicate the values to be extracted. This rule would extract the content $\{($`1`, `675`$), ($`3`, `995`$)\}$ from the example document.

2.4 Hidden Markov models

The work of Freitag and Kushmerick [11] and Soderland [32] are two instances of generalizing finite-state approaches from rigidly structured HTML documents to less structured documents such as email and newspaper articles. However, these approaches are still brittle because they do not have any facility for evaluating the strength of the evidence that guides extraction decisions. For example, suppose the phrase `will be held in` often precedes a seminar location, but a new document contains the typographical error `will held in`. The techniques described so far make binary decisions and thus have no way to use this uncertain evidence.

Hidden Markov Models are a principled and efficient approach to handling this sort of inherent uncertainty. A Hidden Markov Model (HMM) is a stochastic finite-state automaton. States emit tokens according to a fixed and state-specific distribution, and transitions between states occur according to a fixed distribution. HMMs are an attractive computational device because there are efficient algorithms for both learning the model's distribution parameters, and for inferring the most-likely state sequence given some observed token sequence.

To use HMMs for information extraction, states are associated with the tokens to be extracted. For example, with the email seminar announcement corpus, the HMM would contain a state for the start time tokens, the end time tokens, the speaker name tokens, and the location tokens. Optionally, there may be additional states that generate "background" tokens. To perform extraction, the standard HMM Viterbi decoding algorithm is used to determine the most-likely state-sequence to have generated the observed document, and then the extracted fragments can simply by read off this most-likely path.

Hidden Markov models have been used successfully by numerous researchers in a variety of extraction scenarios (eg, [3, 22]). They key challenge is that there is no efficient general-purpose algorithm for determining an appropriate state topology (ie, which state-state distribution probabilities should be forced to be zero and which should be permitted to be positive). Initial work has generally used a hand-crafted topology, in which the states are connected manually in a "reasonable" way after evaluating the training corpus.

More recently, there have been several attempts to automatically learn an appropriate topology. The general approach is to greedily search the space of possible topologies for one that maximizes some objective function. Seymore et al [31] attempt to maximize the probability of the training data given the topology. This approach is reasonably efficient but potentially misguided: the goal of using an HMM is not to model the training data per se, but to perform accurate extraction. Freitag and McCallum [12] therefore use as the objective function the actual accuracy of the proposed topology for extrac-

tion from a held-out validation corpus. While this approach is significantly slower it can result in a more compact topology and better generalization.

2.5 Wrapper maintenance

All of the wrapper-learning work described earlier ignores an important complication. Information agents generally have no control over the sources from which they receive data. As described above, the agent's wrappers tend to be relatively brittle, as the invariably rely on idiosyncratic formatting details observed during the learning process. Unfortunately, if the source modifies its formatting (for example, to "remodel" its user interface) then the observed regularities will no longer hold and the wrapper will fail. As a concrete example, Figure 2 and Figure 3 show the Altavista search engine, before and after a site redesign.

The two key challenges to wrapper maintenance are *wrapper verification* (determining whether the wrapper is still operating correctly), and *wrapper reinduction* (learning a revised wrapper). The second challenge is considerably more difficult, although even wrapper verification is non-trivial. The difficulty is that at most web sites, either the content to be extracted, or the formatting regularities, or both, may have changed, and the verification algorithm must distinguish the two. For example, suppose that the change in the Microsoft stock price is checked three times at a stock-quote server, and the extracted values are `+3.10`, `-0.61` and `<B><IMG src=advert.gif>`. Intuitively our verification algorithm should realize that the relatively the first two values are "similar" and do not indicate trouble, but the third value is an outlier and probably indicates a defective wrapper.

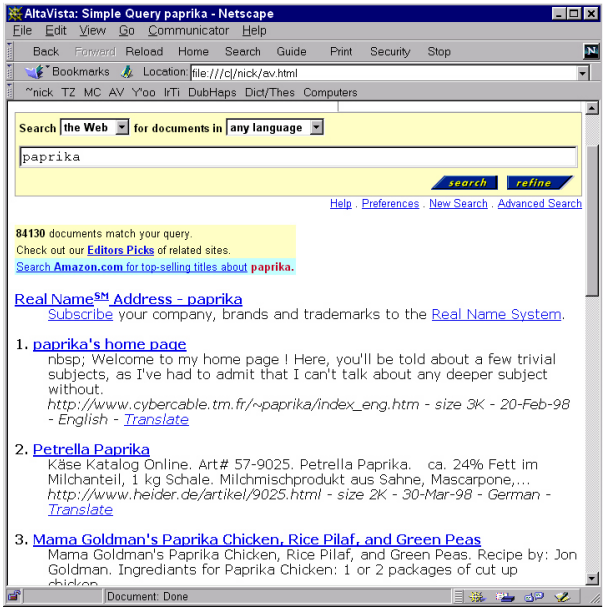

Fig. 2. Altavista snapshot before redesign

Kushmerick [18, 20] describes a simple and accurate algorithm for wrapper verification. The algorithm first learns a probabilistic model of the data extracted by the wrapper during a training period when it is known to be operating correctly. This model captures various properties of the training data such as the length or the fraction of numeric characters of the extracted data. To verify the wrapper after the training period,

the extracted data is evaluated against the learned model to estimate the probability that wrapper is operating correctly. The algorithm is domain independent and is not tied to any particular wrapper class or learning algorithm, but rather treats the wrapper as a black-box and inspects only its output. The algorithm handles (acyclic) XML data, not just relational data, so it is applicable to all of the wrapper classes described above.

Wrapper re-induction has also received some attention. Lerman et al [23] learn a probabilistic model of the extracted data that is similar to (though substantially more expressive than) that used by Kushmerick. This more sensitive model enables wrapper re-induction as follows. After a wrapper is deemed to be broken, the learned model is used to identify probable target fragments in the (new and unannotated) documents. This training data is then post processed to (heuristically) remove noise, and the data is given to a wrapper induction algorithm. Lerman et al demonstrate empirically that this semi-supervised approach is highly accurate in many real-world extraction scenarios.

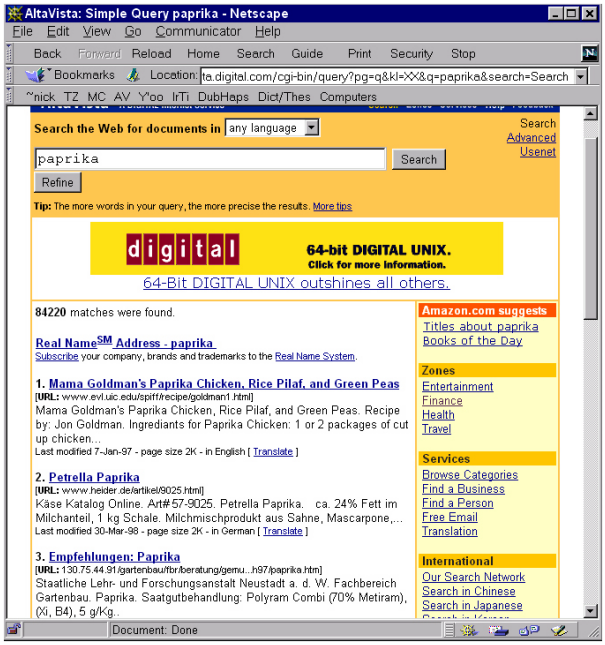

Fig. 3. Altavista snapshot after redesign

2.6 Post-processing extracted content

The work described so far is highly simplified in that the task is assumed to involve simply processing a given document to extract particular target fragments. However, in many extraction scenarios, the information to be extracted is actually distributed across multiple documents, or an attribute value is given only once on a page but is relevant to several extracted objects. For example, Figure 4 shows a simple scenario in which some attribute values are “re-used” across multiple extracted objects, and other values must be harvested from a collection of hyperlinked documents.

Some of these issues are handled by the wrapper classes defined earlier. For example, Muslea et al’s embedded catalog formalism [27] permits an extracted fragment to be “shared” across multiple objects. Furthermore, the information extraction community has long investigated the issue of cross-document references. However, these

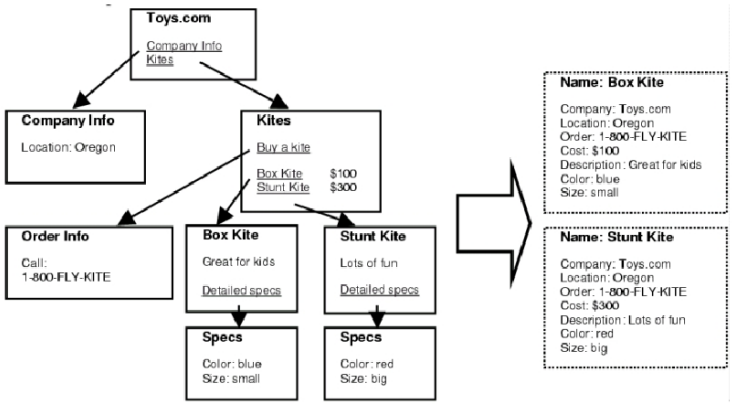

Fig. 4. A complicated extraction task in which attribute values are both distributed across multiple documents, and reused across objects. (Adapted from [15].)

approaches require considerable linguistic processing and are not applicable to the example shown in Figure 4.

Jensen and Cohen [15] address these problems by proposing a language for specifying how the extracted data should be post-processed. Rules express how the raw extracted data should be grouped into larger composite objects. Jensen and Cohen argue that their language is sufficiently expressive to handle the data extracted from 500 web sites exporting job and product advertisements. Furthermore, they suggest (though do not implement) an algorithm for automatically learning such rules from examples of grouped data.

2.7 Beyond supervision

The key bottleneck with adaptive information extraction is obtaining the labeled training data. The use of machine learning is motivated by the fact that the cost of labeling documents is usually considerably less than the cost of writing the wrapper's extraction rules by hand. Nevertheless, labeling documents can require considerable domain expertise, and is generally tedious and error-prone. The approaches described so far simply assumes that an adequate training corpus exists, but considerable research effort has investigated so-called "active learning" methods for minimizing the amount of training data required to achieve a satisfactory level of generalization.

The basic idea of active learning is to start with a small amount of training data, run the learning algorithm, and then used the learned wrapper to predict which of the remaining unlabeled documents is most informative, in the sense of helping the learning system generalize most with the one additional training document. As a trivial example,

if the corpus contains duplicate documents, then the learner should not suggest that the same document be annotated twice.

As one example of the use of active learning in the context of wrapper induction, consider Muslea et al [28]. The basic idea of this approach is that every information extraction task has a "dual", and correlations between the original task and its dual can help the system identify useful unlabeled documents.

Recall that Muslea et al's wrapper learning algorithm learns a sequence of landmarks for scanning from the beginning of the document to the start of a fragment to be extracted. An alternative way of finding the same position is to scan backwards from the end of the document for a (different!) set of landmarks. Muslea's active-learning extensions solves both learning tasks in parallel on the available training data. The two resulting wrappers are then applied to all the unlabeled documents. The system then asks the user to label one of the documents for which the two wrappers give different answers. Intuitively, if the two wrappers agree for a given unlabeled document, then the document is unlikely to be useful for subsequent learning.

Muslea et al demonstrate that the active-learning version of their algorithm requires significantly less training data to obtain the same level of generalization. For example, averaged across a variety of challenging extraction tasks, the error of the learned wrapper is about 50% less when the user annotates ten training documents chosen in this intelligent manner, compared to ten documents chosen randomly.

Brin [4] explores a different sort of extraction task, in which the user gives the system examples of some concept. For example, to learn to extract book title/author pairs, the user would supply a small sample of pairs, such as {(`Isaac Asimov`, `The Robots of Dawn`), (`Charles Dickens`, `Great Expectations`), ...}. The job of the extraction system is then to flesh out this list with as many additional instances as possible.

Brin's algorithm iteratively searches the Web for the seed pairs. When it finds a document that contains a pair, it learns an information extraction pattern for that particular pair, and then applies this pattern to the remainder of the page. The resulting extracted pairs are added to the seeds and the process iterates. There is no guarantee that this process will converge or even that the extracted pairs are correct. Nevertheless, preliminary experiments demonstrated promising results.

Finally, Crescenzi et al [9] focus on an even bigger challenge: wrapper learning without any supervision (labeled training data) at all. Consider the pages from some online bookstore that would be returned by two queries, for Dickens and for Asimov. In most cases, these pages would be formatted the same way, with the only difference being the content to be extracted. The intuition behind Crescenzi et al's approach is that a wrapper can be learned by comparing these two pages and finding similarities and differences. The similarities correspond to common formatting and structural elements; the differences correspond to data to be extracted. By repeatedly replacing the differences with wild-cards and noting repetitive structures, their algorithm can learn a wrapper that corresponds to a regular grammar, without the need for any manually labeled training data. Crescenzi et al report that their algorithm works well in a variety of real-world domains.

3 Relational learning techniques

In this section we introduce adaptive IE systems that use relational learning techniques. We present a short introduction to common relational rule induction algorithms and how they are used as a basis in several information extraction systems. We do not focus on the formal foundations of inductive logic programming [25, 1]; our goal is to provide a summary of relational rule induction approaches as they have been used for adaptive IE over the past decade.

Before introducing the basic concepts of rule induction let us give a short motivation for using relational techniques for learning wrappers. Several existing techniques for IE (like HMM's presented in Section 2) are based on the assumption to determine relevant text parts to be extracted by statistical means. These finite-state techniques can be seen as some sort of rule learning, ie the learning of production (grammar) rules constrained by certain probability measures. When we talk about rule learning in the context of relational rule learning we have logical rules in mind, in the sense of learning rules of first order predicate logic or at least subsets of first order rules like Horn rules or Prolog programs. As we will see, while the different rules learned by the various IE systems vary in their representation and signature, in general they can all be rewritten in a uniform predicate logic representation.

Talking about learning logical rules in combination with IE only makes sense if we abstract from the pure lexical representation of documents. Thus a first step to use relational learning techniques is to find a suitable document representation suited to the formal framework of logic, literals and logic rules. The most common representation used in relational IE is to interpret a document as a sequence of feature terms or tokens having several attributes describing features of grouped symbols from the document. How the tokenization is done is subject a) to the type of extractions needed, and b) the learning methods used. For example, the relevant features may be its type (integer, char, HTML tag), whether it is upper or lower case, its length, linguistic knowledge about the word category, its genus or even additional semantic knowledge drawn from a rich taxonomy.

For example Thomas [34] uses a document transformation into feature terms, in which a fragment like `<b>Pentium 90</b>` is written as a list of tokens: *[token(type=html, tag=b), token(type=word, txt='Pentium'), token(type=int, val=90), token(type=html_end, tag=b)]*. If we replace a feature value like *b* in *token(type=html, tag=b)* with a variable *token(type=html, tag=X)*, we can a) describe the class of all tokens of type *html*, and b) use unification methods to find all non-closing HTML tags in a tokenized document.

It then becomes obvious how more complex patterns can be defined by the use of first order predicate rules. For example, the rule *link(Description, Url) :- pos(P, token(type=html, tag=a, href=Url)), sequence(P, E, TokenSeq), not in(token(type=html_end, tag=a), TokenSeq), next(E, token(type=html_end, tag=a))* extracts tuples of the form *<Description, URL>* from a HTML document. For further details of how logic programs can be used for information extraction, see [35]. In the last decade various representations have been developed, some influenced largely by logic programming [16, 34], and other slot-oriented approaches motivated by natural language processing. In essence they all can be represented without much effort in a first order predicate logic syntax.

Additional representations may be used to capture the documents layout. For example, a parse tree of the HTML structure can give information on paragraphs, titles, subsections, enumerations, tables. For information extraction tasks tailored for HTML or XML documents, the Document Object Model (DOM) can provide additional information into the learning and the later extraction process. Systems like that of Cohen [8] and the wrapper toolkit of the MIA system (Section 3.3) make use of such representations. In general the document representation can not be considered to be independent from the extraction task. For example, if someone wants to extract larger paragraphs from free natural language documents she probably will use a document representation and therefore relational representation reflecting larger document blocks, compared to someone interested in prices from online catalogues. Nevertheless if relational methods are to be used no matter which representation is chosen it must be presented in terms of relations.

Section 2 already presented several state of the art adaptive IE algorithms, so what might be the shortcoming of these systems and what might be the advantages of relational rule based IE systems? One answer is that of human readability. A learned rule in the relational approach has a clear conclusion and conjunctive set of premises, which are all understandable because they refer to certain easily recognizable features of the document representation. This of course only holds if the document representation itself is clear and understandable. Thus such a learned rule will also give a clear explanation why it is used for extraction. Another strong argument is that of extending relational learning approaches. Having sets of first order rules in mind, it becomes apparent that adding additional background knowledge like ontologies or additional domain knowledge is easy to do. And even more important this additional knowledge can also be incorporated into the rule construction algorithm, and the inductive rule learning calculus can be extended by reasoning components from automated deduction systems. For example, some IE systems make use of additional semantic knowledge derived from a taxonomy [5, 33].

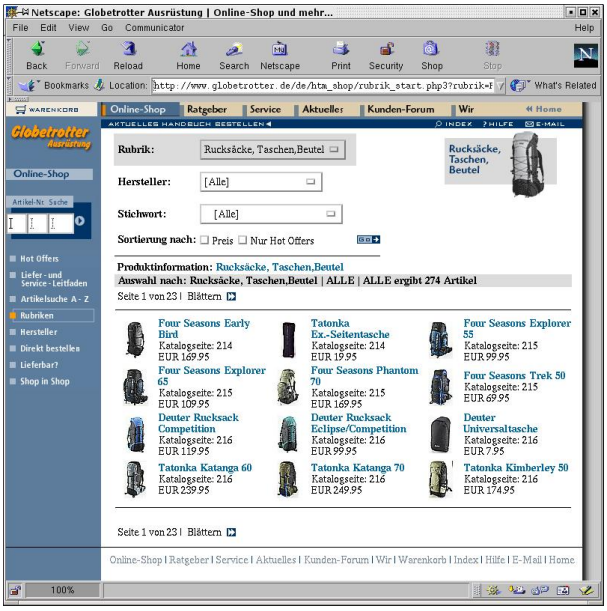

Fig. 5. An online catalogue

By now we can think of relational learning as a core algorithm that expects a set of examples in the form of relation instances, some additional knowledge (the background theory) and a set of predicates from which rules can be build. The algorithm tries to construct rules using the background theory and these special predicates such that the new rules explain (cover) the presented positive examples and exclude (if given) the negative ones. Applying this approach to the tasks of IE involves the use of a appropriate relational document representation, text examples as grounded facts, and additional predicates plus a background theory used to check certain features of text fragments and tokens. In theory the core algorithm is not affected by the representation: by choosing the right representation, a standard relational rule induction algorithm can be used as core learning algorithm for adaptive IE. In practice it turns out that due to complexity issues modification and tailored approaches are needed, but the important point is the theoretical framework of learning logical rules [1] provides a well understood and formal basis for adaptive information extraction.

3.1 Rule Learning

Because IE involves extracting certain fragments from a document (where the key idea is that the extracted fragments are rule variable bindings), we will not discuss propositional rule learning. Since we adopt the general approach from Section 2 that extraction rules consist of delimiters and slots (extraction variables), we are confronted with the problem of inducing left and right delimiters. Additionally many approaches also try to induce some information about the extractions itself, by recognizing certain specific features from the provided example fragments. For example they generalize starting from certain linguistic or semantic features. So far almost all existing relational approaches are using one of the following techniques.

One-shot learning. Given a set of positive examples and perhaps negative examples plus additional information needed (eg the documents the examples are drawn from), these approaches try to learn a rule in one step. One shot learning approaches do no refinement and evaluation at each step during the rule building process, instead they assume their applied learning operators are good enough or they pass the evaluation and further refinement of learned rules to the user. The Autoslog [30] system and the T-Wrapper system [34] use one shot learning approaches.

Sequential covering. In contrast to One Shot Learning a sequential covering involves an iterative process of refinement and testing. The general Sequential Covering algorithm is shown in Figure 6. This algorithm learns a disjunctive set of rules depending on a threshold with regard to the performance of a learned rule. In other words, the learned rule set may still cover some negative examples. The algorithm repeatedly tries to learn one rule meeting the threshold condition. As long as the threshold is not met the example set for the next iteration is built by removing the positive and negative examples covered by the previously learned rule. Sequential Covering serves as a basis for many inductive algorithms. The crucial point of this algorithm is the function Learn-One-Rule. Clark and Niblet [7] provide a K-Beam Search based algorithm, which is by now one standard approach for learning one rule.

```
Sequential-Covering(Target_Attribute,Attributes,Examples,Threshold)
    Learned_Rules ← {}
    Rule ← Learn-One-Rule(Target_Attribute,Attributes,Examples)
    while Performance(Rule,Examples) > Threshold, do
        Learned_Rules ← Learned_Rules + Rule
        Examples ← Examples - {examples correctly classified by Rule}
        Rule ← Learn-One-Rule(Target_Attribute,Attributes,Examples)
    Learned_Rules ← sort Learned_Rules according to Performance over Examples
    return Learned_Rules
```

Fig. 6. Sequential covering algorithm. (Adapted from [24].)

```
FOIL(Target_Predicate, Predicates, Examples)
    Pos ← Examples for which Target_Predicate is true
    Neg ← Examples for which Target_Predicate is false
    Learned_Rules ← {}
    while Pos, do
        NewRule ← rule that predicts Target_Predicate with no preconditions
        NewRuleNeg ← Neg
            while NewRuleNeg, do
            Candidate_Literals ← generate new body literals for NewRule, based on Predicates
            Best_Literal ← argmax l ∈ Candidate_Literals FOIL_GAIN(l,NewRule)
            add Best_Literal to preconditions of NewRule
            NewRuleNeg ← subset of NewRuleNeg that satisfies NewRule preconditions
        Learned_Rules ← Learned_Rules + NewRule
        Pos ← Pos - {members of Pos covered by NewRules}
    return Learned_Rules
```

Fig. 7. The FOIL algorithm. (Adapted from [24].)

FOIL: Learning first order rules. Though the Sequential Covering algorithm builds the basis for many inductive rule learning algorithms it is in combination with CN2 a propositional rule learner. Nevertheless it builds a basis for many first order rule learning approaches. A widely used first order rule learning algorithm is the top-down procedure called FOIL [29]. Modified versions of FOIL are the basis for most adaptive IE systems using relational learning techniques. For example, the SRV system [10] uses a FOIL based core algorithm. FOIL tries to find a description of a target predicate, given a set of examples and some background knowledge. In general the background theory and examples are function-free ground facts, where the examples are positive and negative instances of the target predicate. Here negative instances means explicitly stating for which instantiations the target predicate shall not be true. FOIL uses the closed world assumption during rule learning: every instance not declared positive is assumed to be negative.

Using FOIL for wrapper induction. Assume we chose a document representation where a document D is mapped to a set of facts $T(D)$ of the form *word(Pos,Token)*,

where *Pos* is the starting position of the text fragment described by the token *Token*. *Token* is a feature term in the previously discussed sense. Further we have a set of dedicated predicates, which FOIL uses for rule construction. In general these predicates are of two different types: one to test for certain token features and the other to set tokens into relation to each other. For example, *next(P1, P2)* describes all tokens at position *P1* and its direct successor token referred by *P2*. The reader can think of many different relational predicates like: *fragment(P1, Length, F)* or *near_by(P1,P2)*. Examples for predicates checking certain features may be *has_feature(type, html, Token)*, which checks if a token *Token* has the attribute *type* and if its value is *html*. An additional background theory provides the definition for these predicates. At this point the advantage of relational and first order rule learning becomes apparent. Suppose our document representation is modified so that it contains relational information in the sense of a document object model (DOM). Then to learn rules taking advantage of additional document layout features, we need only extend the background theory with additional predicates for traversal and retrieval of DOM nodes. Let us return to the FOIL algorithm. What is left to demonstrate FOIL are examples. Assuming we want to learn a rule *extract(X,Y)*. Note this is a multi slot rule which itself defines a problem class in the IE context. Because examples have to contain the target predicate our examples will be of the form *extract("Tatonka Kimberley 50","EUR 174.95")*.

Now that we know how to define positive examples let us see how to represent negative ones. In comparison to some finite state techniques the need for negative examples may be a shortcoming. While FOIL needs negative examples, we do not necessarily have to provide them by tedious annotating them by hand. Some assumptions can be taken, such that negative examples can be generated automatically. For example we can take a closed world assumption: negative examples are all those text tuples not explicitly stated to be positive ones. Besides the huge amount of possible tuples that can be generated, this has another serious shortcoming: if a document contains more relevant fragments than annotated, we will run into problems if we rely on the closed world assumption. So either we must exhaustively enumerate all relevant fragments of the document, or else we must use heuristics to construct negative examples automatically. Examples of such heuristics include text fragments consisting of positive examples but extended to the left and right or permutation of positive example arguments. Freitag [10] uses a heuristic that constructs negative examples from all those fragments with more, or fewer, tokens than the positive examples.

Let us start FOIL's top-down learning with the most general rule *extract(X,Y)* $\leftarrow$ *true*. As long as the current rule covers one of the negative examples—for example *extract("Tatonka","EUR")* (see Figure 5)—the body of the rule is extended by the best constraining literal. Most applications of FOIL in the context of IE use a similar function to that of Quinlan's *FOIL_GAIN* which characterizes the information contained in the ratio of positive and negative examples in a set of examples. The SRV system uses following gain function: $I(S) = -log_2(P(S)/(P(S)+N(S)))$, where $P(S)$ and $N(S)$ are the positive and negative number of examples of the example set S. $GAIN(A) = P(S_A)(I(S) - I(S_A))$ with S_A is the subset of S covered by the rule after adding the literal A to it. The computation of the best literal is thus very expensive depending on the document sets and the complexity of the predicates. The following rules

illustrate some hypothetical intermediate steps during a possible rule induction process:
extract(X,Y) ← *fragment(P1,2,X), fragment(P2,2,Y).*
extract(X,Y) ← *fragment(P1,2,X), near_by(P1,P2,Y), fragment(P2,2,Y).*
extract(X,Y) ← *word(P0,Token1), has_feature(type,html,Token1),*
has_feature(color,blue,Token1), next(P0,P1), fragment(P1,2,X),
near_by(P1,P2,Y), fragment(P2,2,Y), in(Token2,Y),
has_feature(type,float,Token2).

3.2 Relational learning techniques in practice

In the following we will introduce some adaptive IE systems that use relational learning techniques or are very strongly connected to it.

One-shot learning. AutoSlog [30] learns a set of extraction patterns by specializing a set of general syntactic patterns. Such patterns are called concept nodes and consist of certain attributes like rule trigger, constraints, constant slots or enabling condition.

AutoSlog needs semantic pre-processing which involves a user to tag the example documents with certain semantic labels. The system expects a part of speech (POS) or linguistically tagged input document. The key idea is to figure out a *trigger* word and matching predefined linguistic patterns. The algorithm then tries to modify these patterns by several heuristics and the use of a taxonomy. The AutoSlog algorithm does not contain any inductive step, it is left to the user to accept or reject a learned rule. It is mentioned in the context of relational learning techniques, because concept nodes may be easily rewritten in the form of logic rules and thus specializing among those is similar to relational learning.

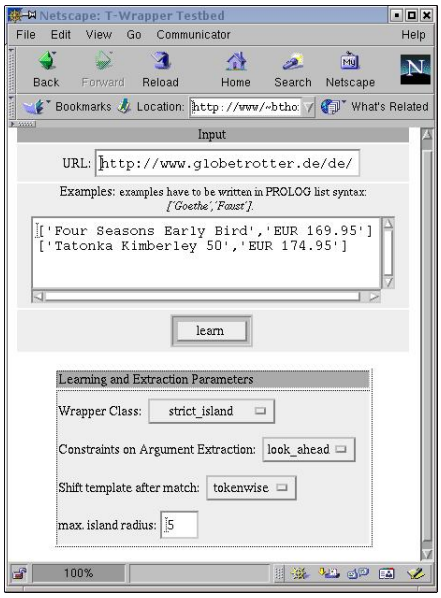

Fig. 8. The T-Wrapper web interface

Another one shot learning system by Thomas is T-Wrappers [34]. Similar to Kushmerick he defines several wrapper classes assuming that the essential part for wrapper construction is it to learn left and right delimiters, which he calls anchors. Thomas describes wrapper in a Prolog-like language [13]. His approach can be summarized into three steps: 1) for each argument of each example tuple collect the left and right text fragments wrt. to a given length 2) generalize on all left and on all right fragments, such that for a 3-tuple for example this generalization process results in 6 anchor patterns. 3) depending on the wrapper class

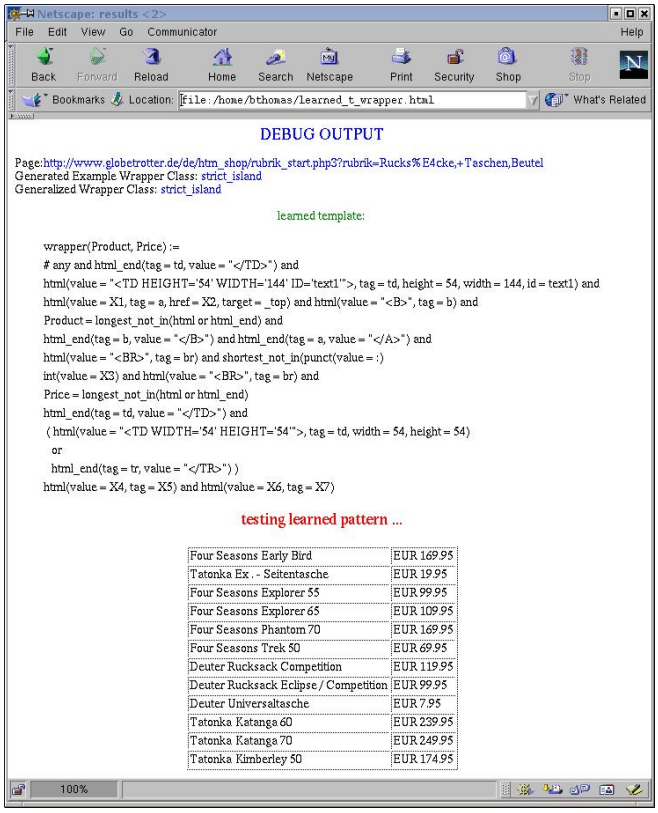

Fig. 9. Learned T-Wrapper rule and extractions.

[13] use a pre-defined structural rule layout where the learned anchors are inserted. Thomas uses a feature term representation for text fragments and anti-unification (LGG) based techniques for generalization. In contrast to Riloff he uses a bottom-up strategy, starting with the surrounding fragments (the most specific ones) of each argument of each example. The T-Wrapper system learns from positive examples only, does not use any linguistic knowledge (though this can be easily encoded into the feature term representation) and it learns multi slot extraction rules. Only a handful of examples is needed to produce wrappers for HTML documents without the need of post processing the learned wrapper. Figure 9 shows such a learned rule and extractions from the web page shown in Figure 5.

FOIL-based systems: Top-down learning. One of the most successful ILP and top-down based learning system used for IE is the SRV system by Freitag [10]. SRV strongly follows the idea of the standard FOIL algorithm as described in Section 3.1. The system is capable of learning single slot wrappers from natural and HTML texts.

It also follows the key idea in learning left and right delimiters. Freitag also extended SRV's feature predicates and document representation by linguistic information. Surprisingly this had little effect on the performance of SRV. Though Freitag uses a standard FOIL algorithm his representation of rules does not follow the strict concept of first order rules or standard Prolog rules. Junker et al [16] illustrates how to use a top-down sequential covering algorithm to learn Prolog rules. Junker et al focuses on single slot extraction and text documents. Their algorithm can be seen as a special ILP learner based on a set of operators used for rule refinement. Each of these operators can be understood as transformations or inference rules, replacing or introducing new or modified literals to the body of a rule.

Bottom-up learning. Bottom-up algorithms start with the most specific rules or example instances and stepwise generalize these rules until a certain stop criterion is reached (eg none of the negative examples are covered and most of the positive ones). Soderland presents the Crystal system [33], which uses a bottom-up sequential covering algorithm. Crystal and the later implemented system WebFoot (for HTML documents) use two different generalization operations: a unification-based operation for syntactic generalization, and a second based on knowledge drawn from a taxonomy to generalize on the slot fillers. For example, assume a semantic constraint allows as slot fillers instances of the class $< PERSON >$ and a word *he* is an instance of class $< GENERICPERSON >$ which is a subclass of $< PERSON >$. Then the semantic constraint for the word *he* is met because of the implication. Furthermore in Crystal whole sentences are linguistically pre-processed and annotated to serve as examples. The system is able to learn multi slot extraction rules given positive and negative examples.

Hybrid relational learning techniques. A serious problem using top-down learning algorithms for IE is to guide the search for good body literals. As long as the search space can be kept small, a pure non-biased top-down approach may be tractable for IE. But in general the great number of negative examples and the use of more sophisticated predicates (eg background theory) for rule refinement blows up the search space. Thus a standard top-down algorithm that exhaustively checks all possible rule refinements is infeasible. On the other hand using a standard bottom-up algorithm often leads to overly specific rules with very low recall.

The Rapier [5] system combines top-down and bottom-up techniques to learn single slot extraction rules from POS-tagged documents. In fact it uses a similar approach adapted from the ILP system CHILLIN to learn three patterns: pre-filler patterns (left delimiter), post-filler-pattern (right delimiter), and a patterns for the slot-filler itself. Like most of the other systems it uses a linguistic pre-processing step to annotate the document with part of speech information. For the generalization step Rapier uses a modified LGG operator, that provides disjunctions of two patterns to be generalized. Rapier begins by generalizing two filler patterns. This generalized pattern is used to initiate a top-down learning step. Elements to be added to the rule are created by generalizing the appropriate portions of the pre-fillers or post-fillers of the pair of rules from which the new rule is generalized. The LGG operator is also modified such that it uses

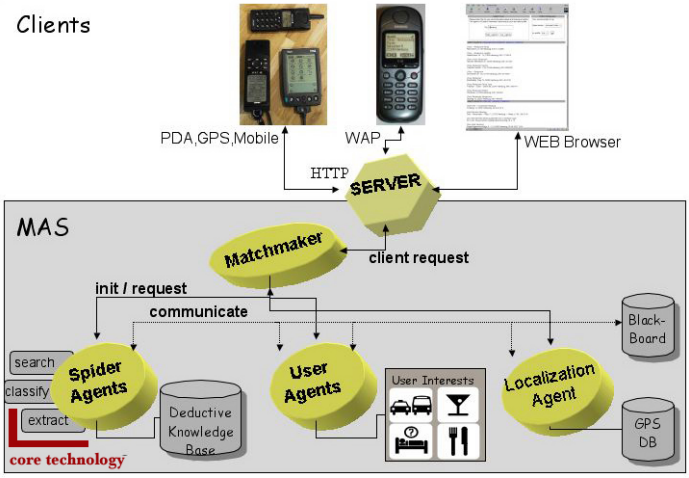

Fig. 10. Architecture of the MIA system.

an additional semantic hierarchy to derive super-classes covering instances in the initial constraints.

Another system combining several learning techniques is that of Ciravegna [6] called Pinocchio. He uses a sequential covering algorithm. The system needs as input a collection of texts pre-processed with a POS tagger. Pinocchio is unique in that it does not try to learn an extraction rule for a complete slot. Instead it learns rules to recognize the left delimiter independently from the right one. The algorithm can be separated into three steps: 1) A bottom-up based rule construction from tag surrounding text fragments, where the *k* best rules for each instance are retained in a best rule pool. 2) To raise the overall recall rate, some rules not contained in the best rule pool are considered for further refinement. The idea is to find a set of rules that are reliable at least in restricted areas. Ciravegna illustrates a method which he calls contextual tagging. 3) Finally, correction rules are learned. These rules shift wrongly positioned tags to their correct position. They are learned with the same algorithm like the tagging rules but also match the tags produced during the previous steps. Pinocchio can be used to extract multiple slots. Results presented by Ciravegna are very promising.

3.3 Application example

As shown in Figure 10, *MIA* [2] is a multi-agent based information system focusing on the retrieval of short and precise facts from the web. Making the immense amount of information on the web available for *ubiquitous computing* in daily life is a great challenge. Besides hardware issues for wireless ubiquitous computing, like wireless communication, blue-tooth technologies, wearable computing units, integration of GPS, PDA and telecommunication devices, one major problem is that of intelligent information extraction from the WWW.

Instead of overwhelming the mobile user with documents found on the web, the MIA system offers the user a short precise piece of information she is really interested in with fast query response times. *MIA* monitors the position of the mobile users and autonomously updates the subject of search whenever necessary. Changes may occur when the user travels to a different location, or when she changes her search interests. Currently MIA supports three different user types: stationary Web Browser, mobile phone with WAP support and PDA with GPS device. The search domains are freely configurable by the user. So far *MIA* is capable of automatic (multi slot) address extraction; extraction of time-tables and event descriptions is also planned.

Using pre-learned wrappers. One heuristic used in the MIA system is that to use a set of pre-selected web sites as entry points for the retrieval. This assures to the user that at least some results are presented. While some known information resources are used, the problem remains of extracting this information. The MIA administrator uses MIA's wrapper toolkit to learn wrappers for certain domains. Whenever the extraction agents visit one of these domains during their search they use these pre-learned wrappers to extract information from one of the web pages. Currently the wrapper toolkit uses a one shot learning strategy [34] extended with a special document representation. Instead of assuming the document to be a linear sequence of tokens the DOM model of the document is used. The general strategy to learn left and right anchors (delimiters) is kept, but extended to path learning in the DOM of the document. Additionally the general intention is now to learn one wrapper for a whole document class (eg found at one web domain) instead learning one wrapper for one document. This is in contrast to [34] and the following method used by MIA.

Learning wrappers during search and retrieval. The major problem someone is confronted with in the context of an autonomous multiagent system like *MIA* is the lack of available examples for learning wrappers online. Because *MIA*'s web page classifier provides unknown web pages to the system, we can not assume the existence of the training data needed for learning. On the one hand, the classifier is good enough to determine if an address is contained on a web page; on the other hand, arbitrary web pages vary too much for a single general purpose address wrapper to be effective. While the use of a large address database and a name recognizer (POS-tagger or named-entity recognizer) might work better for this particular problem, but MIA is aiming at a generic approach with no hard-wired solutions.

To overcome this problem MIA uses a learning algorithm that derives its examples by means of *knowledge representation* (*KR*) techniques. That is, we model our knowledge about addresses with a logic KR language in advance and are able to query this knowledge base to derive example patterns. These example patterns can then be used to construct examples as input to a modified learner. This allows MIA to learn wrappers even for unknown pages. This approach called *learning from meta examples* shows very promising results for the automatic construction of address wrappers (Figure 11). Various experiments showed that the knowledge base can also be used to verify the extracted information on a still limited level, but nevertheless this idea can serve as basis for some kind of self-supervision for autonomous information extraction agents.

Test Setting (meta examples) pages: 461 tuples: 3158 KB: 5 rules, 6 patterns						
wrapper class	pages cov.	pos	neg	recall	precision	coverage
semi, no html	319	785	85	0,69	0,9	0,27
weak, no html	382	1712	904	0,83	0,65	0,54
semi, look ahead	339	895	156	0,73	0,85	0,28

self-supervision:
semi: extractions must be matched by derived pattern from KB
weak: extractions must be matched by generalized patterns from KB
wrapper wlass
no html: extraction are not allowed to contain HTML
look ahead: extractions are not allowed to contain tokens of right delimiter

Fig. 11. Performance of MIA's IE component.

4 Summary

Information extraction is a core enabling technology for a wide variety of information-gathering and -management agents. The central challenge to information extraction is the ability to scale with the number and variety of information sources. We have described a variety of adaptive information extraction approaches that use machine learning techniques to automatically learn extraction rules or knowledge from training data.

Due to the highly practical nature of the IE task, all of the approaches described in this chapter have been tested on various real world examples. That means the gap between pure research and practical usage in agent systems is smaller than it might seem at first glance. For example, the IE component of the MIA multi-agent system described in Section 3.3 is based directly on techniques described in this chapter.

We have segmented the field of adaptive information extraction roughly into two areas: *finite state* techniques that learn extraction knowledge corresponding to regular grammars or automata, and the *relational rule learning* techniques that learn first-order Prolog-like extraction rules. In addition to the core adaptive IE techniques, we also briefly discussed several issues related to the entire information extraction "lifecycle".

The finite-state approaches are generally simpler, and their learning algorithms are generally faster and require fewer training examples. On the other hand, relational representations are substantially more expressive, which can be crucial for natural language domains such as newspaper articles that exhibit substantial variability. Adaptive information extraction thus exhibits a familiar expressiveness–complexity tradeoff.

Perhaps the most fundamental open issue in adaptive information extraction is a method for determining which technique is best suited to any particular extraction task. Today, this complicated judgment requires considerable expertise and experimentation. Ultimately, we foresee a semi-automated methodology in which the heterogeneity of the documents could be measured in various dimensions, in order to predict the simplest approach that will deliver satisfactory performance.

Acknowledgments. Kushmerick was supported by grant N00014-00-1-0021 from the US Office of Naval Research, and grant SFI/01/F.1/C015 from Science Foundation Ireland. Thomas and the MIA project was supported by grant from the state of Rhineland-Palatinate, Germany.

References

1. F. Bergadano and D. Gunetti. *Inductive Logic Programming*. MIT Press, 1996.
2. G. Beuster, B. Thomas, and C. Wolff. MIA - A Ubiquitous Multi-Agent Web Information System. In *Proceedings of International ICSC Symposium on Multi-Agents and Mobile Agents in Virtual Organizations and E-Commerce (MAMA'2000)*, December 2000.
3. D. Bikel, S. Miller, R. Schwartz, and R. Weischedel. Nymble: A high-performqance learning name-finder. In *Proc. Conf. on Applied Natural Language Processing*, 1997.
4. S. Brin. Extracting patterns and relations from the World Wide Web. In *Proc. SIGMOD Workshop on Databases and the Web*, 1998.
5. M. E. Califf. *Relational Learning Techniques for Natural Language Information Extraction*. PhD thesis, University of Texas at Austin, August 1998.
6. F. Ciravegna. Learning to Tag for Information Extraction from Text. In *Workshop Machine Learning for Information Extraction, European Conference on Artifical Intelligence ECCAI*, August 2000. Berlin, Germany.
7. P. Clark and T. Niblett. The CN2 induction algorithm. *Machine Learning*, 3:261–283, 1989.
8. W. Cohen and L. Jensen. A structured wrapper induction system for extracting information from semi-structured documents.
9. V. Crescenzi, G. Mecca, and P. Merialdo. Roadrunner: Towards automatic data extraction from large web sites. In *The VLDB Journal*, pages 109–118, 2001.
10. D. Freitag. *Machine Learning for Information Extraction in Informal Domains*. PhD thesis, Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, November 1998.
11. D. Freitag and N. Kushmerick. Boosted Wrapper Induction. In *Proceedings of the Seventh National Conference on Artificial*, pages 577–583, July 30 - August 3 2000. Austin, Texas.
12. D. Freitag and A. McCallum. Information Extraction with HMM structures learned by stochastic optimization. In *Proceedings of the Seventh National Conference on Artificial*, July 30 - August 3 2000. Austin, Texas.
13. G. Grieser, K. P. Jantke, S. Lange, and B. Thomas. A Unifying Approach to HTML Wrapper Representation and Learning. In *Proceedings of the Third International Conference on Discovery Science*, December 2000. Kyoto, Japan.
14. C. Hsu and M. Dung. Generating finite-state transducers for semistructured data extraction from the web. *J. Information Systems*, 23(8):521–538, 1998.
15. L. Jensen and W. Cohen. Grouping extracted fields. In *Proc. IJCAI-01 Workshop on Adaptive Text Extraction and Mining*, 2001.
16. M. Junker, M. Sintek, and M. Rinck. Learning for Text Categorization and Information Extraction with ILP. In *Proc. Workshop on Learning Language in Logic*, June 1999. Bled, Slovenia.
17. N. Kushmerick. *Wrapper Induction for Information Extraction*. PhD thesis, University of Washington, 1997.
18. N. Kushmerick. Regression testing for wrapper maintenance. In *Proc. National Conference on Artificial Intelligence*, pages 74–79, 1999.
19. N. Kushmerick. Wrapper induction: Efficiency and expressiveness. *Artificial Intelligence*, 118(1–2):15–68, 2000.
20. N. Kushmerick. Wrapper verification. *World Wide Web Journal*, 3(2):79–94, 2000.
21. N. Kushmerick, D. S. Weld, and R. Doorenbos. Wrapper Induction for Information Extraction. In M. E. Pollack, editor, *Fifteenth International Joint Conference on Artificial Intelligence*, volume 1, pages 729–735, August 1997. Japan.
22. T. Leek. Information extraction using hidden Markov models. Master's thesis, University of California, San Diego, 1997.

23. K. Lerman and S. Minton. Learning the common structure of data. In *Proc. National Conference on Artificial Intelligence*, 2000.
24. T. M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.
25. S. Muggleton and L. D. Raedt. Inductive logic programming: Theory and methods. *Journal of Logic Programming*, 19(20):629–679, 1994.
26. I. Muslea. Extraction patterns for information extraction tasks: A survey. In *Proc. AAAI-99 Workshop on Machine Learning for Information Extraction*, 1999.
27. I. Muslea, S. Minton, and C. Knoblock. A hierarchical approach to wrapper induction. In *Proc. Third International Conference on Autonomous Agents*, pages 190–197, 1999.
28. I. Muslea, S. Minton, and C. Knoblock. Selective sampling with redundant views. In *Proc. National Conference on Artificial Intelligence*, 2000.
29. J. R. Quinlan. Learning logical definitions from relations. *Machine Learning*, 5:239–266, 1990.
30. E. M. Riloff. *Information Extraction as a Basis for Portable Text Classification Systems*. PhD thesis, University of Massachusetts Amherst, 1994.
31. K. Seymore, A. McCallum, and R. Rosenfeld. Learning hidden Markov model structure for information extraction. In *Proc. AAAI-99 Workshop on Machine Learning for Information Extraction*, 1999.
32. S. Soderland. Learning information extraction rules for semi-structured and free text. *Machine Learning*, 34(1-3):233–272, 1999.
33. S. G. Soderland. *Learning Text Analysis Rules for Domain-Specific Natural Language Processing*. PhD thesis, University of Massachusetts Amherst, 1997.
34. B. Thomas. Anti-Unification Based Learning of T-Wrappers for Information Extraction. In *Proc. AAAI-99 Workshop on Machine Learning for Information Extraction*, 1999.
35. B. Thomas. Token-Templates and Logic Programs for Intelligent Web Search. *Intelligent Information Systems*, 14(2/3):241–261, March-June 2000. Special Issue: Methodologies for Intelligent Information Systems.