# Towards Discovering Conceptual Models behind Web Sites

Inma Hernández, Carlos R. Rivero, David Ruiz, and Rafael Corchuelo

University of Sevilla, Spain
{inmahernandez,carlosrivero,druiz,corchu}@us.es

**Abstract.** Deep Web sites expose data from a database, whose conceptual model remains hidden. Having access to that model is mandatory to perform several tasks, such as integrating different web sites; extracting information from the web unsupervisedly; or creating ontologies. In this paper, we propose a technique to discover the conceptual model behind a web site in the Deep Web, using a statistical approach to discover relationships between entities. Our proposal is unsupervised, not requiring the user to have expert knowledge; and it does not focus on a single view on the database, instead it integrates all views containing entities and relationships that are exposed in the web site.

**Keywords:** URL Patterns, Conceptual Models, Model Discovery.

## 1 Introduction

The Deep Web comprises a number of web sites that expose data stored in a back-end database, publishing them in a friendly format [8]. Entry points to these web sites are submittable query forms, which return as a response a number of web pages that are generated by filling a template with data [4,11]. The data that fill each template is the result of executing a view over the back-end database [2].

Since query forms are the unique entry points to the Deep Web, the different views that provide the data to fill each template are not accessible. Therefore, the conceptual model of the database, which comprises a number of entities and a number of relationships amongst these entities, remains hidden.

Having access to the conceptual model of a web site is mandatory to perform several tasks, such as integrating different (semantic or non-semantic) web sites [2,14,15], extracting information from the web without supervision [1,7,11], or creating ontologies by means of query forms [16].

As a consequence, there are many proposals in the literature that deal with discovering conceptual models behind web sites [1,2,4,5,6,9,10,11,12,13,16]. Some of these proposals deal with models composed solely of entities, without taking

* Supported by the European Commission (FEDER), the Spanish and the Andalusian R&D&I programmes (grants TIN2007-64119, P07-TIC-2602, P08-TIC-4100, TIN2008-04718-E, TIN2010-21744, TIN2010-09809-E, TIN2010-10811-E, and TIN2010-09988-E).

the relationships between them into account [4,5,6,10,12,13]. Other proposals discover models with entities and relationships [2,16], but they are supervised and require the intervention of the user, providing expert knowledge about each web site. Finally, the rest of the proposals focus on a single template, discovering only one view of the model [1,9,11].

In this paper, we propose a technique to discover the conceptual model behind a web site in the Deep Web. The model our technique is able to discover from each web site does not represent the complete, hidden conceptual model of the back-end database, but the union of the views over that conceptual model, composed of those entities and relationships that are exposed in the web site.

Our technique takes a set of URL patterns as input, each of which represents an entity in a particular web site. It follows a statistical approach to detect relationships between those entities. Our hypothesis is that each relationship is materialised in HTML links that go from pages of one class to pages of another class, so an XPath pattern targeting those links is created to represent each relationship. The URL patterns that support our technique can be either handmade by the user, or automatically built by any of the former proposals [3,6,10,13].

Our proposal presents some advantages: it creates a conceptual model consisting not only of entities, but also of relationships between those entities; it is not supervised, which saves the user a significant amount of time in labelling training sets, and does not require the user to have expert knowledge; and it integrates different views from the different templates in the site. Moreover, our proposal discovers all the possible anonymous relationships in the model, and we leave the user the task of labelling those relationships with an appropriate name and selecting those relationships that are useful for his or her model. Therefore, the set of relationships we automatically discover can be used as a first approach to the model, which can be refined by an expert data modeller, with a significant reduction in time investment [17].

The rest of this article is organised as follows: Section 2 reports on the related work on web site modelling; Section 3 defines our proposal to discover relationships in web sites; Section 4 shows the validation of our technique, using a well-known academical web site; finally, Section 5 lists some of the conclusions drawn from the research and concludes the article.

## 2 Related Work

There are many proposals related to web site modelling in the literature. Some of these proposals deal with models composed solely of entities [4,5,6,10,12,13], while others deal with more complex models including entities and relationships between those entities [1,2,9,11,16]

Models including only entities are usually discovered by web page clustering proposals, which unsupervisedly classify the pages in the web site. This clustering is based in features either from the page content or its structure [17], which implies that the page must be downloaded beforehand [5,12], or from URL features, which prevents having to download it [4,6,10,13]. In the latter case, the result is a collection of URL patterns representing each class. All the former proposals discover models of web sites that are exclusively composed of entities, but none of them discovers relationships amongst those entities.

Other proposals deal with models composed of both entities and the relationships amongst those entities. These proposals are usually focused on web information extraction, since extractors require such a model, that can be either provided by the user (supervised proposals) [2,16], or automatically inferred after analysing the pages of the web site (unsupervised proposals) [1,9,11].

On one hand, supervised proposals rely on the user to define the model. Tao et al. [16] analysed the problem of learning ontologies from web sites. Their proposal, FOCIH, consists of providing the user with a wizard-like application to design the model, and annotate pages from the web site according to that model. From that annotations, FOCIH infers an ontology, composed of concepts and relationship between the concepts. Atzeni et al. [2] proposed the Araneus Data Model, which defines a user-generated model for each web site that describes the different views of the schema of the web site, including the different entities and relationships. Supervised proposals require the user to have both the expert knowledge about each site to model it, and the expertise in data modelling to create a good model from scratch.

On the other hand, unsupervised proposals infer a model from the analysis of the web site. Kayed et al.[11] proposed FivaTech, a technique to discover the model behind a template, by analysing the DOM tree of a reduced set of web pages generated from that schema. Crescenzi and Mecca[9] proposed RoadRunner, an information extractor which automatically discovers the model behind one template in a web site, and uses this model to extract information. Finally, Arasu and Garcia-Molina [1] proposed EXALG, an information extractor based on grammar inference. The former proposals only discover the model behind one single template in the site, although web sites are usually composed of several templates, one for each type of information it offers. Therefore, each template allows discovering one different view on the back-end database, and all views should be integrated to infer a single conceptual model.

## 3 Proposal

Our technique takes a set of URL patterns that describe all classes of information offered in a web site as input, and discovers relationships between the classes. In the following subsections, we first introduce a running example, then we define some concepts that support our technique, and finally we describe the technique.

### 3.1 Running Example: Microsoft Academic Search

Microsoft Academic Search (from now onwards, MsAcademic) is an scholarly web site that offers different classes of information about academic publications (authors, papers, publishing hosts, such as journals or conferences, and research keywords, amongst others). Also, relationships between these classes of information are offered as well, e.g., author pages include a list of papers written by that author, and also a list of papers that cite this author. Furthermore, for each of the former papers, they offer the list of co-authors, the host it was published in, as well as the citations of the paper.

For the sake of simplicity, in this paper we focus on classes *Paper*, *Author*, *Journal*, *Conference* and *Citation*, and the relationships amongst them. An analysis of the MsAcademic site by the pattern building proposal in [10] yields the following URL patterns:

- $p_1 = \langle$ http, academic.research.microsoft.com, Publication, $\star$, $\star\rangle$,
- $p_2 = \langle$ http, academic.research.microsoft.com, Author, $\star$, $\star\rangle$,
- $p_3 = \langle$ http, academic.research.microsoft.com, Journal, $\star$, $\star\rangle$
- $p_4 = \langle$ http, academic.research.microsoft.com, Conference, $\star$, $\star\rangle$
- $p_5 = \langle$ http, academic.research.microsoft.com, Detail, eT, 1, sT, 5, id, $\star\rangle$

For example, pattern $p_4$ matches all URLs in MsAcademic containing information about conferences (e.g., URL `http://academic.research.microsoft.com/Conference/195/er` contains information about the ER conference).

## 3.2 Preliminaries

**Definition 1 (Tokenisation).** *Let $s$ be a string, we define $\tau(s) = \langle s_1, s_2, \ldots, s_n\rangle$ as the sequence of tokens that is obtained after tokenising $s$.*

Note that this definition is applicable to both URLs and XPath locators.

**Definition 2 (Pattern).** *We define a pattern as a sequence of tokens, such that some of the tokens are literals, whereas others are wildcards. We denote a pattern $p$ as $p = \langle t_1, t_2, \ldots, t_m\rangle$. We distinguish between URL patterns and XPath patterns. The latter are class-dependent, since XPath expressions are calculated in the context of a given page of a certain class $c$.*

We represent patterns by means of a subset of regular expressions that includes only literals and wildcard expressions. A wildcard is represented with symbol $\star$, and it represents any sequence of characters, excluding token separators defined for a particular tokenisation. Next, we define the problem of finding a match of a given pattern in another string.

**Definition 3 (Pattern Matching).** *Let $p$ be a pattern $p = \langle p_1, p_2, \ldots, p_l\rangle$, and $s$ be a sequence of tokens $s = \langle s_1, s_2, \ldots, s_l\rangle$, both of length $l$. We define that $s$ matches $p$, and we denote it as $s \sim p$ iff each token in $p$ is either a wildcard, or it is equal to the correspondent token in $s$.*

Note that $p$ can be either a URL or XPath pattern, and that both URLs and XPath expressions are strings, hence we can apply the matching predicate on both URLs and XPaths.

Let $\mathcal{P} = \{p_1, p_2, \ldots, p_n\}$ be a set of URL patterns obtained from website $W$. Each of those patterns, according to the labels assigned by the user, corresponds to some class, e.g., pattern $p_1$ corresponds to class *Paper*, pattern $p_2$ corresponds to class *Author*, and so on.

**Definition 4 (Class-Pattern Correspondence).** *Let $\mathcal{C}$ be the set of classes of information offered by a website $W$, and $\mathcal{P}$ be the collection of URL patterns obtained from $W$. We define the injective function $\Phi : \mathcal{C} \to \mathcal{P}$, which assigns to each class in $\mathcal{C}$ the different patterns from $\mathcal{P}$ that have been labelled as corresponding to that class by the user.*

For example, after obtaining URL patterns $p_1$, $p_2$, $p_3$, $p_4$ and $p_5$ in MsAcademic, we assign a label to each pattern, e.g., stating that $\Phi(Paper) = p_1$, $\Phi(Author) = p_2$, $\Phi(Journal) = p_3$, $\Phi(Conference) = p_4$ and $\Phi(Citation) = p_5$.

**Definition 5 (Detail Page Set).** *Let $\mathcal{C}$ be the set of classes of information offered by a website $W$. We define the set of detail pages of any class $c \in \mathcal{C}$, and we denote it by $D^c$ as the set of pages in $W$ containing information of type $c$.*

Note that the $D^c$ of a class $c$ is composed of those pages whose URLs match the patterns that have been labelled by the user as corresponding to that page.

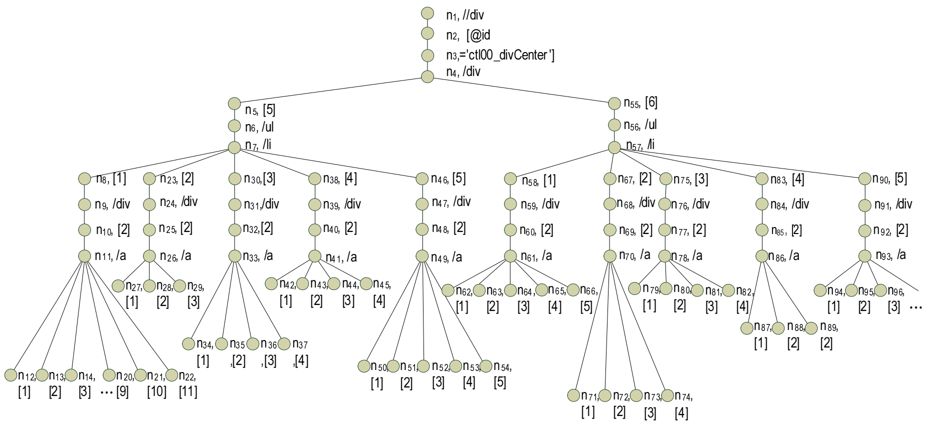
**Definition 6 (Locator).** *Let $w$ be a web page, and $u \in w$ be a URL. We define the locator of URL $u$ in $w$, and we denote it by $XPath(u, w)$ as the XPath expression that points at the position of $u$ in $w$.*

We use a tree notation to represent tokenisations of XPath expressions based on the PATRICIA trees (XPathTree), which allows representing large collections of strings efficiently and compactly. Every node in an XPathTree is an XPathTreeNode, defined by a label $n_i$, and it refers to a token $t_j$. Note that each path from the tree root to a leaf represents a single XPath. An example of a tree containing XPath expressions is presented in Figure 1a. For the sake of readability, each token in each node is preceded by the character that separates it from the previous token.

## 3.3 Relationships Discovery

We base the discovery of relationships between two classes on the detection of HTML links in pages of one class whose target is a page of another class. We extract the XPath locators of those links, and we apply a statistical-based technique to estimate the variability of each token in each locator. Then, we abstract the tokens with a high variability (again, using a statistical criterion), creating XPath patterns. Finally, each XPath pattern represents a particular relationship between the former classes.

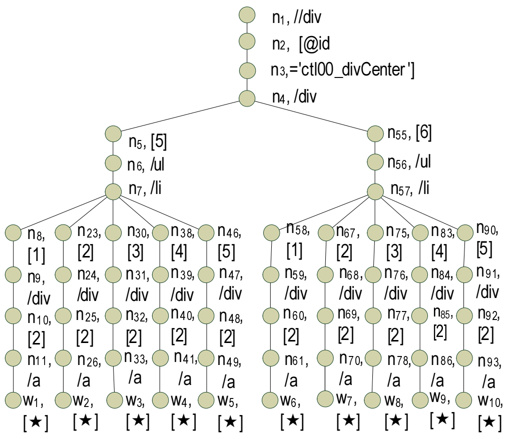
There is an abstract relationship between two classes $a$ and $b$, if there are links to pages of class $b$ in most pages of class $a$. However, more than one type of relationship may exist between any given pair of classes $a$ and $b$. For example, pages of class *Author* in MsAcademic contain both a list of publications, which include coauthors of the publication, and a list of citations, which includes authors that cited this author, as shown in Figure 2. Therefore, there are two different types of relationships in this model between class *Author* and itself: 1) *isCoauthorOf* and 2) *cites*.



(a) XPathTree containing the XPaths of URLs of class *Author* in pages of class *Author*

| node | token | Ω | V(ni) |
|---|---|---|---|
| n11 | /a | {11,3,3} | 4.62 |
| n26 | /a | {3,4,3} | 0.58 |
| n33 | /a | {4,3,3} | 0.58 |
| n41 | /a | {2,4,1} | 1.53 |
| n49 | /a | {1,5,2} | 2.08 |
| n61 | /a | {4,5,5} | 0.58 |
| n70 | /a | {4,3,4} | 0.58 |
| n78 | /a | {3,4,3} | 0.58 |
| n86 | /a | {3,3,2} | 0.58 |
| n93 | /a | {4,5,8} | 2.08 |

(b) $V$ function values

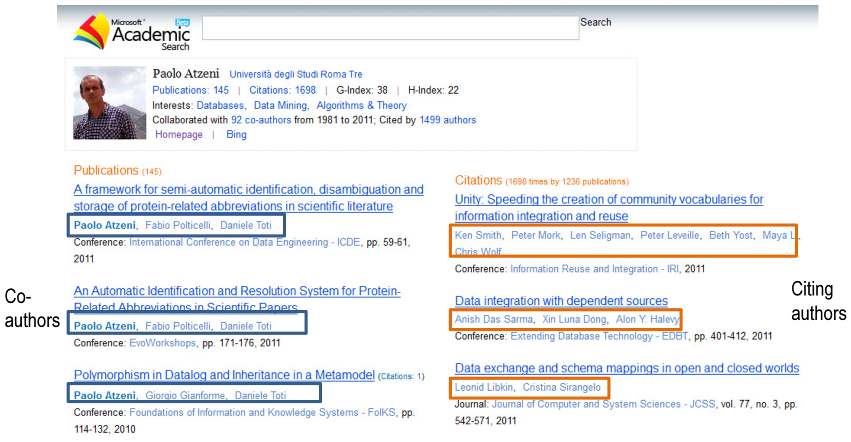


(c) XPathTree, after compression

**Fig. 1.** Representation of the technique, which compresses an XPathTree by abstracting children of nodes with a high variability

Using only URL patterns, we are not able to discern between these different types of relationships, since all URLs match the same pattern, regardless of the type of relationship they represent. Therefore, other features must be extracted from the URLs to classify them according to their role.

We assume that links whose URL matches the same URL patterns may appear in different locations in the page, but all links representing the same relationship appear in similar locations. Therefore, we use the XPath of the different links, which denotes their location in the page. We apply a technique to build patterns for those XPath, which starts by tokenising all XPath locators and inserting all their tokens in order in an XPathTree. Then, we use some criterion to discern tokens that must be abstracted (replaced by a wildcard), based on the concept of variability of a token.



Paolo Atzeni Università degli Studi Roma Tre
Publications: 145 | Citations: 1698 | G-Index: 38 | H-Index: 22
Interests: Databases, Data Mining, Algorithms & Theory
Collaborated with 92 co-authors from 1981 to 2011; Cited by 1499 authors
Homepage | Bing

Publications (145)

A framework for semi-automatic identification, disambiguation and storage of protein-related abbreviations in scientific literature
Paolo Atzeni, Fabio Polticelli, Daniele Toti
Conference: International Conference on Data Engineering - ICDE, pp. 59-61, 2011

An Automatic Identification and Resolution System for Protein-Related Abbreviations in Scientific Papers
Paolo Atzeni, Fabio Polticelli, Daniele Toti
Conference: EvoWorkshops, pp. 171-176, 2011

Polymorphism in Datalog and Inheritance in a Metamodel (Citations: 1)
Paolo Atzeni, Giorgio Gianforme, Daniele Toti
Conference: Foundations of Information and Knowledge Systems - FoIKS, pp. 114-132, 2010

Citations (1698 times by 1236 publications)

Unity: Speeding the creation of community vocabularies for information integration and reuse
Ken Smith, Peter Mork, Len Seligman, Peter Leveille, Beth Yost, Maya L, Chris Wolf
Conference: Information Reuse and Integration - IRI, 2011

Data integration with dependent sources
Anish Das Sarma, Xin Luna Dong, Alon Y. Halevy
Conference: Extending Database Technology - EDBT, pp. 401-412, 2011

Data exchange and schema mappings in open and closed worlds
Leonid Libkin, Cristina Sirangelo
Journal: Journal of Computer and System Sciences - JCSS, vol. 77, no. 3, pp. 542-571, 2011

**Fig. 2.** Detail page of class *Author*

The variability of a token refers to how spread the numbers of tokens that follow that token in different XPath locators in different pages of the same class (i.e., the different numbers of children of the node representing that token in each page) are. Since we do not analyse all pages of a site, but only a representative sample, we estimate the variability by means of the following definition.

**Definition 7 (Variability Estimator:).** *Let $D^c$ be a set of detail pages of class $c$, $x$ an XPath expression and $n_i$ be a tree node referring a token $t$, we define the variability estimator of node $n_i$, and we denote it as $V(n)$ as the standard deviation of the numbers of children of node $n_i$ in the different pages of $D^c$.*

Based on these variability estimators, we define a process to generate XPath patterns. For each node $n_i$ in the XPathTree, we check if its variability estimator is significatively high, and in that case, all its children nodes have their token replaced with a wildcard, and the subtrees rooted at them are merged. Contrarily, children of nodes with a low variability are probably part of a pattern, so they are not abstracted, but kept as literals.

Our technique to mine relationships between classes $a$ and $b$ consists of two steps: XPathTree building and XPathTree compressing.

In the first step, we extract all URLs matching pattern $\Phi(b)$ in pages from $D^a$, and we calculate an XPath locator for each of them. XPath locators are tokenised, and each token is inserted in an XPathTree as a node with a variability estimator. An example of an XPathTree built using this technique is presented in Figure 1a. It contains XPath expressions of URLs matching $\Phi(Author)$ in the running example, extracted from detail pages of class *Author*.

In the second step, we apply a compressing algorithm that performs a depth-first traversal on the XPathTree, and for each visited node, uses its variability estimator to discern nodes with a variability higher than a given parameter $\theta > 0$. Those nodes have their token abstracted into a wildcard ($\star$). As an example, nodes with variability higher than 0.5 are presented in Figure 1b.

After the whole tree has been traversed and processed, each of the resulting tree branches represents a different pattern. Furthermore, each pattern refers to a different type of relationship between classes $a$ and $b$. As an example, in Figure 1c we show the example tree containing XPath expressions of links between class *Author* and itself, after processing all its nodes. The tree contains ten branches, which correspond to ten XPath patterns.

At the end of this process, for each pair of classes $a$ and $b$, we have obtained a set of XPath patterns, that represent the different relationships between them. These relationships are anonymous, and it is left to the user the task of labelling them with an appropiate name. Moreover, we have identified all the possible relationships, but some of them might be duplicated (i.e., we discover a relationship between $a$ and $b$, which is the same as another relationship between $b$ and $a$). Therefore, the user has the opportunity to select the relationships that are most suitable for his or her model, discarding the rest. Therefore, although we are indeed automatically discovering the relationships between entities, the user still has the complete control over the final model.

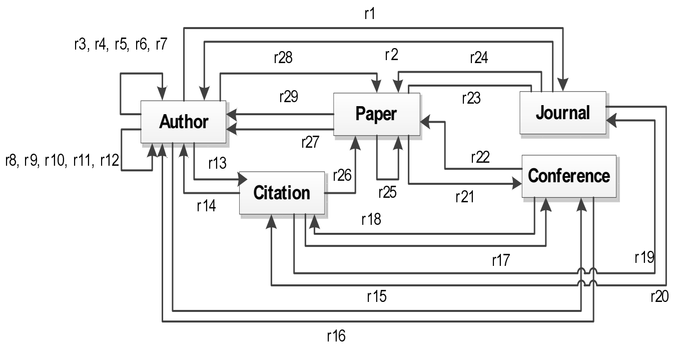
As an example, consider the former patterns discovered in Figure 1c. The first five patterns correspond to links to authors that co-author, respectively, the five most recent papers of an author. Meanwhile, patterns sixth to tenth correspond to links to authors of, respectively, the five most recent papers that cite the author. Therefore, the five first patterns correspond to a particular relationship between class *Author* and itself($isCoauthorOf$), while the five last patterns correspond to a different relationship ($cites$).

## 4 Validation

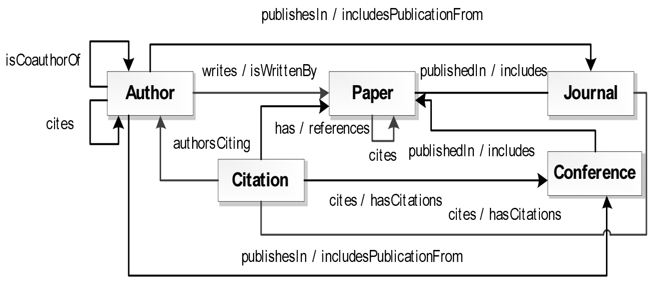
We present an experiment to validate our technique. Microsoft Academic Search was analysed to discover the conceptual model behind it, by means of two steps: in the first step, we discovered the entities in the model, using the URL patterns obtained with the technique described in [10]; in the second step we discovered the relationships between these entities, with the former URL patterns as input, and using the technique described in this paper.

We show the relationships discovered for this site in Figure 3a, using a UML class diagram. After the intervention from the user, a possible model obtained from the former relationships is presented in Figure 3b. For example, relationships $r3$, $r4$, $r5$, $r6$ and $r7$, represent respectively the co-authors of the most recent paper of an author, the co-authors of the second most recent paper, and so on. The user analyses these relationships and decides that all these relationships are actually the same, and labels it $isCoauthorOf$.

Using our technique, it is also possible to infer hierarchical relationships between classes, by identifying classes that share a common group of relationship with other classes. For example, in the former example model for MsAcademic, classes $Journal$ and $Conference$ both share exactly the same types of relationships ($Journal$ is related to *Paper* by means of r23 and r24, to *Author* by means of r1 and r2 and to *Citations* by means of r19 and r20. Similarly, $Conference$



(a) Relationships for MsAcademic



(b) Model for MsAcademic

**Fig. 3.** Model discovered for the validation site

is related to the same set of classes, with two relationships with each class). Therefore, our technique proposes the user the generalisation of *Journal* and *Conference* into another class, and lets the user name it (e.g., *Host*).

## 5 Conclusions

In this paper, we present a technique to discover the conceptual model behind a web site in the Deep Web. Using a set of URL patterns as input, we use a statistical approach to discover all the different relationships between those entities. These relationships can be later analysed by the user, who is responsible for labelling them appropriately, and selecting those relationships that are useful for his or her particular model. We validate our proposal using a well-known academical web site, Microsoft Academic Search.

Other proposals have dealt with the problem of discovering the model behind a web site. Some of them discover models composed only of entities, neglecting the discovery of relationships, which we deal with. Others are supervised, which require expert knowledge from the user, while our technique is completely unsupervised. Finally, other proposals discover only one view of the conceptual model, which corresponds to a particular template; contrarily, we discover a model composed of the union of all views over the complete model that include entities and relationships that are exposed in the web site.

## References

1. Arasu, A., Garcia-Molina, H.: Extracting structured data from web pages. In: SIGMOD, pp. 337–348 (2003)
2. Atzeni, P., Mecca, G., Merialdo, P.: Managing web-based data: Database models and transformations. IEEE Internet Computing 6(4), 33–37 (2002)
3. Bar-Yossef, Z., Keidar, I., Schonfeld, U.: Do not crawl in the dust: different URLs with similar text. In: WWW, pp. 111–120. ACM (2007)
4. Blanco, L., Bronzi, M., Crescenzi, V., Merialdo, P., Papotti, P.: Automatically building probabilistic databases from the Web. In: WWW, pp. 185–188 (2011)
5. Blanco, L., Crescenzi, V., Merialdo, P.: Structure and semantics of Data-Intensive Web pages: An experimental study on their relationships. J. UCS 14(11), 1877–1892 (2008)
6. Blanco, L., Dalvi, N., Machanavajjhala, A.: Highly efficient algorithms for structural clustering of large websites. In: WWW, pp. 437–446. ACM (2011)
7. Chang, C.-H., Kayed, M., Girgis, M.R., Shaalan, K.F.: A survey of web information extraction systems. IEEE TKDE 18(10), 1411–1428 (2006)
8. Chang, K.C.-C., He, B., Li, C., Patel, M., Zhang, Z.: Structured Databases on the Web: Observations and Implications. SIGMOD Record 33(3), 61–70 (2004)
9. Crescenzi, V., Mecca, G.: Automatic information extraction from large websites. J. ACM 51(5), 731–779 (2004)
10. Hernández, I., Rivero, C.R., Ruiz, D., Corchuelo, R.: A statistical approach to URL-based web page clustering. In: WWW, pp. 525–526 (2012)
11. Kayed, M., Chang, C.-H.: Fivatech: Page-level web data extraction from template pages. IEEE Trans. Knowl. Data Eng. 22(2), 249–263 (2010)
12. Mecca, G., Raunich, S., Pappalardo, A.: A new algorithm for clustering search results. Data Knowl. Eng. 62(3), 504–522 (2007)
13. Deepak, P., Khemani, D.: Unsupervised learning from URL corpora. In: COMAD, pp. 128–139 (2006)
14. Popa, L., Velegrakis, Y., Miller, R.J., Hernández, M.A., Fagin, R.: Translating web data. In: VLDB, pp. 598–609 (2002)
15. Rivero, C.R., Hernández, I., Ruiz, D., Corchuelo, R.: Generating SPARQL Executable Mappings to Integrate Ontologies. In: Jeusfeld, M., Delcambre, L., Ling, T.-W. (eds.) ER 2011. LNCS, vol. 6998, pp. 118–131. Springer, Heidelberg (2011)
16. Tao, C., Embley, D.W., Liddle, S.W.: FOCIH: Form-Based Ontology Creation and Information Harvesting. In: Laender, A.H.F., Castano, S., Dayal, U., Casati, F., de Oliveira, J.P.M. (eds.) ER 2009. LNCS, vol. 5829, pp. 346–359. Springer, Heidelberg (2009)
17. Thonggoom, O., Song, I.-Y., An, Y.: Semi-automatic Conceptual Data Modeling Using Entity and Relationship Instance Repositories. In: Jeusfeld, M., Delcambre, L., Ling, T.-W. (eds.) ER 2011. LNCS, vol. 6998, pp. 219–232. Springer, Heidelberg (2011)