

Documentum ECI Self-Repairing Wrappers: Performance Analysis

Boris Chidlovskii
Xerox Research Centre
France
chidlovskii@xrce.xerox.com

Bruno Roustant
EMC Documentum
France
Roustant_Bruno@emc.com

Marc Brette
EMC Documentum
France
Brette_Marc@emc.com

ABSTRACT

Documentum Enterprise Content Integration (ECI) services is a content integration middleware that provides one-query access to the Intranet and Internet content resources. The ECI Adapter technology offers an interface to any application for data and metadata extraction from unstructured Web pages. It offers a unique framework of wrapper production, automatic recovery and maintenance, developed at Xerox Research Centre Europe and based on state-of-art algorithms from machine learning and grammatical inference. In this presentation we analyze the performance of ECI adapters deployed in current commercial installations. We benefit from accessing reports on daily tests for all ECI commercially deployed adapters collected from June 2003 to September 2005. Using the daily reports, we analyze different aspects of the wrapper technology.

Categories and Subject Descriptors

H.2.5 [Heterogeneous Databases]; I.2.6 [Learning]: Induction; D.2.8 [Software Engineering]: Metrics—*performance measures*

General Terms

Algorithms, Performance, Measurement

Keywords

Content integration, Web wrappers, wrapper maintenance

1. INTRODUCTION

Over the last decade, wrapping Web sources has been a hot research topic in the data extraction, mediation and integration domains. Nowadays, despite the progress in XML publishing, service-oriented architectures and Web services, wrapping Web sites that publish human-oriented HTML pages, remains an important component of Semantic Web efforts [16].

The diversity of different wrapping techniques proposed in data mediation and machine learning communities is remarkable [2, 7, 15, 17, 24, 25, 26, 28]. Several efforts to survey and classify existing approaches and techniques have been undertaken in [1, 6,

13]. However, the major open issue remains the comparative evaluation of different wrapping methods. Similarly to the situation in information extraction [19], comparing different methods remains a challenge for the reasons of lack of agreed evaluation testbed and multiple inconsistencies in the experimental procedures.

Furthermore, requirements to wrapper technologies have essentially evolved over the last decade. This evolution went through three major steps. Initially, the wrapper techniques were competing in the *expressiveness* (i.e., a capacity to accurately wrap any given Web site) and *adaptability* (i.e., the automatic generation without coding). Later, the need for wrapper maintenance imposed additional requirements, such as *robustness* and *automatic repairing* of wrappers. Finally, the commercial deployment of wrapper technologies add requirements of *scalability*, *optimal quality control*.

In the absence of an agreed evaluation testbed, in this paper we explore a different dimension. We report on the commercial deployment of one specific wrapper product, namely, the ECI (former askOnce) wrapper technology developed at Xerox Research Centre Europe in 1999-2002.

The first commercial deployment of askOnce technology dated 2000, with Web wrappers being rule-based javaCC parsers. To address the wrapper production requirements, askOnce version 2 integrated the wrapper generation component in 2001. It included the Graphical Wrapper Toolkit, where any user with no programming skills could induce an accurate Web wrapper from few example pages [3]. In January 2004, askOnce committed version 4, where the wrapper architecture was extended with the maintenance component [5]. In March 2004, the askOnce content integrator has been acquired by Documentum to become a part of ECI Services [11]. Currently, ECI wrappers are deployed in more than 100 full Documentum ECI installations in Europe and the United States serving 100,000 to 150,000 users.

Among Web content providers solicited by ECI clients, some do cooperate with content integrators by offering an access to data through the Web services, XForm querying, XML publishing with a predefined schema or RSS feeding. However, a vast majority of domain-specific Web providers do not cooperate with integrators; they target the human visitors only and support the conventional mechanisms of PHP/CGI querying and human-oriented HTML publishing.

The main goal of our study is to analyze the commercial performance of ECI Web wrappers over a period of 28 months, from June 2003 to September 2005, including the migration from version 3 to version 4. We benefit from a unique opportunity of accessing reports on wrapper tests run daily during the whole period of the commercial deployment. The daily reports account for a total of 1,200,000 tests for about 500 wrappers. To our best knowledge, this is the first attempt of conducting such a study.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

SIGMOD 2006, June 27–29, 2006, Chicago, Illinois, USA.
Copyright 2006 ACM 1-59593-256-9/06/0006 ...\$5.00.

Since wrappers are deployed in a large-scale environment and communicate with non-cooperative sites, ECI-4 pays a particular attention to the wrapper maintenance. Once a wrapper production platform became operational, the creation of wrapper instances for a majority of Web providers represents an important but one-time investment. Instead, in the content integrator life cycle, the wrapper maintenance cost is considerably superior to the initial investment in the wrapper creation. The goal of the wrapper maintenance component consists in reducing the maintenance cost but still guaranteeing the top quality of extracted data.

The second issue of our study is how wrapper technologies get accommodated to the commercial deployment context. The human factor and the commercial deployment policy reveal specific requirements of scalability, an optimal quality control, etc. We discuss them and their impact on the ECI wrapper technology.

The remainder of paper is organized as follows. Section 2 introduces ECI Adapter technology. Sections 3 and 4 revise the ECI wrapper architecture and its components, including learning algorithms for training transducers and content views, and the recovery algorithm. In Section 5, we explain the ECI wrapper administration and deployment principles. Section 6 is central in the paper, it reports the detailed performance analysis of commercially-deployed wrappers. Section 7 gives a short survey of wrapper techniques and Section 8 concludes the paper.

2. ECI ADAPTER MECHANISM

Documentum Enterprise Content Integration (ECI) services [11] is a content integration middleware that provides one-query access to the Intranet and Internet content resources. ECI services enhance enterprise's solutions in place without imposing any change and respect native security policies. The ECI functionalities include search, synthesis, sharing and integrating information, dynamic linguistic clustering, personalized ranking, cross-lingual search, scheduled queries and notification [11].

The ECI Adapter (aka askOnce) technology offers an interface to any content resource, including the mechanism of query mapping capabilities, data and metadata extraction from semistructured Web pages. It represents a unique framework for rapid adapter production and maintenance.

The current adapter library [10] covers content providers (Factiva, Lexis Nexis), Web providers¹, full-text engines (Verity, MS Index Server, Google Search Appliance, etc.), enterprise repositories for Documentum products, Lotus Notes, Lotus Domino, MS SiteServer, MS Exchange, Oracle, JDBC/ODBC, etc. It also packs available adapters in domain-specific packages and "bundles". Currently, bundles of upto 30 adapters are available for pharmaceutical industries and science, an smaller packages for aerospace, computer, legislation, health and regulation domains.

An adapter for a Web provider includes a query mapper and one or more wrappers. The query mapper plays the major role in supporting the ECI uniform query interface, as it ensures the optimal translation and rewriting of users' queries into providers' query languages, including the compensation of missing search facilities. Each adapter includes one wrapper for each class of pages returned upon queries.

3. WRAPPER TECHNOLOGY

A wrapper is a set of extraction rules that instruct an HTML parser how to extract and label content of a page. These extrac-

tion rules are often specific to a content provider and are tightly linked to the mark-up and structure of provider HTML pages.

Wrapping Web providers is a special case of the information extraction (IE) problem. Web pages processed by wrappers are more structured than pages processed by conventional IE methods that target unstructured text like FAQ or seminar announces [19]. Instead, wrappers often can achieve a high accuracy of information extraction, with both precision and recall being as least 98%, whereas such a high accuracy is often unreachable with IE from plain text. The high accuracy is critical in data integration and Semantic Web solutions since the extracted data is often entailed with sophisticated post-processing operations, like an answer rating or a database-like join operation, which are very sensitive to inaccurate data. In the following, *Web wrapping* will refer to a *highly accurate information extraction from HTML pages*.

A majority of wrapper techniques assume that no changes can happen to the pages; as consequence, wrappers often become brittle when, for some reasons, the page mark-up or structure is changed. If a wrapper is broken few time per year, in solutions with hundreds of operating wrappers, designers face a burden of repairing multiple wrappers per day. The wrapper maintenance is challenging in cases when provider pages undergo massive and sweeping modifications, due to a complete redesign. Fortunately, a much more frequent case is that of *concept shift*, when provider's pages undergo *local and small changes*. It is therefore important to develop methods to generate wrappers with integrated maintenance components capable to recover from small changes.

The *automatic wrapper maintenance* ensures the robustness of a wrapper under the assumption of small changes. When a provider changes the page format, the wrapper runs in an error and triggers the recovery procedure which attempts to resume the information extraction and repair the wrapper. It is common to distinguish between two sequential maintenance steps, *extraction recovery* and *wrapper repairing* [25]. The extraction recovery resumes the labeling of tokens in a page; it is aimed at extracting as much relevant data as possible. The wrapper repairing becomes possible only if the recovery went successfully; pages in the new format can be automatically re-labeled and extraction rules can be re-learned to match the new page format.

The wrapper maintenance in ECI-4 is based on the ensemble principle from machine learning [9]. An *ensemble of classifiers* is a set of classifiers whose individual decisions are combined to classify new examples. Ensembles of classifiers (views) are more accurate than individual classifiers, provided that the individual classifiers are *independent* (their errors are uncorrelated) and *accurate* (error rates are better than a random guess) [9]. While the accuracy of a classifier depends on chosen learning methods, the independence of classifiers is relevant to features selected for the learning. The independence of classifiers can be ensured if different and uncorrelated features are selected to build the classifiers.

For the view independence, ECI considers two disjoint sets of *context* and *content* features. A context feature of a token in HTML characterizes its surroundings, that is, tags and tokens that precede (prefix) or follow the token (suffix). Instead, content features characterize the string itself; the token length or number of words are examples of content features.

The content and context features are used to build independent classifiers. Also, ECI uses alternative classifiers to recover wrappers that run in error. In the case of concept shift, changes in page format often hurt some of the classifiers, but not all of them. Therefore, it is often possible to use valid classifiers to identify reliable components in input data and eventually reuse them to automatically re-train the wrapper.

¹See <http://www.xrce.xerox.com/people/chidlovskii/home.html> for the full list of ECI-4 wrappers.

3.1 ECI wrapper architecture

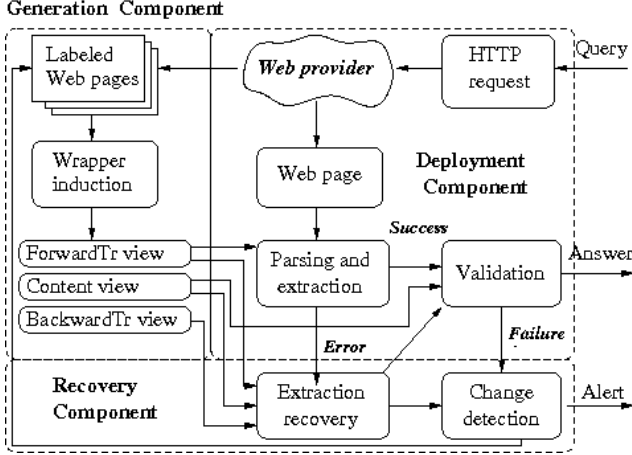


Figure 1: ECI wrapper architecture.

Figure 1 shows the ECI wrapper architecture with three major components, Generation, Deployment and Recovery. A wrapper life cycle begins in the Generation component, with labeling sample HTML pages fetched from a given Web provider. Then, an induction method generalizes the labeled pages into a wrapper instance which is used in Deployment component to process any new provider’s page. All data extracted by the wrapper gets validated. If the provider changes format of HTML pages, the wrapper can fail to complete the data extraction. In such a case, it calls for an extraction recovery procedure which tries to extract as many data as possible from the page. A successful recovery results in automatic re-labeling of new pages which can be used to generate a new wrapper version that accommodates the new page format.

4. WRAPPER INDUCTION AND RECOVERY

The wrapper induction in ECI is a special case of *sequential learning* [8], where an input sequence of tokens is labeled with a sequence of labels from a set L . ECI develops alternative and redundant views of pages; these views are useful for the recovery in the case of concept shift. Three alternative views on the sequential learning, given by 1) a *forward transducer* which parses the input file from the beginning to the end, 2) a *backward transducer*, parsing the file in the opposite direction, and 3) a *content classifier*. The three views are described below, they are followed by the recovery algorithm.

4.1 Forward transducers

The basic ECI wrapper component has a form of *string transduction*, where input sequences of tokens from set T are transduced in output sequences over alphabet L . A *regular transducer* $Tr : \mathbf{T} \rightarrow \mathbf{L}$, $\mathbf{T} = (t_1, \dots, t_k)$, $\mathbf{L} = (l_1, \dots, l_k)$ is a finite-state automaton when any transition consumes one input token $t_i \in T$ and emits output symbols $l_i \in L$. Input strings accepted by a regular transducer form a regular set Dom .

When transducers apply for wrapping HTML pages, input is tokenized into a sequence of *textual* and *tag tokens*. Textual tokens compose the page content and are denoted as $t \in T$; tag tokens like $\langle tr \rangle$ or $\langle hr \rangle$, control the content visualization. Wrappers usually target the extraction of textual tokens and components of some tag

tokens, like href attribute of $\langle a \rangle$ tags. In labeled samples, only these tokens may have semantic labels, while all non-extracted tokens are assumed to be labeled with a special none label, $none \in L$. Labeling a token t with class $l \in L$ is denoted $l(t)$ and the result of transduction of input sequence $\mathbf{T} \in Dom$ is denoted $\mathbf{L} = Tr(\mathbf{T})$.

In regular transducers, consuming an input token does not necessarily lead to emitting an output symbol, because of facing multiple emission choices in transducer states; in such cases the emission is postponed till reaching a state where the ambiguity is resolved.

ECI transducers extends the transducer induction in [29] and represents wrappers as sets of extraction rules [4]. The *Optimal Context Extraction Rules* (OCER) transducer is an incremental version of the regular transducer adopted to the information extraction task. In the following, we distinguish between labeled and unlabeled HTML fragments. Class of unlabeled fragments is denoted S^u , $S^u = \{v | \exists x \in T^*, xv \in Dom\}$. Class of labeled HTML fragments is a sequence of tokens with their labels, $S^l = \{(x, Tr(x)) | \exists u, v \in T^*, u.x.v \in Dom\}$.

An OCER transducer Tr is a triple (K, L, R) , where K is an input tokenizer, L is the semantic label set and R is a set of extraction rules $R = \{r_i\}$, where each rule r_i is a triple (p, s, l) , where $p \in S^l$ and $s \in S^u$ are prefix and suffix, and $l \in L$.

An OCER transducer Tr parses a page F from the beginning to the end and applies the extraction rules in R as follows. For a current textual token t , the labeled prefix \mathcal{P} of t contains all tokens from the beginning to t , with all previous textual tokens labeled, and \mathcal{S} is suffix of t , $F = \mathcal{P}t\mathcal{S}$, $\mathcal{P} \in S^l$, $\mathcal{S} \in S^u$. Pair $(\mathcal{P}, \mathcal{S})$ forms the *full context* of token t . Wrapper compares then \mathcal{P} and \mathcal{S} to prefixes and suffixes in extraction rules. Pair $(\mathcal{P}, \mathcal{S})$ *matches* a prefix-suffix pair (p, s) of a rule $r = (p, s, l)$, if p is a suffix of \mathcal{P} , $\mathcal{P} = up$, and s is a prefix of \mathcal{S} , $\mathcal{S} = sv$, for some labeled u and unlabeled v . In the match is found, token t is labeled with label l in the rule. If no exact rule is found for \mathcal{P} and \mathcal{S} , the wrapper runs in an *error*.

A prefix-suffix pair in an extraction rule $r \in R$ forms its *context*. A method for detecting optimal and minimal prefix-suffix pairs for extraction rules was developed in [4]. It finds out all ambiguities in sample data and detects minimal delays between an input token consumption and its label emission. In addition, the OCER is incremental; it aligns the token consumption with the labeling. It replaces emission delays with corresponding lookahead in the input data; these lookaheads are given by suffixes in rules. Finally, OCER method disregards variations in input that are irrelevant to the result information extraction. For the majority of Web providers, the input data does fit the class of regular languages, thus allowing to infer regular transducers, and therefore OCER wrappers, from positive examples.

Finally, from the ensemble point of view, any transducer as a *partial classifier*, as it runs in an error when no exact rule is found.

4.2 Backward transducers

Backward transducers represent an alternative view on HTML pages processed by a wrapper. A backward transducer scans a file backward; its extraction rules use optimal and minimal set of *labeled suffices* and *unlabeled prefixes* to uniquely label textual tokens. Like forward transducers, a backward transducer is partial and can run in error when the format changes. However, it would fail at positions different from those where the forward transducer would. Therefore, backward extraction rules can help complete information extraction in positions where the forward wrapper fails. This leads us to the idea of combining the forward and backward transducers during the extraction recovery.

The joint use of forward and backward wrappers can bring to a multi-pass scan of a file, when the direction of the scan can change one or more times. In the following, direct and backward wrappers are denoted as $Tr^{forward}$ and $Tr^{backward}$.

4.3 Content classifier

To classify textual tokens by content features only, ECI extracts a set F_C of $k=54$ content features, including 42 syntactic and 12 semantic ones. *Syntactic features* are the token length, word counts, density of digits, and standard delimiters (comma, semicolon, dot, etc.). *Semantic features* count typed components of textual tokens, such as proper names, url and time strings and noun phrases.

Content classifier C is generated from the content feature set F_C of textual tokens in sample pages. Any of existing techniques for classifier generation can be used here; ECI uses decision trees available with Weka package [31]. For textual token t , classifier C returns a pair $C(t) = (l_c, pr)$ where l_c is the most probable label for t , $l_c \in L$ and pr is its probability. Similarly, $C(t, l)$ returns the probability of labeling token t with l .

The content classifier C validates information extracted by the forward transducer in the basic page parse. The recovery algorithm starts by scanning page F from the place from the beginning to the end. It probes the transducer $Tr^{forward}$ with a current token t ; if Tr finds a matching rule with label l_{Tr} , t is labeled with l_{Tr} if C validates l_{Tr} by observing content features of t , for some *validation threshold* value, $C(t, l_{Tr}) \geq thVal$.

Algorithm 1. Basic page scan with forward transducer and content classifier.

```

for each token  $t$  in file  $F$  do
   $l_w := Tr^{forward}(t)$ 
  if  $l_w \in L$  and  $C(t, l_w) \geq thVal$  then
    label  $t$  with  $l_w$ 
  else return 'error'
return 'success'

```

4.4 Multi-scan recovery

The recovery algorithm starts by scanning page F from the place where the forward transducer runs in error. It keeps probing the transducer $Tr^{forward}$ with a current token t ; if Tr finds a matching rule with label l_w , t is labeled with l_w if C validates l_w by observing content features of t , for the threshold validation value, $C(t, l_w) \geq thVal$. If Tr can not find a matching rule, C provides the most probable label l_c for t . If the confidence of l_c is superior to a *recovery threshold* value $thRec$, t is labeled with l_c , otherwise t remains unlabeled.

As one can see, the content classifier C plays the double role in the recovery algorithm, which is reflected by deploying two threshold parameters. The validation threshold $thVal$ confirms the label choice by the transducer, it is therefore lower than recovery threshold $thRec$ in cases when the transducer runs in error and labeling decision is made only by the content view, $thVal < thRec$.

The ECI recovery algorithm is multi-scan, it combines the forward and backward transducers with the content classifier and tries to take advantages of each view in the most appropriate manner [5]. The algorithm switches the scan direction and tries to label not yet labeled textual tokens probing their context with forward and backward wrappers and content classifiers. Algorithm stops when none of the tokens gets labeled during the last scan. It returns status *success* if all tokens are labeled and *error* otherwise.

The recovery can trigger the automatic wrapper repairing if the recovery went well and all tokens have been labeled with selected threshold values. It can then automatically re-label sample pages and use them as input to automatic re-train the wrapper.

Algorithm 2. Multi-scan recovery with forward, backward and content views.

```

 $status := 'error'; Recovery := true; direction := 'bkwd'$ 
while  $Recovery$  do
   $Recovery := false; status := 'success'$ 
  for each unlabeled token  $t$  in  $F$  do
     $l_w := Tr^{direction}(t); (l_c, pr) = C(t)$ 
    if  $l_w \in L$  and  $C(t, l_w) \geq thVal$  then
      label  $t$  with  $l_w; Recovery := true$ 
    elif  $pr \geq thRec$  then
      label  $t$  with  $l_c; Recovery := true$ 
    else skip  $t; status := 'error'$ 
  reverse  $direction$ 
return  $status$ 

```

5. DEPLOYMENT AND TESTING

The ECI adapters reside in central and client repositories. Client adapters operate independently from the central storage, they can evolve through the automatic maintenance and repairing procedures. In the case of visible problems with adapters, the client downloads the correct and up-to-date adapter instances from the central repository. Optionally, new versions can be uploaded automatically.

The central repository is assumed to maintain fully operational and correct adapters. To maintain all wrappers and to promptly detect possible concept shifts, all working wrapper instances in the central repository are tested once a day, at midnight (GMT+1). Each wrapper instance is tested with a series of associated queries. Each query can return multiple pages for which it deploys a number of associated wrappers². There are three possible status given to a test:

Success: the wrapper completes the data extraction from answer pages and extracted data satisfies validation criteria. The success status can be achieved either by the basic parse or through applying the recovery procedure.

Error: the wrapper fails to complete the extraction process, even after the recovery.

Failure: the wrapper successfully completes the extraction process, but extracted data does not meet post-processing constraints set by the designer.

When the basic scan by $Tr^{forward}$ runs in error, it triggers the recovery procedure. If the recovery procedure manages to extract 98% of data, the query is considered as success. Otherwise, the query gets the status of error of failure that implies the same status for the whole adapter.

Constraints that cause the failure status are set by the designer upon the wrapper output. The most frequent case is a condition on the number of answers to be returned. If the provider returns less answers than expected³, this triggers the failure status.

²For multi-page answers, adapters may have one wrapper for the first page and other wrapper for the following pages.

³For example, if Google wrapper returns 'no match' answer for the 'computer' query.

The failure status can be caused by any communication dysfunction between the ECI platform and a multitude of Web providers. We distinguish between internal and external causes. An internal cause comes from other adapter components, like a query translation, for example, due to a change in the provider’s query language. External causes of the failure status include unavailable service or long delays at provider’s side, and network traffic. All these causes are considered as external to the wrapper maintenance component.

The maintenance charges are distributed among members of the ECI team, with the same person being responsible for both the wrapper creation and its further maintenance. In the case of error and failure, the wrapper designer receives an alert and test details. The designer may take no immediate action if she considers the wrapper as operational (false alert). If the situation is considered as a concept shift, she makes the wrapper evolve.

The *wrapper evolution* can be done in one of two modes. In the *evolutionary mode*, the wrapper training set is extended with new pages (where the errors were reported), without removing the previous pages. In the *breakout mode*, the wrapper is created from scratch. The decision on the evolution mode is left to the wrapper designer. The evolutionary mode requires less time and often represents a sufficient remedy for small format changes, as an important number of extraction rules may remain valid. Often, a high recovery performance is a clear indication for a minor concept change. Instead, a low recovery performance indicates an important change and is often followed by inferring the wrapper from scratch, using the new pages only.

The wrapper algorithms presented in Section 4 address the core requirements to Web wrappers, namely, their expressiveness, adaptability and automatic maintenance. They are rather sufficient for research prototypes and modest wrapper collections. However, in the context of the large-scale commercial deployment, the ECI team found it important to impose several additional requirements. They are the following:

Scalability: with a growing number of wrappers to maintain, the set of parameters controlling the wrapper performance should not depend on the wrapper number.

Easy quality control: the technology should allow an easy adaptation and a smart support in the wrapper quality control.

To meet these deployment requirements, ECI-4 wrappers underwent certain adjustments. The target accuracy level was set to 98% for all providers. The tokenization and feature selection remain specific to each wrapper, in order to ensure its optimal adaptation to the providers’ formats. The major adjustment concerns the two threshold parameters *thVal* and *thRec* controlling the basic scan and the recovery procedure. Since controlling individual threshold values represents a high burden for the support team, the thresholds have been set the same values for all wrappers in the repository. For the administrator, these thresholds offer a powerful and simple mechanism to globally control the wrapper performance and maintenance, in the function of quality criteria.

6. PERFORMANCE ANALYSIS

In this section, we analyze the wrapper and recovery performance using the daily wrapper test reports. For ECI-3, we dispose 418 daily reports covering the period from June 1st, 2003 to October 31st, 2004. They account for a total of 611,987 test queries, with an average of 1464.1 tests per day and 11.6 tests per wrapper.

For ECI-4, 510 daily reports cover the period from February 1st, 2004 to September 31st, 2005. In total, they account for 423,953

queries, with an average of 846.71 tests per day and 6.09 tests per wrapper.

ECI-3 reports included only the necessary information (global query status, execution time and wrapper version). During the migration from version 3 to version 4, the daily reports have been extended with detailed statistics on the wrapper performance for each page processed by the wrapper. In total, tests covered 1,262,735 pages with a minimum 1 page and a maximum 107 pages per query. Additionally, each entry reports whether the recovery was called, and if so, the accuracy of information extraction before and after the recovery. Finally, we could access the internal representation of wrapper components (in their latest versions). This allowed us to analyze the transducers and the content classifiers.

6.1 Wrappers and new versions

We start with a global view on the wrapper evolution for both versions. The major challenge of moving from ECI-3 to ECI-4 was the maintenance effort reduction. Indeed, the permanent changes and the brittleness of wrappers in ECI-3 under the concept shifts led to extending the basic wrapper architecture with the recovery component in ECI-4.

The main maintenance cost can be expressed in the number of new wrapper versions. Figure 2 presents a global view on the wrapper maintenance effort for both versions 3 and 4. The figure reports the total number of wrappers growing over months, and the number of new versions for existing wrappers committed monthly. Both absolute and normalized numbers (per 100 wrappers) of new versions are reported. The transition period of migrating ECI-3 wrappers to version 4 lasted 9 months, from April 2004 till October 2004 when the maintenance of ECI-3 wrappers was discontinued. As the figure suggests, the migration allowed to *reduce the number of new versions from 21.2 to 7.7* (per month and per 100 wrappers).

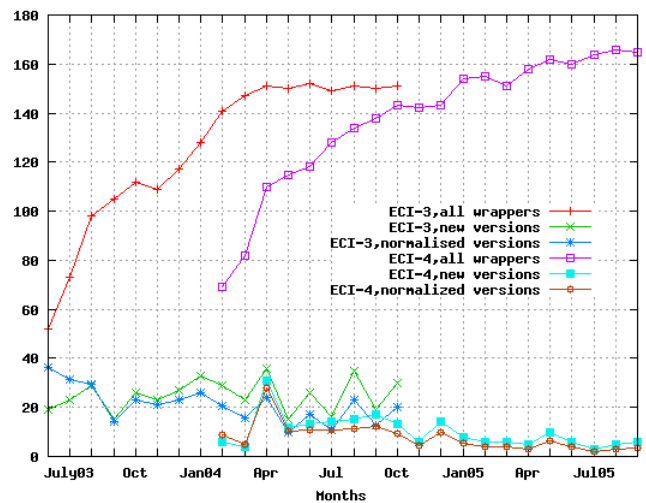


Figure 2: The wrapper set evolution (the total and new versions) for ECI-3 and ECI-4.

6.2 Wrapper components

Out of 214 wrappers deployed at least once with ECI-4 (see Appendix for the full list), we have selected eight top groups with wrappers present in at least 20% of daily tests. The eight selected groups are Aerospace (7 wrappers), Computer (9), General (8), Health (8), Library (4), Regulation (15), News (2)

and Science (11). Below we analyze the wrapper components. We report on the size of forward and backward transducers (the number of rules in R), the average confidence of content views and the problem size (the number of classes in L).

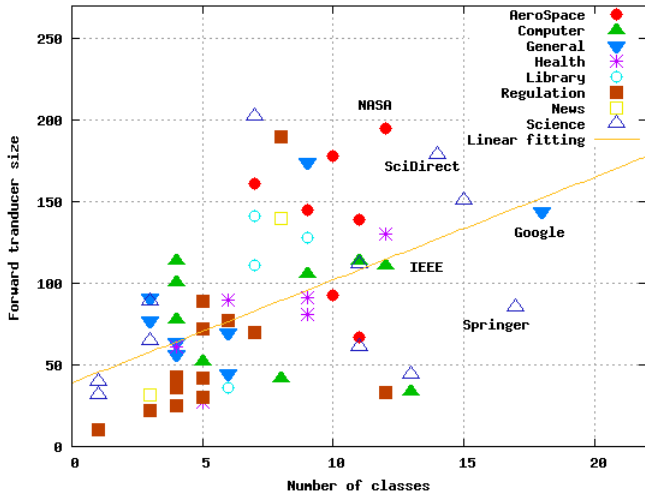


Figure 3: The size of forward transducers versus the number of classes.

Wrappers in selected groups extract data of 1 to 18 classes, with the average of 7.1 classes per wrapper. Forward transducers have 10 to 203 states (see Figure 3), with an average of 85.7 rules per wrapper. For backward transducers, the corresponding numbers are 32, 435 and 109.9. The confidence of content classifiers vary between 71.7% and 100%.

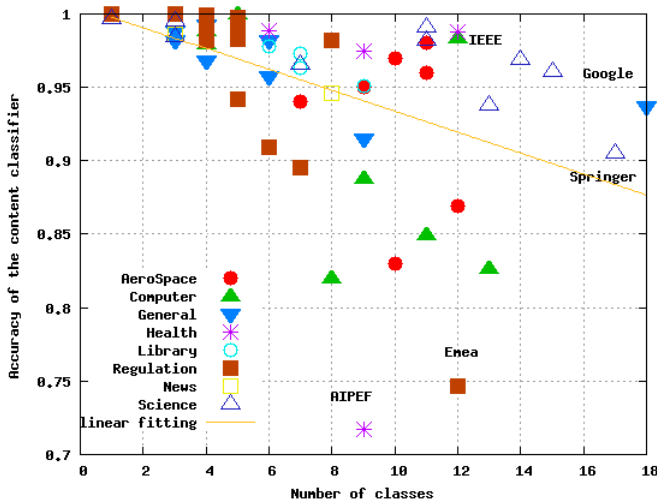


Figure 4: The number of classes versus the confidence of content classifiers.

As Figure 3 suggests, there exists a strong correlation between the transducer size and the class number. The more classes, the more rules are needed to disambiguate the class choice from the context. Similarly, Figure 4 shows that the confidence of content classifiers decreases with the number of classes.

A different look at the wrapper components in Figure 5 confirms the observed phenomenon. It shows that a larger size of a forward

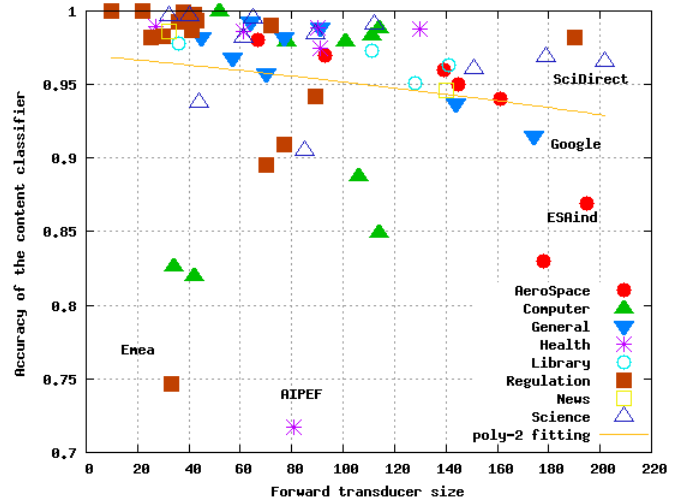


Figure 5: The size of forward transducers versus the confidence of content classifier.

transducer correlates with a lower prediction power from the content, since both views describe a more complex problem. In other words, for small and simple Web sites, if the concept shift breaks the forward transducer, the content classifier and backward transducer have good chances to accomplish the extraction process. Instead, for complex sites, even their joint deployment in the case of concept shift may be still insufficient to achieve the 98% level.

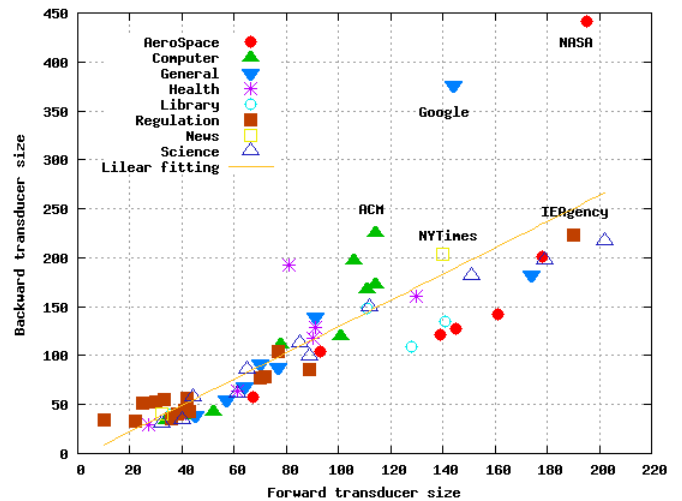


Figure 6: The size of forward and backward transducers.

Figure 6 confirms an expected strong correlation between the forward and backward transducers. However, it appears surprising that backward transducers have on average 30% more rules than forward ones. As a deeper analysis of rules pointed out, many providers present their answers as tuples of variable size. The beginning of tuples are implicitly better managed by the page layout and are therefore easier detectable from the context by a forward transducer. Instead, if scanning the page backward, it appears more difficult and ambiguous to recognize the end of variable tuples. This results in more rules induced for backward transducers.

Finally, Figure 7 shows classes targeted by at least 10% of ECI-4

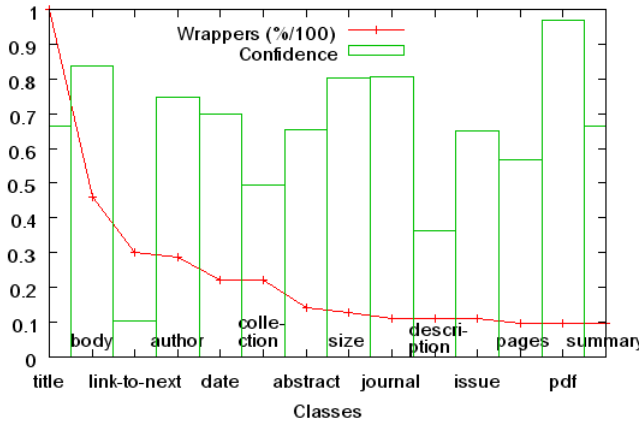


Figure 7: The most frequently used classes and their confidence.

wrappers and their confidence. Class `title` used by all wrappers shows a modest confidence. All wrappers face a hard problem of distinguishing a link to the next page from other links on a page. Instead, some classes like PDF or PS attachments are easily recognizable by corresponding patterns.

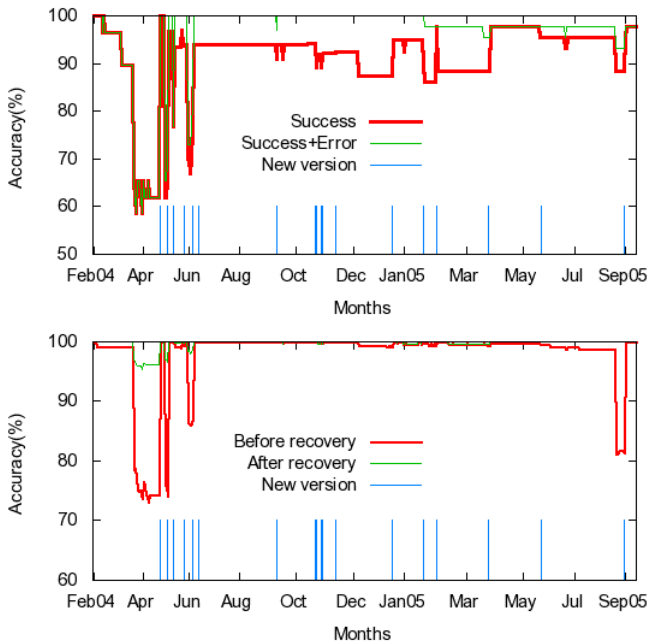


Figure 8: The daily performance of Google wrapper; a) successes, errors and failures; b) basic scan and recovery performance.

6.3 Daily recovery performance

The wrapper recovery procedure brings an important additional value to the wrapper performance, by increasing their robustness in the case of concept shifts and reducing by 65% the number of human interventions in all commercial installations. In this section, we report on the daily performance of ECI-4 wrappers.

We start by analyzing the Google wrapper which suffered the most from concept shifts; it represents by far the largest number

of new versions committed in ECI-4. The initial version was a product of migration from ECI-3, then 18 new versions have been committed since February 2004 to September 2005.

In Figure 8 we plot the day-to-day performance of the Google wrapper. Figure 8.a shows the success, error and failure ratios and Figure 8.b shows the basic scan and recovery performance. In both plots, we trace the emission of new versions.

The analysis shows that the emission of a new version is often caused by a sharp drop in the wrapper performance, in particular, when Google pages underwent important changes in April-June 2004, often invisible to the human eye. Then, for a long period, the top performance of the recovery procedure (superior to 99%) co-existed with an important error ratio (around 8%). So, several new versions have been committed, with a goal of improving the basic transducer and reducing the number of recovery calls and test errors. The last important change took place in September 2005. All wrapper updates are realized in the evolutionary mode, by adding new queries and new annotated pages to the existing training set. Consequently, the number of tests grew from 26 in February 2004 to 39 in September 2005⁴.

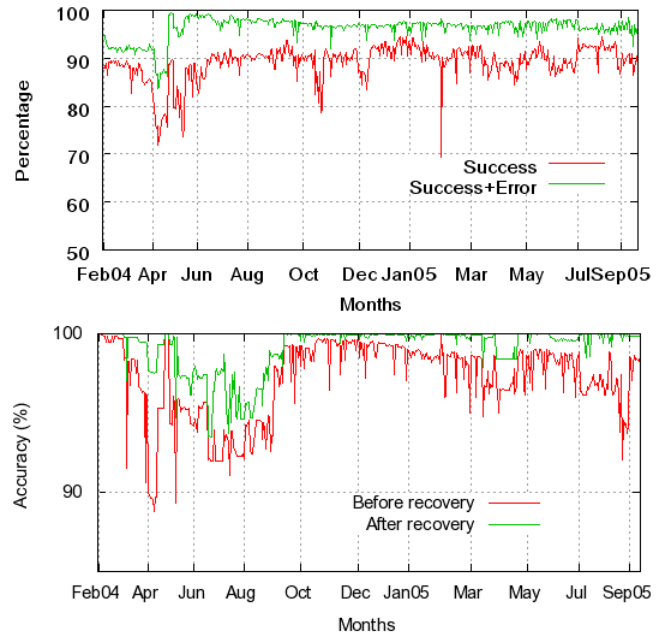


Figure 9: Daily wrapper and recovery performance for ECI-4.

Figure 9 plots the day-to-day performance of 214 wrappers in ECI-4. The average failure ratio is about 3%; it remained stable during the testing period, with an exception of few starting months of migration from the previous version.

The error ratio varies around 7%. The fluctuation of values are characteristic to any open or noisy systems, with several important performance drops at the beginning and at the end of the testing period. In Figure 9.b, we show the accumulative scan and recovery performance, which particularly suffered during the migration period before achieving the optimal performance level.

6.4 Global recovery performance

We conclude the analysis by the global performance of the recovery algorithm. We estimate the impact the recovery has had

⁴During the same period, the number of rules in the transducers increased approximately by 80%.

on the final wrapper accuracy. In Figure 10 we present the two-dimensional grid where x and y axes show five ranges of the wrapper accuracy before (x axis) and after the recovery (y axis). The value in a cell $[x,y]$ indicates the percentage of wrapper tests happened in range x before recovery and in range y after recovery.

Five selected ranges are $[0,90[$, $[90,95[$, $[95,98[$, $[98,100[$ and $[100]$. The largest value (71.2%) goes to the cell $([100],[100])$, where the basis transducer performs well and no recovery was called for. For 11.5% of parses, the extraction was less than 100% but more than 98%, and 93.9% of them have been recovered to 100%.

A similar situation takes place for other ranges. In general, 28.8% parses failed to achieve 100%, and the recovery managed to achieve 100% result for 77.8% of them. Similarly, out of 17.3% of tests that failed to reach the 98% level, the recovery helped 82.6% to achieve 98%, and 67.1% to achieve 100%. Only 4.0% of tests remained under 98% after the recovery.

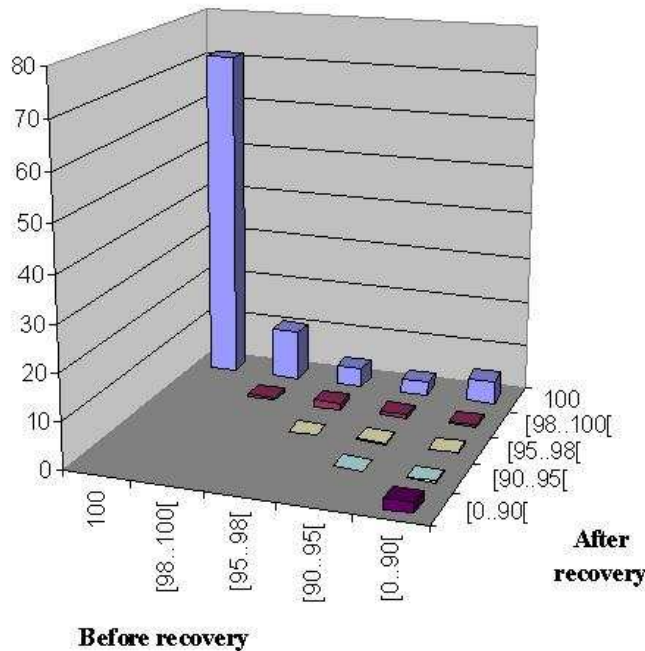


Figure 10: Recovery performance for ECI-4, by ranges.

6.5 False positives and false negatives

A *false negative* (or false alarm) occurs when a wrapper test reports a problem whereas none exists in reality. With an exception of internal causes (see Section 5), a majority of failures are caused by provider access and network traffic problems; they are of a volatile nature and are therefore *false failures*. A false error often occurs due to insufficient wrapper training or a really complex Web site. An unexpected variation in the page structure would run the transducers in error and the content classifier would be unable to recover the extraction process.

A *false positive* occurs when a wrapper test reports a success, whereas the wrapper extracts data incorrectly. If the test fails to detect an important shift, it does not alert the designer. This may happen when a context of a token confuses the transducer and triggers a wrong rule from R which then fits the validation threshold. Luckily, such a combination was rarely observed in ECI-4 and has been managed by adding extra constraints on the wrapper output. The second cause is due to incorrectly annotated sample pages used

for wrapper induction. If, for some reasons, a badly annotated sample is included in the training set, it can generate a rule that extracts wrong results from new pages. This appears particularly dangerous in the automatic wrapper repairing phase, where the recovery manages to complete the extraction from test pages and certifies the newly annotated pages as a new training set for inferring a new wrapper version.

A reasonable trade-off between false positives and false negatives can be achieved by tuning the recovery algorithm and alarm criteria. To ensure the wrapper quality, the ECI deployment policy is made more restrictive with respect to false positives, however at the risk of introducing more false alarms. The ECI deployment policy includes the following rules of thumbs:

Recovery parameters. The basic (forward) transducers appear to be very accurate with few or none false positives. Instead, the predictive power of content classifiers vary considerably over the entire set of wrappers. In order to meet the scalability requirement and to hold one global value $thVal$ and to keep low the false alert level, the decision was made to lower the threshold values initially recommended in [5]. Similarly, after multiple iterations, the recovery threshold value $thRec$ was set to 0.68.

Automatic wrapper repairing. An important decision was made of disallowing the automatic wrapper repairing at the central repository, because of a high risk of false positives. The automatic repairing remains activated at client repositories. If an automatically repaired version fails to correctly accomplish the extraction task, the client can retrieve a better controlled version from the central repository.

7. STATE OF ART

All wrapper techniques generate wrapper instances in manual, semi-automatic or automatic manner. Manual methods use low level mechanisms when extraction rules are embedded directly in HTML parsers [30]. Semi-automatic methods generate wrappers automatically but assume certain assistance from users who select extraction patterns or labeling samples. Automatic methods require no user assistance; they use a specific domain knowledge and apply various heuristics to identify and extract information from pages. Below we briefly describe the semi-automatic and automatic methods and refer interested readers to [1, 6] for recent surveys on methods for information extraction from the Web.

Semi-automatic wrapper generation. These methods assume the user supervision during the generation process. Users do not work with the HTML source directly, but through the page rendering in a browser [2, 3, 26]. Users assist or supervise the process by providing examples or revising patterns. Upon user's suggestions, the wrapper generator comes up with extraction rules which are applied to the HTML source.

Two main groups of methods represent two different type of user assistance required for the wrapper generation. In the first group, users define patterns of extracted information. Systems like Lixto transform selected patterns into internal Datalog-like languages [2, 15]. For the pattern design, Lixto offers an advanced visual interface to define and iteratively refine structural patterns. Once a final pattern is designed, the system transforms it into a set of extraction rules.

The second group exploits the supervised machine learning and induces wrappers from a set of labeled pages. Initially, learning-based wrappers had so called landmark-based structure. In WIEN [24] and STALKER [28], a landmark wrapper combines special *skipTo*-rules with extraction rules which guide the information extraction.

Extraction rules are based on *landmarks* which are groups of consecutive tokens that enable a wrapper to locate the start and end of an item within a page.

SoftMealy [18] extended landmark-based rule to the finite-state transduction mechanism. The expressive power of SoftMealy extractors is higher than WIEN wrapper classes, as it tries to manage various irregularities in pages. SoftMealy transducers may not have arbitrary structure, the user can design this structure of the extraction or adopt a “generic” structure from the SoftMealy extractor library. Once the structure is defined, the system learns the transitions between transducer states in the chosen structure. To get rid of transducer structure design, a method for the wrapper inference from labeled samples that allows any structure of regular transducers was developed in [4]. Finally, since string transducers do not explore the internal structure of HTML pages, [21] proposed a technique for inducing tree automata from sample files.

Automatic wrapper generation. To fully automate the wrapper generation, methods of unsupervised learning have been exploited in [7, 17]. Methods use the domain knowledge and certain assumptions about the page structure, like a tabular [17] or list-like structure [7]. If extracted items have a similar composition, automatic methods generalize items by comparing values of items. However, the accuracy of automatic wrappers remains insufficient to replace the semi-automatic wrapper generation.

Wrapper maintenance. All automatic and semi-automatic methods address the wrapper expressiveness but few of them pay attention to the wrapper robustness and maintenance. Other methods assume they can easily regenerate a wrapper from scratch when needed, without defining how the ‘when needed’ criteria can be managed.

The automatic detection of page changes was first addressed in [23] that proposed a solution that analyzes the page and extracted information and detects the page change with a given accuracy. Knoblock et al. [25] developed a method for wrapper repairing in the case of small mark-up change; it detects the most frequent patterns (like starting or ending words) in labeled strings; these patterns are searched in a page when the wrapper gets broken. [5] tested a number of recovery mechanisms allowing to better control the wrapper recovery performance.

Different commercial and non-commercial products include components to build wrappers for Web sites [11, 12, 14, 20, 22, 26]. [22] represents the most advanced effort on reviewing and comparing functionalities of different wrapper toolkits. Among commercial products, one approach accepted by Fetch Technologies [12], ItemField [20] and Oracle [14] is to provide solutions for integrating and accessing heterogeneous data sources on the top of a commercial product. Another approach is to distribute the software through contracting with a commercial spin-off, like with Lixto Software GmbH [27]. To our best knowledge, no performance analysis is available for any of mentioned products.

8. CONCLUSION

We have presented a detailed analysis of ECI wrapper technology deployed in Documentum ECI commercial installations. ECI wrappers are based on state-of-art techniques from machine learning and grammatical inference. It deploys a multi-view approach to data extraction from Web pages published by content providers. A set of about 1000 daily wrapper reports collected over a period of 28 months has been used to evaluate the performance of ECI wrappers. In our analysis, we address the wrapper adaptability and robustness under concept shifts and we pay a particular attention to the maintenance effort reduction. Additionally, we discuss requirements imposed by the large-scale deployment context. They

discuss the issues of scalability and easy quality control, and how they influence the technology tuning and customization. We quantitatively measure the importance of wrapper architecture components and their impact on the successful and efficient technology deployment.

9. ACKNOWLEDGEMENT

We want to thank all people at XRCE and Documentum who helped us in conducting this performance analysis. In particular, we thank Pierre-Yves Chevalier (EMC Documentum) for his valuable comments and suggestions at all stages of the study.

10. REFERENCES

- [1] A. Laender and B. Ribeiro-Neto and A. da Silva and J. Teixeira. A Brief Survey of Web Data Extraction Tools. *SIGMOD Record*, 31(2), 2002.
- [2] R. Baumgartner, Sergio Flesca, and Georg Gottlob. Visual Web Information Extraction with Lixto. In *Proc. Very Large Database Conference, Rome, Italy*, pages 119–128, 2001.
- [3] D. Bredelet and B. Roustant. Java IWrap: Wrapper Induction by Grammar Learning. Master’s thesis, ENSIMAG, Grenoble, France, 2000.
- [4] B. Chidlovskii. Wrapping web information providers by transducer induction. In *European Conference on Machine Learning, Lecture Notes in Computer Science*, volume 2167, pages 61–73, 2001.
- [5] B. Chidlovskii. Automatic Repairing of Web Wrappers by Combining Redundant Views. In *Proc. of 14th IEEE Intern. Conference On Tools with Artificial Intelligence*, pages 399–406, 2002.
- [6] V. Crescenzi and G. Mecca. Automatic information extraction from large websites. *J. ACM*, 51(5):731–779, 2004.
- [7] V. Crescenzi, G. Mecca, and Paolo Merialdo. RoadRunner: Towards Automatic Data Extraction from Large Web Sites. In *Proc. Very Large Database Systems, Rome, Italy*, pages 109–118, 2001.
- [8] T. G. Dietterich. Machine learning for sequential data: A review. In T. Caelli, editor, *Lecture Notes in Computer Science*. Springer-Verlag, 2002.
- [9] T.G. Dietterich. Ensemble methods in machine learning. In *Multiple Classifier Systems, First International Workshop*, pages 1–15. Springer Verlag, 2000.
- [10] Documentum Services ECI Adapter Library. <http://www.documentum.com/products/glossary/al.htm>.
- [11] Documentum Enterprise Content Integration. <http://www.documentum.com/solutions/eci>.
- [12] Fetch technologies. <http://www.fetch.com/>.
- [13] D. Florescu, A. Y. Levy, and A. O. Mendelzon. Database techniques for the world-wide web: A survey. *SIGMOD Record*, 27(3):59–74, 1998.
- [14] A Primer for Building Portlets Using Oracle Dynamic Services. Oracle Portal Development Kit. <http://portalstudio.oracle.com/pls/ops/docs/>, 2000.
- [15] G. Gottlob, Ch. Koch, R. Baumgartner, M. Herzog, and S. Flesca. The lixto data extraction project: back and forth between theory and practice. In *Proc. 23rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 1–12, New York, NY, USA, 2004. ACM Press.

- [16] A. Hogue and D. Karger. Thresher: automating the unwrapping of semantic content from the world wide web. In *WWW'05: Proceedings of the 14th international conference on World Wide Web*, pages 86–95, New York, NY, USA, 2005. ACM Press.
- [17] Th. W. Hong and K. L. Clark. Using Grammatical Inference to Automate Information Extraction from the Web. In *Proc. 5th European Conf. PKDD, Germany, Freiburg*, volume 2168, pages 216–228. Springer, 2001.
- [18] C.-N. Hsu and C.-C. Chang. Finite-State Transducers for Semi-Structured Text Mining. In *Proceedings of IJCAI-99 Workshop on Text Mining: Foundations, Techniques and Applications*, 1999.
- [19] N. Ireson, F. Ciravegna, M.-E. Califf, A. Lavelli, D. Freitag, and N. Kushmerick. Evaluating machine learning for information extraction. In *Proc. Int. Conf. Machine Learning*, 2005.
- [20] Itemfield. <http://www.itemfield.com/>.
- [21] R. Kosala, J. den Bussche, M. Bruynooghe, and H. Blockeel. Information extraction in structured documents using tree automata induction, 2002.
- [22] S. Kuhlins and R. Tredwell. Toolkits for generating wrappers: A survey of software toolkits for automated data extraction from websites. In *NetObjectDays, NODe 2002, Erfurt, Germany, LNCS*, volume 2591, pages 184 – 198, 2002.
- [23] N. Kushmerick. Regression Testing for Wrapper Maintenance. In *Proc. AAAI*, pages 74–79, 1999.
- [24] N. Kushmerick. Wrapper Induction: Efficiency and Expressiveness. *Artificial Intelligence*, 118:15–68, 2000.
- [25] K. Lerman, S. Minton, and C. A. Knoblock. Wrapper maintenance: A machine learning approach. *Journal of Artif. Intell. Research (JAIR)*, 18:149–181, 2003.
- [26] L. Liu, C. Pu, and W. Han. XWRAP: An XML-Enabled Wrapper Construction System for Web Information Sources. In *Proc. Intern. Conf. Data Engineering*, pages 611–621, 2000.
- [27] Lixto software gmbh. <http://www.lixt.com/>.
- [28] I. Muslea, S. Minton, and C. Knoblock. A Hierarchical Approach to Wrapper Induction. In *Proc. the Third Intern. Conf. on Autonomous Agents Conference, Seattle, WA*, pages 190–197, 1999.
- [29] J. Oncina, P. Garcia, and E. Vidal. Learning subsequential transducers for pattern recognition interpretation. *IEEE Trans. on Pattern Analysis*, 15:448–458, 1993.
- [30] A. Sahuguet and F. Azavant. Building light-weight wrappers for legacy Web data-sources using W4F. *The VLDB Journal*, pages 738–741, 1999.
- [31] Ian H. Witten and Eibe Frank. *Data Mining: Practical machine learning tools and techniques, 2nd Edition*. Morgan Kaufmann, San Francisco, 2005.