Integrating Deep-Web Information Sources[*]

Iñaki Fernández de Viana, Inma Hernandez, Patricia Jiménez,
Carlos R. Rivero, and Hassan A. Sleiman

Abstract. Deep-web information sources are difficult to integrate into automated business processes if they only provide a search form. A wrapping agent is a piece of software that allows a developer to query such information sources without worrying about the details of interacting with such forms. Our goal is to help software engineers construct wrapping agents that interpret queries written in high-level structured languages. We think that this shall definitely help reduce integration costs because this shall relieve developers from the burden of transforming their queries into low-level interactions in an ad-hoc manner. In this paper, we report on our reference framework, delve into the related work, and highlight current research challenges. This is intended to help guide future research efforts in this area.

Keywords: Information, web, integration.

1 Introduction

Our work focuses on deep-web information sources that provide advanced search forms to build search constraints using a number of search fields, e.g., title, author, or price [23]. Our goal is to provide the technology a developer requires to develop agents that can integrate these sources into typical business applications. Such integration is usually addressed by means of wrappers, which are software agents that provide an API that abstracts developers from the details required to simulate a human interacting with a search form.

Iñaki Fernández de Viana · Patricia Jiménez
University of Huelva
e-mail: {i.fviana,patricia.jimenez}@dti.uhu.es

Inma Hernandez · Carlos R. Rivero · Hassan A. Sleiman
University of Sevilla
e-mail: {inmahernandez,carlosrivero,hassansleiman}@us.es

[*] Supported by the European Commission (FEDER), the Spanish and the Andalusian R&D&I programmes (grants TIN2007-64119, P07-TIC-2602, P08-TIC-4100, and TIN2008-04718-E).

Note that typical wrapping agents do not relieve developers from the burden of transforming their queries into a number of actions on the search form. Virtual integration techniques, a.k.a. metasearch, provide a means to increase the abstraction level since they allow to create a unified search form that abstracts away from the details of several related actual forms in a given domain, e.g., flights, hotels, or bibliography [6]. These techniques increase the abstraction level, since developers only need to map their queries onto a unified search form, i.e., they are relieved from the burden of simulating the interaction with actual search forms.

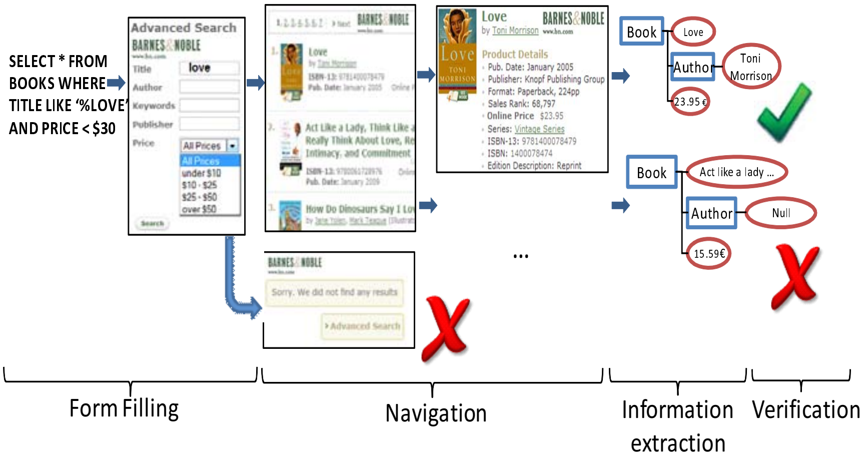

Fig. 1 An overall image of our proposal

Our hypothesis is that the effort might be reduced further if wrapping agents were able to understand higher-level queries written in languages such as SQL or SPARQL. As a motivating example, consider the bookstore scenario in Figure 1, in which a user searches for books that contain "love" in the title and cost less than $30. Answering this query goes through the following: 1) form filling, i.e., the query is analysed to find out how to fill in the search form appropriately; 2) navigation, i.e., the search button is pressed and the resulting page is navigated until pages about books are found; 3) information extraction (IE), i.e., once book pages are retrieved, the information of interest is extracted and structured according to a given ontology; 4) verification, i.e., to check the retrieved information for errors.

In this paper, we report on a proposal that allows to build wrapping agents with the above capabilities. Our goal is not to dive into details, but to provide an overall picture in which the emphasis is on making it explicit what the related work is and what the research challenges are. We expect these analysis to guide future efforts regarding building wrapping agents. In Sections 2–5, we delve into the details concerning each of the previous phases; in Section 6, we present our conclusions.

2 Form Filling

In this phase, the wrapping agent takes a high-level structured query over a deep-web information source as input and it has to translate the query into a number of search forms filled in with the suitable values. The first problem to be addressed is to use a semantic model of the search form, which is not machine-processable.

Existing approaches model a search form using several semantic levels [11]. One level deals with the query capabilities of the search form, i.e., what type of queries are issued to the information source through its search form. The query capabilities of a search form are modelled by parameterised views over the source [26, 27]. Some approaches deal with the automatic extraction of the search form query capabilities. Shu et al. [31] extract them by issuing predefined queries that help detect mandatory fields. Zhang et al. [36] extract hidden database attributes, operators which are applied to these attributes, and their ranges. Attributes are used in conjunctive queries because it is enough to capture a wide range of query capabilities.

When a query is posed over a deep-web information source, it has to be answered using only the views offered by the source. This problem is addressed by techniques for answering queries using views [10], which are based on selecting a number of views to answer a query. Another issue is known as heterogeneities in the predicate level [37], which happens when a query and the source may use different predicates for the same attribute, e.g., the query has a predicate: 'bookPrice $<$ \$15'; and the source accepts: 'bookPrice $<$ \$10' or 'bookPrice between \$10 and \$25' (cf. Figure 1). In this case, it is needed to fill two search forms in, one with 'bookPrice $<$ \$10' and another with 'bookPrice between \$10 and \$25', but a filter is needed in the second one to remove books whose price exceed \$15. To solve this problem, Zhang et al. [37] use a predicate mapping based on data types: depending on the data type of the attribute, there are a number of handlers that solve the heterogeneity problem.

An important aspect in the process of answering queries using views is to analyse the query feasibility, i.e., to study whether a query can be issued without executing it using the search form query capabilities. This analysis avoids a trial and error process in which the user writes a query and executes it until a suitable query is obtained. Petropoulos et al. [27] present a user interface for building SQL queries over a set of parameterised views that warns when a query is not feasible. Pan et al. [26] report on a generic framework for representing query capabilities that analyses the feasibility of SQL queries over deep-web information sources. An implementation of this framework and a recap on its main drawbacks is presented in [30].

After applying the techniques for answering queries using views and obtaining a number of ground views, each ground view can be seen as a search form that is filled in. The next step consists of actually filling each search form in with the values specified in each view. To perform this task, the next semantic level of existing search form models deals with the relations between the search form fields and the attributes [11], e.g., if the source contains an attribute 'publicationDate', this date can correspond to three fields in the search form: the day, the month and the year; these three fields are semantically related with the attribute 'publicationDate'.

3 Navigator

As we saw in the introduction, (cf. Figure 1), the navigation agent is responsible for reaching the pages relevant to the query, discarding or processing any other intermediate or error pages that may appear in the navigation sequence. For example, a no-results page when the information source does not have any information relevant to the query, an error page when a web application server raises and exception, or a disambiguation page in which the user is asked to clarify the query.

Traditional exhaustive crawlers [29] take a blind approach at navigation, following every link in each page. This is useful for certain tasks, like indexing pages for a search engine, but for virtual integration purposes it has an important drawback: usually web pages contain a large number of links, some leading to relevant information, but most having other purposes, like advertising or internal site navigation.

Virtual integration systems retrieve information online, hence system response time should be reasonably fast. Traditional crawlers are not suitable for virtual integration tasks, since crawling every single site means retrieving, analysing and classifying thousands of pages, most of them useless for this task. This results in an increment in cost and time that should be avoided.

As opposed to blind navigation, other approaches include some criteria to decide which are the links that must be visited, therefore reducing the number of irrelevant visited links. Relevancy criteria can be handcrafted by the user or automatically decided by the navigator, with the support of some reasoning process, usually in the form of a classifier [33]. The latter is the focus of our research, as we are interested in following only relevant links and retrieving only relevant pages. This kind of navigation is more efficient and is less costly than traditional crawlers.

Next, we briefly describe and analyse the existing proposals in the Navigation area. All of them define a navigation sequence, but we can distinguish between supervised (recorders) and non-supervised proposals (automated navigators), although we must note that, to the best of our knowledge, there is not a completely non-supervised proposal.

Recorders [1, 2] are the most supervised proposals. They present a simple user interface to record a user's navigation steps through an information source and store them in a file, in order to replay them automatically in future. These proposals interact directly with the web browser interface, so they do not deal with issues like scripts, posting forms, required authentication, or sites that keep session information. They depend completely on the user's knowledge, who is responsible for defining the navigation sequences, providing the values to fill in the forms and redefining the sequences whenever the target web site changes. Recording navigation steps makes navigation inflexible, and error-prone. However, it produces more efficient navigation patterns than traditional crawlers.

User-Defined Navigation [4, 9, 25] is less supervised than recorders, but still the system learns the in a supervised way, i.e., the user demonstrates how to obtain the relevant information, and the system is able to generalise this knowledge.

Automated navigators [3, 19, 21, 33] extract navigation sequences automatically from information sources, instead of learning them from the user. In order to accomplish this, the user has to provide some examples, but in this case the system needs less information than in the former proposals, like one or two examples of the relevant pages to be reached, or the set of keywords identifying them, inter alia.

Our focus is on following the least supervised paradigm. Therefore, our navigator agent crawls every information source. In every step (web page), the agent classifies it, and according to this class, a specifical set of actions is performed, in order to reach another page (e.g., clicking on a link or submitting a form). Once the agent has reached a relevant page, it is stored for latter processing. This is a lightly supervised system because the user only provides a reduced number of examples to train the classifier and to teach the navigator which links to follow and which ones to ignore.

4 Information Extractor

Results provided by the navigation module contain web pages in which data of interest is formatted in HTML. This format is easily interpreted by humans but agents face a difficulty in finding and extracting data of interest from this type of documents. The key is to use information extractors, which are algorithms that extract data of interest from the Web simplifying and reducing costs of the extraction task.

Despite the variety of proposals about IE algorithms, e.g. [8, 13, 12, 16], none of these solutions is universally applicable, and in an integration process, more than one proposal might be used. Besides, effectiveness and efficiency results are rarely comparable since they were obtained over different data sets.

Proposals are usually built using and providing different interfaces and technology, hence side-by-side comparisons are a tedious task since all these proposals should be implemented using same technology and tested over same data sets. Existing studies and surveys, such as [5, 14, 18], use taxonomies and attributes that don't provide any new information about extraction algorithms.

Our solution is an IE framework. IntegraWeb provides and uses a framework where all existing information extractors and perhaps the majority of new IE proposals can work side-by-side and whose results can be compared for each case of use. Here is a brief description of some of the framework components:

- Preprocessor: A large number of IE algorithms preprocess input documents before deploying them for learning or even for extraction. Cleaning the DOM tree, part of speech tagging or even separating DOM trees and removing unnecessary trees using [22] are typical preprocessing techniques.
- TrainingSet: Algorithms that learn rules for extraction need a training set that contains a set of samples marked by a user which are then used to infer extraction rules. Also, a configurable tokeniser should be used to tokenise the input samples.
- Learner: Algorithms that infer extraction rules use a learner. The learner can use a set of preprocessors, and some other utilities such as string or tree alignment classes. Besides, predefined algorithms are provided where user can define his algorithm's policy by implementing some template methods. For example, Branch and Bound algorithm is predefined and user should only define its main methods.

- WorkingSet: It contains an information extractor, input documents and generates a ResultSet. The information extractor is executed over input documents to extract the data of interest and structure it by means of attributes, slots and records.

We believe that an IE framework is the best solution for IntegraWeb since more than one information extractor could be needed depending on the web sites we are integrating. Our framework provides necessary components for developing existing and new IE algorithms. Developing an information extractor using our framework reduces costs since many reusable components can be used and there is no need to start from blank every time a new IE algorithm is developed.

Apart from the framework, we are working on a new algorithm to induce information extractors that is based on FOIL [28], which is a technique to induce first-order rules. FOIL is a climbing algorithm that starts from an empty rule that includes just a clause with a predicate that characterises a piece of information to be extracted and uses a heuristic that is based on a modified version of the information gain criterion to guide the search of clauses that can be used to complete the rule. A typical rule looks as follows: title(X) :- bold(X), link(X), next(X, A), author(A); intuitively, it means that a title in the web page is a piece of text annotated into the html document as bold and hyperlink; and that it is followed by another piece of text that has been previously identified as an author, whose rule also should be defined.

Unfortunately, the search space FOIL explores grow exponentially on the number of predicates available to characterise a piece of information. This motivated us to work on a number of optimisations and heuristics that may help to reduce this space. For instance, to define the semantics of predicates to detect inconsistencies amongst predicates in a rule or to reduce the number of new predicates that are going to be created and evaluated. Further, we use heuristics to define the order of in which predicates are explored by prioritising those that determine the left and right part of the target information (like next and previous) proposed in the extractors of Kushmerick [16] or using feature selection techniques to identify the most useful predicates depending on the studied domain. Therefore we translate the rules obtained on regular expressions that are easily undertandable by a machine.

5 Verifier

When information extractors are composed of extraction rules that rely on HTML land marks, they can only extract information from the same information source where the training was performed. Therefore, if the source rendering changes then the returned data could be incorrect. Unless the information generated by wrapping agents is verified in an automatic way, these data can go unnoticed for the applications using them.

On our analysis of the current literature, we have built a general verification framework composed by the following: Reaper, Assembler and Verification model Builder. The Reaper and the Assembler collaborate to generate a collection of valid result sets. This collection is used to infer a verification model that characterises the features of the correct data. Furthermore, we deal with the possibility of introducing perturbations to generate incorrect result sets: models that are built only on correct data leads to a overgeneralisation problem [34].

Regarding the Verification model Builder, a verification model is a characterisation of a training set that builds on the analysis of a number of features. Features are quantifiable characteristics and their values can be used as a form of evidence to decide if a result is valid or not. Features can be classified along two orthogonal axes: whether they are numeric or categorical, and whether they are applicable to slots or result sets. Numeric features transform slots or result sets into real numbers. The literature [35] reports on many numeric features, so we have grouped into several categories that range from counting the number of slots of a given class to counting attributes of a given class that match a given starting or ending pattern. Categorical features [24, 7] range from patterns that describe the structure of a record to constraints on the values of some attributes.

When models are constructed, a function has to be inferred from the training set, which should be constructed such that for a given feature vector x, an estimate of its quality is obtained, i.e., if this vector is similar to the rest of the training set.

In [20] the training set is characterised by a vector in which every feature is associated with its average value in the training set. In [15, 17] features are modelled as if they were random variables whose Gaussian distributions can be inferred from the training set; thus, to profile the value of a feature on an unverified result set, one can compute the probability that the corresponding random variable takes this value. The technique presented in [24] models every feature as if it was a random variable with a Gaussian distribution, but the profiles are calculated as the probability that a feature might have another value with a higher probability.

There are chances that alarms report false positives. Our approach in this cases is to use sanity checks, e.g., they use the Web as an information source to check if the data that has triggered the alarm is correct but infrequent.

6 Conclusion and Future Work

In this paper, we present a reference framework to build wrapping agents, which are pieces of software that query deep-web information sources. We focus on helping software engineers to construct wrapping agents that accept high-level structured queries. Our reference framework is composed of four phases and we highlight current research challenges in each phase.

In the form filling phase, the research challenges are:

1. Semantic models of the search forms are a cornerstone for the form filling task. In the bibliography, there are some proposals that devise complex semantic models of search forms but they fail on representing search forms using web technologies such as Javascript or AJAX. These technologies make search forms more interactive, i.e., a field can be hidden depending on the value of other field.
2. In the bibliography, a search form is seen as a number of parameterised views over the source, and a query over a deep-web information source is answered by applying the techniques of answering queries using views. These techniques are

mainly based on the relational or XML model but they have to be adapted to the Semantic Web model, where the concept of view is not so well-known.

Regarding the navigation phase, the main challenges are:

1. Response pages have to be classified into the different roles that they play. There are many proposals dealing with the web page classification problems, hence the problem comes down to choosing between one of them.
2. Links leading to relevant pages have to be identified before clicking on them, to avoid visiting useless pages.
3. The navigator should interact with the user as little as possible. Therefore, learning is unsupervised, or at least, very little supervised.
4. Instead of building an ad-hoc navigation model for every site, our focus is on developing a general model that can adapt to most sites just by tuning some parameters, and preferably change-resilient.

Besides mentioned challenges, here are some research challenges concerning IE:

1. The construction of an universal framework where earlier and new proposals can be integrated.
2. A survey that classifies information extractors to compare effectiveness and that uses comparable results over the same data set to compare efficiency.
3. Optimisation and improvement of existing IE algorithms.
4. An universally applicable effective and efficient information extractor.

Last, but not least, the verifying phase presents the following research challenges:

1. The verification modelling techniques described assume that the data sets returned by the reaping plan are homogeneous. To work with truly homogeneous data sets we propose to analyse the training set data and to obtain a series of new data sets, which will be homogeneous. It is interesting to study also the candidate features set before creating the verification model to reduce its size.
2. For the verifier training to be adequate, it is advisable that the training set has both valid and invalid examples. This is a problem as the training set has only valid examples.
3. We cannot assume all features follow a normal distribution and hence we must find techniques that allow modelling without any assumptions of its distribution.
4. The wrapper verification problem is closely related to the novelty recognition problem [32] if it is rephrased in terms of feature vectors and their similarities.

References

1. Anupam, V., et al.: Automating web navigation with the webvcr. Computer Networks 33(1-6) (2000)
2. Baumgartner, R., et al.: Deep web navigation in web data extraction. In: CIMCA/IAWTIC (2005)
3. Blanco, L., et al.: Efficiently locating collections of web pages to wrap. In: WEBIST (2005)

4. Blythe, J., et al.: Information integration for the masses. J. UCS 14(11) (2008)
5. Chang, C.-H., et al.: A survey of web information extraction systems. IEEE Trans. Knowl. Data Eng. 18(10) (2006)
6. Chang, K.C.-C., et al.: Toward large scale integration: Building a metaquerier over databases on the web. In: CIDR (2005)
7. Chidlovskii, B., et al.: Documentum eci self-repairing wrappers: performance analysis. In: SIGMOD Conference (2006)
8. Crescenzi, V., et al.: Roadrunner: Towards automatic data extraction from large web sites (2001)
9. Davulcu, H., et al.: A layered architecture for querying dynamic web content. In: SIGMOD Conference (1999)
10. Halevy, A.Y., et al.: Answering queries using views: A survey. VLDB J. 10(4) (2001)
11. He, H., et al.: Towards deeper understanding of the search interfaces of the deep web. World Wide Web (2007)
12. Hogue, A., Karger, D.R.: Thresher: automating the unwrapping of semantic content from the world wide web. In: WWW (2005)
13. Hsu, C.-N., Dung, M.-T.: Generating finite-state transducers for semi-structured data extraction from the web. Inf. Syst. 23(8) (1998)
14. Jung, K., et al.: Text information extraction in images and video: a survey. Pattern Recognition 37(5) (2004)
15. Kushmerick, N., et al.: Regression testing for wrapper maintenance. In: AAAI/IAAI (1999)
16. Kushmerick, N., et al.: Wrapper induction: Efficiency and expressiveness. Artif. Intell. 118(1-2) (2000)
17. Kushmerick, N., et al.: Wrapper verification. World Wide Web 3(2) (2000)
18. Laender, A.H.F., et al.: A brief survey of web data extraction tools. SIGMOD Record 31(2) (2002)
19. Lage, J.P., et al.: Automatic generation of agents for collecting hidden web pages for data extraction. Data Knowl. Eng. 49(2) (2004)
20. Lerman, K., et al.: Wrapper maintenance: A machine learning approach. Journal of Artificial Intelligence Research 18 (2003)
21. Liddle, S.W., et al.: Extracting data behind web forms. In: Spaccapietra, S., March, S.T., Kambayashi, Y. (eds.) ER 2002. LNCS, vol. 2503. Springer, Heidelberg (2002)
22. Liu, B., et al.: Mining web pages for data records. IEEE Intelligent Systems 19(6) (2004)
23. Madhavan, J., et al.: Harnessing the deep web: Present and future. In: CIDR (2009)
24. McCann, R., et al.: Mapping maintenance for data integration systems. In: VLDB (2005)
25. Montoto, P., et al.: A workflow language for web automation. J. UCS 14(11) (2008)
26. Pan, A., et al.: A model for advanced query capability description in mediator systems. In: ICEIS (2002)
27. Petropoulos, M., et al.: Exporting and interactively querying web service-accessed sources: The clide system. ACM Trans. Database Syst. 32(4) (2007)
28. Quinlan, J.R., et al.: Learning first-order definitions of functions. J. Artif. Intell. Res. (JAIR) 5 (1996)
29. Raghavan, S., Garcia-Molina, H.: Crawling the hidden web. In: VLDB (2001)
30. Rivero, C., et al.: From queries to search forms: an implementation. IJCAT 33(4) (2008)
31. Shu, L., et al.: Querying capability modeling and construction of deep web sources. In: Benatallah, B., Casati, F., Georgakopoulos, D., Bartolini, C., Sadiq, W., Godart, C. (eds.) WISE 2007. LNCS, vol. 4831, pp. 13–25. Springer, Heidelberg (2007)
32. Tax, D.M.J., et al.: One-class classification, concept learning in the absence of counter example. PhD thesis, Delft University of Technology (2001)

33. Vidal, M.L.A., et al.: Structure-based crawling in the hidden web. J. UCS 14(11) (2008)
34. Witten, I.H., Frank, E.: Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations (1999)
35. Wong, T.-L., Lam, W.: Adapting web information extraction knowledge via mining site-invariant and site-dependent features. ACM Trans. Internet Techn. 7(1) (2007)
36. Zhang, Z., et al.: Understanding web query interfaces: Best-effort parsing with hidden syntax. In: SIGMOD Conference (2004)
37. Zhang, Z., et al.: Light-weight domain-based form assistant: Querying web databases on the fly. In: VLDB (2005)